Universidade Federal da Bahia
Instituto de Matemática

Programa Multiinstitucional de Pós-graduação em Ciência da Computação

EDUCAÇÃO EM ENGENHARIA DE SOFTWARE COM A ADOÇÃO DE PROJETOS DE CÓDIGO ABERTO: UMA ANÁLISE DETALHADA

Debora Maria Coelho Nascimento

TESE DE DOUTORADO

Salvador
14 de julho de 2017

DEBORA MARIA COELHO NASCIMENTO

EDUCAÇÃO EM ENGENHARIA DE SOFTWARE COM A ADOÇÃO DE PROJETOS DE CÓDIGO ABERTO: UMA ANÁLISE DETALHADA

Esta Tese de Doutorado foi apresentada ao Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Roberto Almeida Bittencourt
Co-orientadora: Profa. Dra. Christina von Flach Garcia Chavez

Salvador
14 de julho de 2017

Sistema de Bibliotecas - UFBA

Nascimento, Debora Maria Coelho (usado em CITACOES).

Educação em Engenharia de Software com a Adoção de Projetos de Código Aberto: Uma Análise Detalhada / Debora Maria Coelho Nascimento – Salvador, 2017.

396p.: il.

Orientador: Prof. Dr. Prof. Dr. Roberto Almeida Bittencourt.

Co-orientadora: Prof. Dr. Profa. Dra. Christina von Flach Garcia Chavez.

Tese (Doutorado) – Universidade Federal da Bahia, Instituto de Matemática, 2017.

1. Educação em Computação. 2. Educação em Engenharia de Software. 3. Projetos Open Source. I. Bittencourt, Roberto Almeida. II. Chavez, Christina von Flach Garcia. III. Universidade Federal da Bahia. Instituto de Matemática. IV. Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

TERMO DE APROVAÇÃO

DEBORA MARIA COELHO NASCIMENTO

EDUCAÇÃO EM ENGENHARIA DE SOFTWARE COM A ADOÇÃO DE PROJETOS DE CÓDIGO ABERTO: UMA ANÁLISE DETALHADA

Esta Tese de Doutorado foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa Multi-institucional de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 14 de Julho de 2017

Prof. Dr. Roberto Almeida Bittencourt
PMCC - Universidade Estadual de Feira de Santana

Prof. Dr. Cláudio Nogueira Sant'Anna
PMCC - Universidade Federal da Bahia

Profa. Dra. Tayana Uchôa Conte
Universidade Federal do Amazonas

Prof. Dr. Dalton Dario Serey Guerrero
Universidade Federal de Campina Grande

Prof. Dr. Sandro Santos Andrade
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

Aos meus pais, Paulo e Frassinetti, sempre presentes em meu coração. À Adriana, minha alma gêmea.

AGRADECIMENTOS

"Eu pensei que poderia viver por mim mesmo, eu pensei que as coisas do mundo não iriam me derrubar , o orgulho tomou conta do meu ser (...) tudo é do Pai, toda honra e toda a glória, é Dele a vitória alcançada em minha vida"!!

Desde o início... desde o primeiro dia que fui para Salvador (quero dizer, há mais de cinco anos), eu sonho com este momento - escrever os agradecimentos deste trabalho. Isto porque ele significa que eu consegui chegar ao final de mais uma etapa de minha vida. Este momento foi sonhado, esperado, desejado... mas por muitas vezes pareceu que seria impossível alcançá-lo. Faltou fé, confiança, esperança... apesar de a cada situação, Deus mostrar que era isso que ele queria para mim. Dele apresentar soluções, caminhos, colocar pessoas para me ajudar. Eu fraquejei... chorei... me desesperei e percebi que minha fé é menor que um grão de mostarda. Mas da mesma forma que Moisés orou com os braços levantados para que Josué vencesse a batalha pela terra prometida, eu tinha pessoas muito queridas orando por mim: Adriana, tia Neóbio, tia Flosceli, tia Mena (infelizmente agora não posso lhe contar isso pelo telefone), tia Bá, tia Zi, Tânia, Eliane, Dona Nádia, Conceição... todas a quem eu pedi orações. Enquanto eu vacilava na fé, elas estavam com os braços levantados, orando para que eu enfrentasse a batalha. Então, o que posso dizer agora é o meu muito obrigado.

Sei que papai e mamãe também estavam lá no céu intercedendo por mim. Pai, atendi o seu desejo de fazer o doutorado. Faltaria escrever um livro... mas para este seu pedido, podemos considerar que pelo tamanho da tese, ela já é um livro, né? Para mamãe, eu justifico que fazer o doutorado era exigência de minha profissão, mas sim, a senhora estava certa, com a sensibilidade que sempre teve, foi ainda mais difícil de que foi o mestrado. A vocês, eu devo tudo em minha vida, tudo o que sou, tudo o que aprendi e todas as vitórias que conquistei. Vocês foram pais maravilhosos!!!

Adriana, você que agora é meu único porto seguro, como posso agradecer tudo o que fez e tem feito por mim? No aspecto material, você providenciou tudo aquilo de que eu precisava e estava ao seu alcance. No emocional, você me confortava, dava forças, coragem, carinho. No espiritual, orava por mim. Quantas vezes liguei para o seu trabalho chorando, pedindo para você rezar por mim e a sua oração me acalmava?? Não consigo expressar, em palavras, o quanto lhe sou grata. Vamos voltar a curtir a vida, passeando no shopping, andando de bicicleta, viajando!!!

Ao prof. Roberto, agradeço primeiramente por ter aceitado ser meu orientador e depois, pela compreensão e paciência por entender o meu ritmo de trabalho (devagar, quase parando), sem grandes produções. Também por ter procurado me animar em algumas situações, escutado alguns desabafos e já ia esquecendo, das correções de português e inglês. Tenho que admitir que sempre fui fraca nestas disciplinas...

À profa. Christina, agradeço por ter inicialmente me acolhido no programa de pós-graduação como sua orientanda, por ter acreditado em mim entre tantos candidatos. Entendi quando quis passar a bola para Roberto e obrigada por o ter convidado. Vocês foram orientadores com os quais eu me senti à vontade. Eu adoro pessoas inteligentes, competentes e ao mesmo tempo, simples.

Dentre outras pessoas que Deus colocou em meu caminho, para me ajudar na execução das atividades que eram necessárias, não poderia esquecer de agradecer ao prof. José Henrique, que permitiu que eu realizasse os estudos de caso ao longo de suas disciplinas e que me substituiu no tempo que foi necessário para que eu terminasse a tese. Ao 'anjo' Gabriel (meu aluno de TCC) que me ajudou com o código do projeto de código aberto utilizado. Ao meu ex-aluno e agora amigo, Wendell, que me ajudou com a formatação Latex e na realização do mapeamento sistemático. Como revisores do mapeamento, também agradeço à profa. Kenia e a outro ex-aluno Thiago. Aos meus colegas, prof. Marcos Dósea, que perdeu um bom tempo comigo discutindo os objetivos de aprendizagem em engenharia de software e à profa. Leila, pelas orientações informais na época da qualificação. Já estava esquecendo... obrigada aos alunos da disciplina de Evolução e Manutenção de Software (2013-2) na UEFS e das disciplinas de DS I e DS III (2015-2) na UFS, principalmente àqueles que voluntariamente responderam aos questionários e participaram das entrevistas.

Institucionalmente agradeço a todos que compõem o Departamento de Computação da UFS , aos meus colegas professores e técnicos administrativos (em especial: prof. Admilson, prof. Leonardo, prof.Alberto, prof. Sérgio, prof. Monteiro, Erickson, Elaine) que me ajudaram com o atendimento às necessidades burocráticas e administrativas, referentes ao meu afastamento. Ao pessoal do DICADT, também pelo mesmo motivo. Na UFBA, aos professores do programa de doutorado, em particular, ao prof. Manoel que compreendeu a fase difícil que passei durante o internamento de meu pai, permitindo que eu finalizasse sua disciplina em momento posterior. Ao pessoal da CEAPG, que prontamente atendia a todas as minhas solicitações. Aos professores Tayana e Ecivaldo, cujas orientações, no decorrer da qualificação, foram essenciais.

Ademais, a toda a família Jasmim Reis, destacando o seu patriarca Sr. José Emílio e matriarca D. Miracy, que sempre me acolheram com tanto carinho, principalmente após a perda de meus pais. Aos médicos, fisioterapeutas e psicólogos essenciais à recuperação de minha saúde, física e psicológica: Eliete Wolf, Adriana Prata, Tatiana Aiello, Paulo Viana, Renata Reis, Reinaldo Figueiredo, Fernanda Nascimento, Marcelli, Rodrigo, Maridete Jasmin, Fernanda Reis, André, Angélica Hermínia, Cybelle...

Geane, não poderia me esquecer de você, que além de Adriana, testemunhou tão de perto tudo pelo que passei nesses anos e foi importante dando-me forças, nos dias que estava aqui em casa.

Finalmente agradeço a todos familiares e amigos que estavam torcendo por mim: Analuiza, Marcela, Maria dos Prazeres, tia Vera, tio Carlinhos, Elissandro, Jonas, Myrna, Simone... Opa, por último, a minha Golden Sandy, que esteve deitada ao meu lado todo esse tempo, pelo seu amor e companhia incondicional :-))

"Olho em tudo e sempre encontro a Ti, Estás no céu, na terra, aonde for, em tudo que me acontece encontro Teu Amor"

—AUTORIA DESCONHECIDA

RESUMO

Contextualização: Estudos apontam várias deficiências apresentadas pelos profissionais de engenharia de software recém-formados que trabalham na indústria. O método de ensino usualmente aplicado na educação em engenharia de software resume-se à extensa exposição de conteúdo e à realização de um projeto prático, que pouco reflete a situação real dos projetos existentes na indústria. Dentre as possibilidades de aproximar os estudantes a ambientes mais realísticos, a abordagem de adoção de projetos de código aberto (*open source*) tem-se apresentado como opção interessante, principalmente para os casos onde o contrato de cooperação com a indústria não seja viável. Diferentemente de projetos proprietários, o código e ambientes de desenvolvimento reais estão acessíveis por meio da Internet. O estudante pode estudar seu código, gerar contribuições e interagir com a comunidade de usuários e desenvolvedores do projeto.
Objetivo da pesquisa: o objetivo do presente trabalho foi investigar como projetos de código aberto têm sido incorporados na educação formal em engenharia de software e a eficácia desta abordagem, com relação ao alcance de um subconjunto de objetivos de aprendizagem, a ajudar a suprir um subconjunto de deficiências de formação apontadas pela indústria e a contribuir para a preparação do estudante para o mercado de trabalho por representar uma experiência de mundo real.
Metodologia: para identificarmos como a abordagem de uso de projetos de código aberto tem sido empregada para a aprendizagem de engenharia de software, executamos um mapeamento sistemático da literatura. Desse modo, ao longo do processo sistemático pré-definido, selecionamos e classificamos 72 publicações. Antes do início da análise da eficácia da abordagem, estabelecemos um conjunto de objetivos de aprendizagem que poderiam ser trabalhados por meio do emprego de projetos de código aberto, tendo como base a visão acadêmica, as deficiências de formação apontadas pela indústria de software, as indicações dos trabalhos relacionados e a nossa própria reflexão. Em seguida, estudamos a adoção de projetos de código aberto em três estudos de caso distintos, para a aprendizagem de (a) evolução e manutenção de software, (b) testes de software e (c) engenharia reversa de requisitos. No julgamento da possibilidade de alcance dos objetivos de aprendizagem, definimos algumas heurísticas baseadas na análise de quatro dimensões: (i) a percepção dos estudantes sobre a sua aprendizagem a respeito de cada objetivo de aprendizagem diretamente investigado, antes e depois da adoção do projeto de código aberto; (ii) a análise dos trabalhos realizados pelos estudantes; (iii) a percepção dos estudantes sobre a contribuição da execução das atividades no projeto de código aberto para a aprendizagem de cada objetivo de aprendizagem diretamente investigado e, (iv) a aprendizagem relatada pelos estudantes entrevistados. Para não ficarmos restritos aos nossos resultados, executamos também a síntese das evidências reportadas pelos estudos relacionados. Dos 72 trabalhos recuperados pelo processo de mapeamento sistemático, 18

publicações atenderam aos critérios que definimos para a síntese. Ao final dos trabalhos, realizamos a triangulação entre todos os achados, ressaltando as interseções e particularidades de cada investigação, discutindo-os à luz dos objetivos traçados para a pesquisa e da literatura existente.
Contribuições: as principais contribuições obtidas com a realização desta pesquisa foram: (i) uma visão geral e abrangente de como projetos de código aberto têm sido aplicados no processo de ensino-aprendizagem de engenharia de software; (ii) identificação dos objetivos de aprendizagem em engenharia de software a partir da correlação entre a visão acadêmica e as lacunas na formação dos estudantes recém-formados relatadas pela indústria de software; (iii) identificação de quais objetivos de aprendizagem em engenharia de software têm sido alvos da utilização de projetos de código aberto, segundo a literatura existente, e quais ainda podem ser favorecidos por esta utilização; (iv) síntese dos resultados obtidos pelos diversos trabalhos publicados relacionados à adoção de projetos de código aberto na educação em engenharia de software; (v) evidências sobre a utilização de projetos de código aberto para a aprendizagem de evolução e manutenção de software, testes de software e requisitos de software; (vi) análise sobre a possibilidade de alcance de um subconjunto dos objetivos de aprendizagem por parte dos estudantes, com a adoção de projetos de código aberto na educação em engenharia de software; (vii) uma visão sobre as possíveis consequências para a aprendizagem dos estudantes com a adoção de projetos de código aberto na educação em engenharia de software, por último, (viii) análise do suporte da adoção de projetos de código aberto na educação em engenharia de software à solução de deficiências na formação dos estudantes recém-formados apontadas pela indústria.
Conclusão: Os resultados encontrados são majoritariamente positivos em termos da eficácia investigada. Por outro lado, a adoção de projetos de código aberto também impõe dificuldades que precisam ser transpostas. Em virtude das diversas variáveis envolvidas em pesquisas em educação e do caráter qualitativo empregado neste estudo, os resultados não podem ser generalizados para qualquer contexto. Mesmo assim, servem para o direcionamento de futuras pesquisas e, principalmente, proveem evidências para que educadores tomem suas decisões quanto à adoção ou não da abordagem. Quanto à adoção de projetos de código aberto na educação formal em engenharia de software, concluímos que representa uma solução factível, viável e apropriada como cenário de projeto de software real a ser trabalhado pelos estudantes.

Palavras-chave: Educação em Computação; Educação em Engenharia de Software; Projetos de Código Aberto (*Open Source*); Objetivos de Aprendizagem; Aprendizagem Experiencial; Aprendizagem a partir de Projetos Reais; Eficácia da Aprendizagem.

ABSTRACT

Contextualization: Studies point to several shortcomings presented by recently graduated software engineering professionals working in industry. The teaching approach usually applied in software engineering education is based on extensive content presentation and a practical project that hardly reflects actual existing industry projects. The approach of incorporating open source projects in software engineering education has been presented as an option to let them work with more realistic environments, especially when cooperation with industry is not feasible. Unlike proprietary software, real-world code and development environments are accessible through the Internet. Students can study their code, generate contributions and interact with the community of users and developers of the project.
Goals: The purpose of this research was to investigate how open source projects have been incorporated into formal education in software engineering and the effectiveness of this approach. Such effectiveness was measured against the achievement of a subset of learning objectives, the support to fill the gap of a subset of deficiencies pointed out by the industry, and the contribution to the student's preparation for the marketplace by providing a real world experience.
Methodology: To identify how the approach of using open source projects for software engineering learning, we performed a systematic mapping study of the literature. Thus, throughout a systematic process, we selected and classified 72 papers. Before analyzing the approach effectiveness, we established a set of learning objectives that could be practiced by using open source projects. This set was based on the points of view of academia, industry, related work and our own reflection. Next, we studied the adoption of open source projects in three separate case studies for learning (a) software evolution and maintenance, (b) software testing, and (c) reverse engineering of requirements. To judge the feasibility of accomplishing the learning objectives, we defined some heuristics based on the analysis of four dimensions. They were: (i) students' perception on their learning of each investigated learning objective, pre- and post-intervention; (ii) analysis of students' artifacts; (iii) students' perception of their open source project's activities that contributed to learning each investigated learning objective and (iv) learning reported by interviewed students. To gather results different from our own experiences, we also performed a synthesis of evidence from related studies. From 72 papers retrieved by the systematic mapping study, 18 papers met the criteria defined for the synthesis. At the end of our work, we triangulated all the findings, highlighting the intersections and particularities of each case, and discussed them in the light of our research goals and the existing literature.
Contributions: The main contributions of this research were: (i) a general and comprehensive overview of how open source projects have been applied to teach and learn

software engineering; (ii) identification of software engineering learning objectives from correlating the viewpoints of academia and industry; (iii) identification of particular software engineering learning objectives that have been targeted by the use of open source projects and that are favored by such use, according to the literature; (iv) synthesis of the evidence from the literature regarding the adoption of open source projects in software engineering education; (v) Evidence on the use of open source projects for learning evolution and maintenance of software, software testing and software requirements; (vi) analysis of the feasibility of a subset of learning objectives with the adoption of open source projects in software engineering education; (vii) a view on the possible consequences of open source projects in students' learning in software engineering education, and, last, (viii) analysis of support for the adoption of open source projects in software engineering education to fill the gaps in training of newly graduated students as pointed out by the industry.

Conclusion: The results found are mostly positive in terms of the investigated effectiveness. On the other hand, the adoption of open source projects also imposes difficulties that need to be overcome. Due to the variables involved in education research and the qualitative approach employed in this study, results may not be generalized to other contexts. Nonetheless, they serve to guide future research and, especially, provide evidence for educators to decide whether to adopt the approach. Regarding the adoption of open source projects in formal education in software engineering, we concluded that it is a feasible and appropriate solution as a scenario of real software project to involve students.

Keywords: Computing Education; Software Engineering Education; Open Source Projects; Learning Goals; Experiential Learning; Learning from Real Projects; Learning effectiveness.

SUMÁRIO

Capítulo 1—Introdução 1

1.1 Contextualização, problemática, questões de pesquisa 1
1.2 Objetivos 11
1.3 Justificativas 12
1.4 Contribuições 14
1.5 Organização do documento 16

Capítulo 2—Fundamentação Teórica 19

2.1 Definição de aprendizagem 20
2.2 A metáfora do processamento da informação 20
2.3 Elementos relevantes para a aprendizagem 23
2.4 Teorias que fundamentam o emprego de OSP 26

Capítulo 3—Objetivos de Aprendizagem vs. OSP 33

3.1 Objetivos de Aprendizagem 33
3.2 Objetivos de Aprend. em ES sob a Perspectiva Acadêmica 34
3.3 Objetivos de Aprend. em ES sob a Perspectiva da Indústria 40
3.4 Indicações de uso de OSP na Educação em Engenharia de Software . . . 42
3.5 Objetivos de Aprendizagem em Engenharia de Software x OSP 44

Capítulo 4—Procedimentos de Investigação 51

4.1 Mapeamento Sistemático 52
4.2 Revisão de Literatura e Definição dos Objetivos de Aprendizagem Factíveis 55
4.3 Síntese de evidências 55
4.4 Estudos de Caso 56
4.5 Julgamento do alcance dos objetivos de aprendizagem 59
4.6 Comparação de resultados 62
4.7 Questões Éticas 64
4.8 Considerações sobre a validade 65

Capítulo 5—Como OSP têm sido Incorporados na EES e quais as evidências existentes 71

5.1 Quem, para que e como OSP têm sido utilizados na EES 71
5.2 Síntese das evidências reportadas pelos estudos selecionados 73

5.2.1 Restrição dos estudos 73
5.2.2 Definição da estratégia para síntese das evidências 77
5.2.3 Síntese das evidências 79
5.3 Integração da síntese aos objetivos da pesquisa 95
5.3.1 Objetivos de aprendizagem em ES alcançados segundo evidências existentes nos estudos relacionados 97
5.3.2 Outras consequências para aprendizagem identificadas a partir das evidências reportadas 99
5.3.3 Relação entre as evidências reportadas e as lacunas de formação apontadas pela indústria 100

Capítulo 6—Estudo de Caso 1 – Evolução e Manutenção de Software 103

6.1 Contexto do Estudo de Caso em Evolução e Manutenção de Software . . 104
6.1.1 Cenário do Estudo de Caso 104
6.1.2 População do Estudo e Envolvidos 105
6.1.3 Aplicação da abordagem 105
6.1.4 Coleta de Dados 106
6.1.5 Análise de dados 107
6.2 Resultados 107
6.2.1 Projeto de Código Aberto 107
6.2.2 Experiência de Mundo Real 110
6.2.3 Dificuldades 115
6.2.4 Associação Teoria e Prática 116
6.2.5 Motivação 119
6.2.6 Aprendizagem 121
6.2.7 Satisfação 128
6.2.8 Escolha do Software 136
6.3 Estudo de Caso Exploratório e Resultados para a Pesquisa 138
6.3.1 Integração com a pesquisa final 138
6.3.2 Análise dos objetivos de aprendizagem alcançados 138
6.3.3 Análise da contribuição da adoção de OSP no suprimento das lacunas de formação apontadas pela indústria 139

Capítulo 7—Estudo de Caso 2 - Testes de Software 143

7.1 Contexto do Estudo de Caso em Testes de Software 143
7.1.1 Cenário do Estudo de Caso 143
7.1.2 População do Estudo e Envolvidos 144
7.1.3 Aplicação da abordagem 145
7.1.4 Coleta de Dados 149
7.1.5 Análise de dados 150
7.2 Resultados 150
7.2.1 Experiência Real de Desenvolvimento de Software (Ob7) 150
7.2.2 Conexão teoria e prática (Ob6) 161

7.2.3 Objetivos de Aprendizagem Alcançados (Ob4) 168
7.2.4 Sentimento de preparação para o mercado de trabalho (Ob8) . . . 188
7.2.5 Consequências para a aprendizagem (Ob5) 193
7.2.6 Lacunas na Indústria de Software (Ob9) 200

Capítulo 8—Estudo de Caso 3 - Engenharia Reversa de Requisitos 205

8.1 Contexto do Estudo de Caso em Requisitos de Software 206
8.1.1 Cenário do Estudo de Caso 206
8.1.2 População do Estudo e Envolvidos 207
8.1.3 Aplicação da abordagem 207
8.1.4 Coleta de Dados 211
8.1.5 Análise de dados 213
8.2 Resultados 213
8.2.1 Experiência Real de Desenvolvimento de Software (Ob7) 213
8.2.2 Conexão teoria e prática (Ob6) 222
8.2.3 Objetivos de Aprendizagem Alcançados (Ob4) 228
8.2.4 Sentimento de preparação para o mercado de trabalho (Ob8) . . . 244
8.2.5 Consequências para a aprendizagem (Ob5) 248
8.2.6 Lacunas na Indústria de Software (Ob9) 257

Capítulo 9—Comparação entre os Estudos e Discussão 261

9.1 Experiência Real de Desenvolvimento de Software (Ob7) 261
9.2 Conexão teoria e prática (Ob6) 267
9.3 Objetivos de Aprendizagem Alcançados (Ob4) 272
9.4 Sentimento de preparação para o mercado de trabalho (Ob8) 279
9.5 Lacunas na Indústria de Software (Ob9) 282
9.6 Consequências para a aprendizagem (Ob5) 284
9.7 Dificuldades provenientes da aplicação da abordagem 293

Capítulo 10—Conclusões e Trabalhos Futuros 299

Apêndice A—Conjunto de Objetivos de Aprendizagem em Engenharia de Software 317

Apêndice B—Questionário LET - Estudo de Caso 2 379

Apêndice C—Questionário QT1 - Estudo de Caso 2 381

Apêndice D—Questionário QT2 - Estudo de Caso 2 383

Apêndice E—Questionário LER – Estudo de caso 3 389

Apêndice F—Questionário QR1 - Estudo de Caso 3 391

Apêndice G—Questionário QR2 - Estudo de Caso 3 393

LISTA DE FIGURAS

2.1 Modelo de Processamento de Informação 21

3.1 Visão dos objetivos de aprendizagem factíveis. 49

4.1 Processo do mapeamento sistemático, adaptado de Petersen et al. (2008). 54
4.2 Algoritmo de aplicação das heurísticas para o julgamento da contribuição da adoção de projetos de código aberto para o alcance dos objetivos de aprendizagem. 69

6.1 Temas provenientes da adoção de um OSP para a aprendizagem de evolução e manutenção de software. 108
6.2 Situações proporcionadas por um projeto de código aberto. 108
6.3 Características que aproximam a abordagem de uso do OSP de uma experiência de mundo real. 111
6.4 Dificuldades a serem enfrentadas. 115
6.5 A associação entre teoria e prática para a aprendizagem. 116
6.6 Condições para a motivação do estudante. 119
6.7 Grau de motivação para a realização das atividades considerando o tamanho e a complexidade do JabRef. 121
6.8 Aprendizagem proporcionada pela adoção de OSP no estudo de caso sobre evolução e manutenção de software (EC1). 122
6.9 Sequência de contribuição do método didático para a aprendizagem. . . . 128
6.10 Satisfação com a disciplina e projetos práticos. 129
6.11 Fatores que influenciam a escolha do software. 137

7.1 Níveis de concordância sobre a abordagem como uma experiência real de desenvolvimento de software no estudo EC2. 151
7.2 Percepção dos estudantes sobre a proximidade do projeto a uma experiência real de desenvolvimento de software no estudo EC2. 152
7.3 Compreensão dos estudantes sobre a conexão entre a teoria e a prática em projetos reais no estudo EC2. 162
7.4 Percepção dos estudantes sobre a conexão teoria e prática no estudo EC2. 162
7.5 Contribuição da abordagem para a aprendizagem sobre testes de software. 170
7.6 Contribuição do projeto para o alcance de objetivos de aprendizagem relacionados à prática profissional de engenharia de software no estudo EC2. 182
7.7 Níveis de concordância com relação à preparação para o mercado no estudo EC2. 189

7.8 Níveis de concordância com relação à contribuição do projeto para conscientização dos estudantes no estudo EC2 197
7.9 Contribuição da abordagem para a aprendizagem no estudo de caso sobre testes de software (EC2). 201

8.1 Níveis de concordância sobre a abordagem como uma experiência real de desenvolvimento de software no estudo EC3. 214
8.2 Percepção dos estudantes sobre a proximidade do projeto a uma experiência real de desenvolvimento e manutenção de software no estudo EC3. 215
8.3 Compreensão dos estudantes sobre a conexão entre a teoria e a prática em projetos reais no estudo EC3. 222
8.4 Percepção dos estudantes sobre a conexão teoria e prática no estudo EC3. 223
8.5 Contribuição da abordagem para a aprendizagem sobre requisitos de software. 229
8.6 Contribuição do projeto para o alcance de objetivos de aprendizagem relacionados à prática profissional de engenharia de software no estudo EC3. 238
8.7 Níveis de concordância com relação à preparação para o mercado no estudo EC3. 244
8.8 Níveis de concordância com relação à contribuição do projeto para conscientização dos estudantes no estudo EC3. 253
8.9 Contribuição da abordagem para a aprendizagem no estudo de caso sobre requisitos de software (EC3). 258

9.1 Contribuição da prática para o processo de aprendizagem. 270
9.2 Contribuição da prática com projetos reais para o desenvolvimento de competências. 271
9.3 Importância da teoria para a prática. 271
9.4 Visão sobre o processo de aprendizagem com OSP. 291

LISTA DE TABELAS

3.1 Dimensão Conhecimento da Taxonomia de Bloom Revisada 35
3.2 Dimensão Processo Cognitivo da Taxonomia de Bloom Revisada 35
3.3 Níveis de Aprendizagem no CS2013. 37
3.4 Resultados de Aprendizagem (CS2013) x Objetivos Aprendizagem. 39
3.5 Deficiências de Formação x Objetivos de Aprendizagem. 41
3.6 Escopo de Indicações de Adoção de OSP. 45
3.7 Objetivos de Aprendizagem Factíveis por Subárea do SWEBOK. 45

4.1 Procedimentos de investigação empregados ao longo da pesquisa. 53
4.2 Visão geral dos estudos de caso realizados. 58
4.3 Heurística utilizada para o julgamento da contribuição da adoção de projetos de código aberto para o alcance dos objetivos de aprendizagem. . . 63

5.1 Artigos provenientes do mapeamento sistemático que apresentam evidências sobre o uso de OSP. 74
5.2 Artigos cujas evidências foram sintetizadas. 76
5.3 Escala estabelecida para interpretação das evidências. 77
5.4 Heurísticas para interpretação dos resultados quantitativos. 80
5.5 Evidências reportadas sobre a aprendizagem de modo geral. 81
5.6 Evidências reportadas sobre a aprendizagem conceitual. 81
5.7 Evidências reportadas sobre o desenvolvimento de habilidades técnicas. . 82
5.8 Evidências reportadas sobre o desenvolvimento de habilidades gerais. . . 84
5.9 Evidências reportadas sobre mudanças de atitudes. 86
5.10 Evidências reportadas sobre a experiência profissional proporcionada. . . 88
5.11 Evidências reportadas sobre a associação teoria e prática. 89
5.12 Evidências reportadas sobre os benefícios proporcionados pelo uso de projetos de código aberto. 90
5.13 Evidências sobre as dificuldades encontradas no uso de projetos de código aberto. 90
5.14 Evidências reportadas sobre as dificuldades de seleção do OSP por parte dos estudantes. 93
5.15 Evidências reportadas sobre motivação. 93
5.16 Evidências reportadas sobre satisfação. 94
5.17 Evidências reportadas sobre a possibilidade de continuidade. 96
5.18 Heurística para análise do alcance do objetivo segundo as evidências reportadas. 97
5.19 Alcance de objetivos de aprendizagem segundo as evidências reportadas. 97

5.20 Outras aprendizagens proporcionadas. 100
5.21 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com a síntese das evidências. 101

6.1 Entrevistas realizadas por fase e experiência prévia dos estudantes participando de projetos. 106
6.2 Objetivos de aprendizagem alcançados no estudo de caso sobre Evolução e Manutenção de Software (EC1). 140
6.3 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Evolução de Software (EC1). 141

7.1 Atividades necessárias para emprego da abordagem. 146
7.2 Encontros presenciais da pesquisadora com os estudantes no estudo EC2. 147
7.3 Material disponibilizado no estudo EC2. 148
7.4 Caracterização dos estudantes entrevistados no estudo EC2. 150
7.5 Comparação entre a aprendizagem após o método tradicional de ensino e após o emprego da abordagem com OSP (EC2). 168
7.6 Resumo da produção de testes por equipe 175
7.7 Falhas e problemas encontrados nos testes implementados 176
7.8 Julgamento do alcance dos objetivos de aprendizagem associados aos aspectos técnicos no estudo EC2. 178
7.9 Objetivos de aprendizagem explicitamente investigados no estudo EC2. . 181
7.10 Outras competências também desenvolvidas pela abordagem no estudo EC2.183
7.11 Julgamento do alcance dos objetivos de aprendizagem associados às competências da prática profissional para o estudo de caso sobre testes de software (EC2). 187
7.12 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Testes de Software (EC2).201

8.1 Atividades necessárias para emprego da abordagem 208
8.2 Encontros presenciais da pesquisadora com os estudantes no estudo EC3. 210
8.3 Material disponibilizado no estudo EC3. 210
8.4 Caracterização dos estudantes entrevistados no estudo EC3. 212
8.5 Comparação entre a aprendizagem após o método tradicional de ensino e após o emprego da abordagem com OSP (EC3). 230
8.6 Julgamento do alcance dos objetivos de aprendizagem associados aos aspectos técnicos para o estudo de caso sobre requisitos de software (EC3). 236
8.7 Objetivos de aprendizagem explicitamente investigados no estudo EC3. . 237
8.8 Outras competências também desenvolvidas pela abordagem no estudo EC3.239
8.9 Julgamento do alcance dos objetivos de aprendizagem associados às competências da prática profissional para o estudo de caso sobre requisitos de software (EC3). 242

8.10 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Requisitos de Software (EC3). 259

9.1 Características do JabRef que o aproximam dos softwares existentes nas empresas. 262
9.2 Proximidade das atividades realizadas nas empresas. 264
9.3 Dificuldades ao ter que lidar com um OSP. 265
9.4 Habilidades necessárias percebidas pelos estudantes após a participação no projeto. 267
9.5 Resumo das percepções dos estudantes sobre a conexão entre a teoria e a prática nos três estudos de caso. 269
9.6 Competências da prática profissional alcançadas. 276
9.7 Contribuição do OSP para preparação para o mercado capturada em cada estudo de caso. 279
9.8 Contribuição para o suprimento das lacunas de formação apontadas pela indústria. 283
9.9 Características do OSP que favorecem a aprendizagem. 286
9.10 Conscientizações geradas pela aplicação da abordagem em cada estudo. . 287
9.11 Concordância dos estudantes sobre a contribuição da abordagem para compreensão da importância dos aspectos estudados. 288
9.12 Dificuldades enfrentadas pelo professor ao aplicar a abordagem. 293

A.1 Objetivos de Aprendizagem em ES 320

Capítulo 1

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO, PROBLEMÁTICA, QUESTÕES DE PESQUISA

Responsabilidades do engenheiro de software

Todo profissional é especialista em determinada área de conhecimento, devendo aplicar este conhecimento em favor dos membros da sociedade que não o possuem (ACM AND IEEE, 2015). Assim, os engenheiros são os profissionais que usam a matemática, a ciência e a tecnologia mais recente para construir produtos que são adequados e importantes para a segurança e bem-estar da comunidade (PARNAS, 1999). A fim de gerar as soluções adequadas para problemas práticos, o engenheiro precisa analisar alternativas, em virtude de restrições técnicas, tempo, custo ou mesmo de negócio, que possam existir. Desse modo, a opção mais apropriada para o problema em mãos, muitas vezes considera tanto razões técnicas quanto não técnicas. Shaw (2000) salienta que os engenheiros preferencialmente aplicam o conhecimento científico e matemático, todavia, em outros momentos, precisam lidar com conhecimento menos sistemático.

Os futuros engenheiros, para projetar produtos confiáveis precisam então aprender (PARNAS, 1999): (i) o que é verdadeiro e útil dentro da área de conhecimento de sua especialidade; (ii) como aplicar este conhecimento; (iii) como aplicar o conhecimento mais abrangente que é necessário para construir produtos que funcionem em um ambiente real, e (iv) um modo disciplinado de análise e projeto que deve ser seguido, de maneira que cumpram as responsabilidades que recaem sobre aqueles que constroem produtos para os outros.

Tratando-se da engenharia de software (SE - *Software Engineering*) o produto em questão é o software. Sobre o software pode-se destacar três características: intangibilidade, complexidade e importância. A característica de intangibilidade diferencia-o dos produtos gerados pelas outras engenharias, de maneira que o engenheiro de software, diferentemente dos demais engenheiros, também precisa lidar com a possibilidade de modificação constante de seu produto durante todo o ciclo de vida. A evolução dos ambientes computacionais fez com que o software deixasse de ser um conjunto de programas desenvolvidos utilizando-se apenas de uma simples linguagem de programação e executados em

um único dispositivo. Logo, atualmente, o software pode ser um produto extremamente complexo, não só pelo domínio de problema que se propõe a resolver, como também, pelas diversas linguagens e tecnologias que são empregadas. Enfim, a importância do software retrata-se pela sua presença direta ou indiretamente em vários aspectos do cotidiano das pessoas: fornecimento de energia, comunicação, transporte, suprimento de produtos industrializados, educação, saúde, entretenimento, etc. Enquanto que as duas primeiras características do software representam desafios a mais com os quais os engenheiros de software precisam saber lidar, a dependência da sociedade moderna exige maior responsabilidade desses profissionais em gerar produtos de qualidade.

Paralelamente, Shaw (2000) destaca que a qualidade do software depende, dentre outros fatores, da proficiência e atualização dos desenvolvedores do software, consequentemente, da formação desses desenvolvedores. Meyer (2001) ressalta que, independente da graduação ser em ciência da computação ou em engenharia de software, as instituições de ensino são as responsáveis em formar profissionais que irão construir e manter os softwares capazes de satisfazer às necessidades dos usuários. Radermacher e Walia (2013) acrescentam à ciência da computação e engenharia de software, outros cursos de graduação relacionados (sistemas de informação e tecnologia da informação), de modo que um dos principais focos dos educadores deve ser preparar os estudantes de graduação para futuras carreiras na indústria. Concluindo, independentemente do curso de graduação relacionado à computação, a depender do interesse dos estudantes pela área de desenvolvimento de software, os mesmos devem aprender os princípios e as práticas atuais de engenharia de software, de maneira que sejam preparados para trabalhar na indústria de software.

Falta de capacitação de profissionais para a Indústria

Há quase 20 anos, estudos comentam sobre a existência de alguma insatisfação da indústria em relação à capacitação dos profissionais recém-formados. Em 1999, Parnas (1999) relatou que enquanto os consumidores queixavam-se da baixa qualidade dos softwares, as empresas desenvolvedoras de software reclamavam da pouca oferta de pessoal qualificado. Meyer (2001) mencionou a escassez de pessoal qualificado na Europa e Austrália, uma vez que os gerentes procuravam por desenvolvedores excelentes, não apenas empregados. Lutz, Naveda e Vallino (2014) declararam que a motivação para a criação do curso de graduação em engenharia de software no Rochester Institute of Technology nos Estados Unidos em 1996 foi a necessidade de prover desenvolvedores de software mais capacitados para a indústria. Porém, até o momento, não acreditam que esta questão esteja resolvida, já que muitos profissionais ainda são oriundos de cursos de ciência da computação. Segundo os autores, tais estudantes adquirem conhecimento no projeto de compiladores, sistemas operacionais, outros tópicos de interesse de cientistas da computação, mas têm pouca experiência em trabalhar em equipe, testes ou mesmo outras práticas comuns na indústria, como gerenciamento de configuração, controle de versões, etc. (LUTZ; NAVEDA; VALLINO, 2014).

Ao longo desse período, vários estudos procuraram identificar quais eram as dificuldades que os recém-formados deparavam-se no início de seu primeiro emprego. A fim de consolidar os resultados desses trabalhos, Radermacher e Walia (2013) conduziram uma revisão sistemática. Essencialmente, os trabalhos inseridos na revisão apresentavam

resultados empíricos e indicavam ou uma lacuna entre as habilidades e conhecimento apresentados pelos egressos e as expectativas da indústria ou academia, ou áreas nas quais os recém-formados eram fracos ou apresentavam baixa performance. Desse modo, foram encontradas as mais diversas deficiências, envolvendo habilidades técnicas (e.g., gerenciamento de projetos, uso de ferramentas, testes, programação, requisitos, design), habilidades pessoais (e.g., comunicação oral e escrita, trabalho em equipe, solução de problemas, pensamento crítico) e profissionalismo (comportamento ético).

Em um trabalho subsequente, o mesmo grupo de pesquisa entrevistou 23 gerentes e indivíduos responsáveis pela contratação de pessoal de empresas situadas nos Estados Unidos e Europa (RADERMACHER; WALIA; KNUDSON, 2014). O objetivo era constatar as deficiências que recém-formados enfrentavam ao trabalhar nessas empresas, bem como, quais deficiências de conhecimento evitariam que recém-formados fossem contratados pelas mesmas. Paralelamente, os pesquisadores buscaram comparar os resultados com os da revisão sistemática da literatura, com a finalidade de fortalecer os resultados e prover uma visão mais atualizada da área, além de buscar informações qualitativas mais detalhadas das deficiências, uma vez que, nem todos os trabalhos que compreenderam a revisão disponibilizavam-nas. Os resultados do estudo apontaram as dificuldades dos recém-formados com várias habilidades, sendo as mais frequentes: (i) o uso de ferramentas (principalmente de gerenciamento de configuração); (ii) a comunicação efetiva com colegas de trabalho e clientes, e (iii) a realização de testes de unidade (construção de casos de teste não redundantes). Comparando os resultados das pesquisas, entre as deficiências mais comuns na revisão sistemática, seis também estavam entre as mais frequentes neste último estudo. Por fim, o estudo também destacou que a falta de experiência na participação de projetos e habilidade na solução de problemas são as questões comumente citadas como empecilho à contratação dos egressos.

Necessidade de prática

Para Parnas (1999), é razoável que estudantes aprendam "sobre coisas" em cursos de ciências, entretanto estudantes de engenharia precisam aprender como "fazer coisas". Segundo ele, a experiência prática para a engenharia é muito mais importante do que para algumas ciências. Os cursos precisam integrar a teoria com a prática e essencialmente, como aplicar a teoria na prática. Lembrando que, "para aprender a fazer, é preciso fazer" Parnas (2014).

Os currículos de referência de engenharia de software e ciência da computação, definidos por um trabalho conjunto entre a *Association for Computing Machinery* (ACM) e a *Institute of Electrical and Electronics Engineers Computer Society* (IEEE-CS), enfatizam a necessidade da prática profissional e participação dos estudantes em projetos reais (ACM AND IEEE, 2015) (ACM AND IEEE, 2013). Definem a "Prática Profissional" e, "Questões Sociais e Prática Profissional" , respectivamente, como áreas de conhecimento que precisam ser desenvolvidas ao longo dos cursos de graduação. Espera-se que, no decorrer das disciplinas, o estudante adquira e/ou aperfeiçoe por meio de experiência prática as habilidades de comunicação, trabalho em equipe, ética profissional, solução de problemas, etc. Especificamente, o guia para elaboração dos currículos de graduação em engenharia de software trata a questão da prática em duas diretrizes (ACM AND IEEE,

2015). A diretiva 12 destaca que mesmo que o foco da aprendizagem não seja o uso de ferramentas, os estudantes devem adquirir experiência no emprego de ferramentas apropriadas e atuais. Enquanto que, a diretiva 14 ressalta que o currículo do curso deve ter uma base significativa no mundo real, uma vez que a incorporação de elementos do mundo real é essencial para a aprendizagem efetiva dos conceitos da engenharia de software. Já o guia para a elaboração de currículos em ciência da computação (ACM AND IEEE, 2013) espera que, dentre outras características, os graduados em ciência da computação apresentem experiência com projetos. Desse modo, os estudantes devem participar de pelo menos um projeto que os desafie a integrar os conhecimentos adquiridos, que exija que os mesmos avaliem as soluções em potencial, enfim, que represente um projeto em larga escala, diferentemente de projetos típicos das disciplinas. O documento ainda realça que apesar de outras experiências sejam apropriadas em circunstâncias particulares, na maioria das vezes, essa experiência dá-se com o desenvolvimento de um projeto de software.

No Brasil, a proposta de diretrizes curriculares do Ministério da Educação para os cursos de computação (MEC, 2012) expressa que os currículos devem prover a coexistência de relações entre teoria e prática, de maneira que o estudante futuramente possa adaptar-se às novas situações de sua área de formação. Especificamente para os cursos de engenharia de software e ciência da computação, estabelece, dentre o desenvolvimento de outras competências, que os estudantes sejam capazes de elaborar soluções, individualmente ou em equipe, para problemas complexos relacionados aos domínios de conhecimento e de aplicação.

Método tradicional de ensino e os problemas decorrentes

No entanto, ao longo das diversas disciplinas dos cursos de graduação, percebe-se que usualmente os estudantes são acostumados a escrever programas relativamente pequenos, trabalhar de forma individual e sem seguir nenhum processo formal de desenvolvimento, para resolver os exercícios definidos pelo professor (ZHANG; SU, 2007). Comumente, os estudantes acreditam que basta que o programa funcione para que tenham conseguido obter uma solução aceitável (ZHANG; SU, 2007). O conteúdo da disciplina de engenharia, frequentemente, é trabalhado a partir de uma extensa exposição de conteúdo e a realização de um projeto prático. O projeto geralmente é pequeno, abrange um pequeno grupo de estudantes, tem seus requisitos definidos pelo professor e estudantes envolvidos, segue o modelo cascata de desenvolvimento e é desenvolvido durante apenas um semestre letivo. Ao final, termina apresentando uma arquitetura simplificada e baixa qualidade de código, reflexo das limitações do modelo cascata e do conhecimento e experiência prévia dos estudantes (RAJLICH, 2013).

Esse modo tradicional de ensino, aqui descrito de maneira bem genérica, apesar de corresponder a uma forma onde o estudante precisa colocar em prática o conteúdo aprendido, acarreta em vários problemas, discutidos a seguir:

- Os estudantes aprendem a desenvolver os programas a partir do zero. Situação que não representa a realidade da indústria de software, onde a maioria dos programadores gasta a maior parte de seu tempo modificando ou evoluindo produtos já

existentes (SHAW, 2000) (BUCHTA et al., 2006) (SZABO, 2014) (PARNAS, 2014).

- Raramente os estudantes defrontam-se com a necessidade de compreender bons exemplos de código. Assim como a leitura de bons textos é importante para aprender a escrever bem, a leitura de bons exemplos de código, desenvolvidos previamente por profissionais competentes, é importante para que o estudante aprenda a construir programas de qualidade (SHAW, 2000) (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008) (BUDD, 2009). Shaw (2000) ainda lembra que os estudantes aprendem melhor por meio de exemplos concretos e o emprego de bons exemplos acarreta que eles reutilizem esses conceitos.

- Os estudantes não precisam trabalhar com código desenvolvido por terceiros. Isso impede que eles entendam o desafio de escrever código que outros possam compreender e manter facilmente (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008). Spinellis (2006) salienta que somente quando o desenvolvedor 'suja' suas mãos, ele passa a adotar um estilo de código mais compreensível e mais fácil de manter. Enquanto que Horstmann (2009) ressalta que quando o estudante sofre com código mal estruturado e pouco documentado, ele pode motivar-se em não provocar sofrimento similar aos outros.

- Os programas desenvolvidos pelos estudantes são descartados, servindo apenas como forma de avaliação da disciplina. Como os programas não serão utilizados, os estudantes não enxergam a necessidade de criar software bem documentado, fácil de compreender e manter (SHAW, 2000).

- Os projetos desenvolvidos pelos estudantes não correspondem à realidade com a qual os profissionais na indústria precisam lidar. Os projetos acadêmicos não são sistemas reais (SHAW, 2000), de modo que os estudantes não têm experiência com projetos grandes (MEYER, 2001) (ZHANG; SU, 2007) (BUCHTA et al., 2006) (LUTZ; NAVEDA; VALLINO, 2014), complexos (LUTZ; NAVEDA; VALLINO, 2014) e onde a expectativa de qualidade é bem maior (BUCHTA et al., 2006) (LUTZ; NAVEDA; VALLINO, 2014).

- O ambiente não retrata um ambiente real de desenvolvimento. Os estudantes interagem entre si e com o professor. As equipes são pequenas e uniformes (os estudantes encontram-se no mesmo nível de conhecimento, já que pertencem a mesma turma) (BUDD, 2009). Existe uma carência de participação de usuários (MEYER, 2001), clientes (TUCKER, 2009) e outros desenvolvedores profissionais experientes (HISLOP; ELLIS; MORELLI, 2009). Os estudantes não são expostos a ferramentas de desenvolvimento atuais (ZHANG; SU, 2007).

- O modelo cascata é tradicionalmente empregado, enquanto que nos dias de hoje a indústria propõe a utilização de métodos ágeis e evolucionários (BUCHTA et al., 2006) (RAJLICH, 2013).

- Os estudantes não visualizam a importância de muitos princípios da engenharia de software. Como os estudantes não vivenciam a grande maioria dos problemas

existentes no ciclo de vida de um software grande e complexo, eles não conseguem compreender a importância prática dos princípios da engenharia de software (LIU, 2005) (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008). Consequentemente os princípios tornam-se meramente conceitos acadêmicos (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008). A partir da necessidade de modificar código complexo, o estudante entende a relevância da manutenibilidade, a aplicabilidade de conceitos como coesão e acoplamento, o valor da documentação, do uso de padrões, etc. Na mesma linha de pensamento, (RAJLICH, 2013) prega o ensino de algumas habilidades fundamentais a partir da experiência prática com a manutenção de projetos reais, não só devido à aplicabilidade dessas habilidades nas futuras carreiras dos estudantes, mas porque elas também permitem a melhor compreensão de questões que são discutidas em outros tópicos da engenharia de software.

Abordagens para aproximar os estudantes de ambientes mais realísticos

A universidade não é uma empresa da indústria de software e nem deve ser, assim é impossível reproduzir todas as circunstâncias do mundo real completamente (MEYER, 2001). Incorporar práticas realísticas aos cursos não é uma tarefa fácil. Muitos obstáculos precisam ser transpostos, por exemplo: alinhar os objetivos didáticos à experiência prática real pretendida; o tempo restrito do semestre letivo combinado com a necessidade de cobertura do conteúdo estabelecido (RAJLICH, 2013), disponibilizar e manter a infraestrutura tecnológica para o desenvolvimento do projeto atualizada, entre outros.

A literatura descreve diversas abordagens para aproximar os estudantes de ambientes mais realísticos. Meyer (2001) relata que na Universidade de Monash, em Melbourne na Austrália, os estudantes de cada nova turma, a partir dos resultados da turma predecessora, devem dar continuidade ao projeto do software definido pelo professor e que se estende por vários anos. Szabo (2014) descreve abordagem semelhante, onde os estudantes precisam adicionar funcionalidades e corrigir erros em um projeto de médio porte construído por estudantes do ano anterior. As abordagens de Nauman e Uzair (2007) e Tvedt, Tesoriero e Gary (2002) tentam simular o ambiente da indústria. No primeiro caso, o objetivo é que os estudantes agrupados em equipes trabalhem com sistemas grandes e complexos (NAUMAN; UZAIR, 2007). Todo projeto engloba uma equipe de estudantes de ciência da computação e uma equipe de estudantes de engenharia de software, cada qual com a sua responsabilidade, mas que precisam colaborar para a finalização do mesmo. No segundo caso, o currículo de ciência da computação é combinado com a aprendizagem baseada em experiência, por meio da participação dos estudantes em vários projetos que buscam refletir a tendência atual da indústria (TVEDT; TESORIERO; GARY, 2002). Ao longo de oito semestres letivos, os estudantes de ciência da computação ganham experiência assumindo, em cada novo projeto e nova equipe, os diversos papéis envolvidos no ciclo de vida do software (analista, projetista, testador, gerente de projetos). Os projetos são definidos pelos estudantes que assumem o papel de gerente (estudantes do quarto ano do curso). O professor atua apenas como facilitador, orientando e atribuindo-se papéis externos, como cliente, usuário, certificador, entre outros. Outros estudos reportam projetos de cooperação com a indústria (KNUDSON; RADERMACHER, 2009) (REICHLMAYR, 2006). No projeto final do curso em ciência

da computação da Universidade Estadual de Dakota do Norte (NDSU), os estudantes participam de projetos patrocinados por empresas locais ou regionais (KNUDSON; RADERMACHER, 2009). No início do semestre, os projetos são apresentados aos estudantes, de maneira que eles podem indicar o interesse de qual projeto gostariam de participar. O professor define as equipes de acordo com as habilidades requeridas para o projeto e interesse dos estudantes. As equipes são responsáveis por toda a condução do projeto e são autônomas no sentido de decidir como vão estruturar-se internamente, contudo devem seguir o processo de desenvolvimento previamente definido ou usar o processo da empresa patrocinadora. Para garantir a qualidade, revisões formais e informais são realizadas ao longo do projeto, bem como os clientes são envolvidos durante todo o ciclo de vida do mesmo, por meio de procedimentos de comunicação formal e informal. Já no curso de graduação em engenharia de software do Instituto de Tecnologia Rochester (RIT), os estudantes alternam períodos de estudo formal com períodos de experiência profissional remunerada em empresas de pequeno a grande porte (REICHLMAYR, 2006). A primeira experiência de cooperação com a indústria ocorre ao final do segundo ano. Até o final do quinto ano do curso, são mais três períodos de experiência, o que ao final equivale a um ano de experiência de trabalho. O projeto de final de curso provê outra oportunidade para os estudantes trabalharem com patrocinadores externos, que podem ser tanto organizações sem fins lucrativos, departamentos não acadêmicos dentro do próprio RIT, outras universidades, quanto empresas da indústria local. Para a execução do projeto, os estudantes colocam em prática as habilidades adquiridas ao longo das disciplinas e das experiências profissionais realizadas; os professores atuam apenas como orientadores.

A Abordagem Utilizando Projetos de Código Aberto

Por outro lado, o desenvolvimento de projetos de código aberto (*Open Source Project* – OSP) tornou-se um elemento importante para a engenharia de software (HORSTMANN, 2009). A expansão da Internet e o sucesso de vários projetos que atendem aos mais diversos domínios de aplicação, como sistemas operacionais (Linux), servidores de aplicação (Apache), navegadores (Mozilla Firefox), sistemas de banco de dados (PostgreSQL, MySQL), entre outros, têm provocado uma verdadeira revolução no modo pelo qual o software é desenvolvido, disseminado e adaptado (SMITH et al., 2014), ou mesmo, no modelo de negócio das empresas (SAMUELSON, 2006) (MEGIAS et al., 2009). Cada vez mais é esperado que os engenheiros de software avaliem soluções de código aberto, integrem soluções próprias com produtos ou bibliotecas de código aberto ou mesmo participem de OSP (HORSTMANN, 2009).

Corriqueiramente, um projeto de código aberto é construído por desenvolvedores voluntários através de um processo de desenvolvimento colaborativo. Tanto o executável quanto o código fonte são disponibilizados publicamente, de modo que além do software poder ser utilizado livremente, os usuários interessados podem criticar, indicar erros, propor mudanças ou ainda, caso sejam desenvolvedores, implementar tais solicitações. A qualidade é garantida devido ao intenso processo social existente com rápidos *feedbacks* (SHAW, 2000). Whitehurst (2009) ainda destaca o poder da revisão por pares, a transparência do processo e o ecossistema criado com milhares de desenvolvedores e clientes ao redor do mundo. Embora este processo de desenvolvimento aparentemente seja *ad*

hoc, muitos projetos, como os exemplos dados anteriormente, estão obtendo sucesso, provendo funcionalidades importantes e com alta confiabilidade (SMITH et al., 2014). Na verdade essas comunidades auto-organizadas de voluntários têm conseguido desenvolver, dar suporte e manter software de forma sem precedentes (STAMELOS, 2009), atraindo inclusive a participação da indústria (CARRINGTON, 2003) (SAMUELSON, 2006).

A utilização de projetos de código aberto aparece como uma importante opção para incorporar a realidade da construção de software ao ambiente acadêmico. Estes projetos representam código pré-existente com características similares às encontradas na indústria e dificilmente disponíveis na academia: software de médio e grande porte, cujo ciclo de vida extrapola o final do semestre letivo, diversas funcionalidades, requisitos de qualidade restritos, arquitetura complexa, envolvem desenvolvedores com diferentes habilidades e experiência, usuários demandando correções, modificações ou melhorias, entre outras. Código e ambiente de desenvolvimento reais estão acessíveis por meio da Internet, sem qualquer restrição de localização de instalações (YAMAKAMI, 2012), representando assim, boa oportunidade para expor os estudantes a sistemas de software realísticos e consequentemente aprimorar a aprendizagem em engenharia de software.

Argumentos para utilizar OSP

Dada a quantidade e diversidade de projetos de código aberto disponível, o professor tem em mãos uma variedade de exemplos que podem ser aplicados para que os estudantes avaliem o que é bom, o que é ruim, o que funciona, o que não funciona (BUDD, 2009). Essencialmente, o estudante pode ser exposto a uma pluralidade de estilos de programação e assim desenvolver sua capacidade crítica (CARRINGTON, 2003).

Em geral, os estudantes deparam-se com bons exemplos, uma vez que OSP costumam oferecer excelentes exemplos de projetos bem organizados (STAMELOS, 2009). Graças à importância dada à garantia da qualidade (YAMAKAMI, 2012), código mais estável e confiável é resultado da intensa inspeção realizada no processo de desenvolvimento (RAJ; KAZEMIAN, 2006). Por sua vez, Smith et al. (2014) acreditam que os desenvolvedores que participam dos projetos de código aberto possuem maior comprometimento em aplicar os princípios da engenharia de software, consequentemente produzir projetos cuidadosamente concebidos, construídos e mantidos, já que a maioria deles participa dos projetos OSP para gerar reputação, auto desenvolvimento e altruísmo (OREG; NOV, 2008).

Stamelos (2009) também ressalta que nem todo OSP é sinônimo de sucesso. Todavia, como Horstmann (2009), acredita que a aprendizagem a partir de exemplos negativos também pode ser interessante. Horstmann (2009) exemplifica que estudantes com hábitos ruins de programação são motivados a melhorar, quando se deparam com exemplos ruins, similares aos seus, em código real. McCartney, Gokhale e Smith (2012) consideram que OSP também podem exibir propriedades semelhantes a sistemas legados como código mal escrito, desestruturado, documentação desatualizada, resultado de muitas correções e funcionalidades acrescentadas ao longo do ciclo de vida destes sistemas. Contudo, essa possibilidade aproxima ainda mais tais projetos à realidade da indústria, tornando-os apropriados para o ensino de engenharia de software com foco em aspectos de manutenção.

Após a disponibilidade de código, o acesso a um ambiente real de desenvolvimento de

software é o segundo argumento mais importante para a utilização de projetos de código aberto. Os estudantes podem experimentar a dinâmica de trabalhar em um projeto de grande porte (HORSTMANN, 2009). O ambiente de desenvolvimento compreende tanto a infraestrutura tecnológica usada quanto a comunidade de desenvolvedores e usuários envolvidos.

Considerando a infraestrutura tecnológica, os estudantes deparam-se com ferramentas de controle de versões, automação de *build* e gerenciamento de projetos (HORSTMANN, 2009). Spinellis (2006) lembra que o software não é um bloco monolítico, mas na verdade, um ecossistema heterogêneo, grande e complexo. Desse modo, ao participar de um OSP, o desenvolvedor precisa trabalhar com componentes, ferramentas e configurar todo o ambiente de desenvolvimento, o que pode incluir, por exemplo, a instalação e configuração de banco de dados, configurações de rede e segurança (SPINELLIS, 2006), ou seja, desenvolver habilidades de configuração de ambientes também valorizadas pelo mercado de trabalho atualmente (SPINELLIS, 2006) (RADERMACHER; WALIA; KNUDSON, 2014).

Com relação à comunidade, Yamakami (2012) adverte para a quebra do modelo onde o software era construído por apenas um desenvolvedor, para o modelo onde a colaboração é fundamental. A comunidade de OSP é uma excelente oportunidade para os estudantes aprenderem a lidar com a diversidade (YAMAKAMI, 2012). Além dos usuários, os estudantes precisam interagir com desenvolvedores distribuídos geograficamente e temporalmente, de diferentes culturas e comportamento, com diferentes níveis de conhecimento e experiência (BUDD, 2009). A comunidade por si mesma é um ambiente de aprendizagem. Tanto o trabalho de Oreg e Nov (2008) quanto o de Ghosh et al. (2002) ressaltam que um dos motivos pelos quais indivíduos participam das comunidades OSP é o desenvolvimento próprio, isto é, para aprender e desenvolver novas habilidades. Ghosh et al. (2002) ainda destacam que mais da metade (67,2 % dos entrevistados em sua pesquisa) continuam participando da comunidade para compartilhar conhecimento e habilidades. A interação com desenvolvedores mais experientes permite a visualização de como eles resolvem os problemas e como eles interagem com outros desenvolvedores, ou mesmo, podem dar respostas de como o desenvolvedor que está contribuindo pode melhorar seu trabalho (SPINELLIS, 2006). Ademais, Spinellis (2006) aponta o desenvolvimento de habilidades de comunicação escrita uma vez que é preciso reportar requisitos, *design*, detalhes de implementação, descrições de erros de modo preciso e objetivo, por meio de ferramentas como correio eletrônico, mensagens instantâneas, listas de discussão, *wikis*, *bug trackers*, etc.

Outros argumentos que podem ser fornecidos para o emprego de OSP são a motivação do estudante pela possível visibilidade proporcionada aos resultados de seu trabalho (RAJ; KAZEMIAN, 2006) e a discussão menos abstrata sobre tópicos como, por exemplo, direitos autorais, patentes, pirataria e ética (CARRINGTON, 2003) (YAMAKAMI, 2012). Alguns pesquisadores ainda ressaltam que a participação dos estudantes em projetos de código aberto permitem diferentes estilos de aprendizagem: colaborativa (CARRINGTON, 2003) (PAPADOPOULOS; STAMELOS; MEISZNER, 2012), baseada em problemas (PAPADOPOULOS; STAMELOS; MEISZNER, 2012), baseada em projetos (PAPADOPOULOS; STAMELOS; MEISZNER, 2012), onde o estudante precisa

primordialmente ser autodidata (PAPADOPOULOS; STAMELOS; MEISZNER, 2012) (AUER; JUNTUNEN; OJALA, 2011) (BUDD, 2009).

Questões de Pesquisa

Portanto, considerando: (i) a necessidade de experiência prática em projetos reais para os estudantes de computação que futuramente irão trabalhar na indústria de software; (ii) que a utilização de projetos proprietários depende de contratos de cooperação com a indústria; (iii) que nem sempre estes contratos são viáveis, por razões acadêmicas ou de negócio das empresas e (iv) os vários argumentos favoráveis aqui apresentados para a utilização de projetos de código aberto; esta abordagem aparece como possibilidade importante a ser ponderada para a educação em engenharia de software (EES).

Nesse contexto, surge uma questão: O uso de projetos de código aberto representa uma experiência apropriada de mundo real a ser utilizada na educação formal em engenharia de software?

Esta questão pode desdobrar-se nas seguintes questões de pesquisa:

RQ1. Como projetos de código aberto podem ser incorporados como uma abordagem de ensino-aprendizagem de engenharia de software no ambiente acadêmico?

RQ2. Quais objetivos de aprendizagem de engenharia de software podem ser cobertos através dessa abordagem? Em que nível podem ser atingidos?

RQ3. Quais as consequências para a aprendizagem de engenharia de software, por parte dos estudantes, quando projetos de código aberto são empregados no processo ensino-aprendizagem?

RQ4. Que lacunas identificadas pela indústria na formação dos estudantes a abordagem ajuda a suprir?

Essas questões, além de motivarem e acompanharem o presente trabalho de pesquisa, conduziram à formulação das seguintes hipóteses:

- A adoção de projetos de código aberto no processo de ensino-aprendizagem da educação formal em engenharia de software propicia alcançar um subconjunto dos objetivos de aprendizagem propostos para essa área de conhecimento em nível de graduação.
- A adoção de projetos de código aberto ajuda a suprir um subconjunto das deficiências apresentadas pelos profissionais recém-formados atuantes na indústria de software.
- A adoção de projetos de código aberto contribui para a preparação do estudante para o mercado de trabalho em virtude da experiência de mundo real proporcionada.

Assim, estabelecemos a seguinte tese a ser defendida:

Projetos de código aberto são eficazes para a aprendizagem formal de engenharia de software em termos de permitir alcançar um subconjunto dos objetivos de aprendizagem, ajudar a suprir um subconjunto de deficiências de formação apontadas pela indústria e contribuir para a preparação do estudante para o mercado de trabalho por representarem uma experiência de mundo real.

1.2 OBJETIVOS

O objetivo geral deste trabalho foi investigar como projetos de código aberto têm sido incorporados na educação formal em engenharia de software e a eficácia desta abordagem, com relação ao alcance de um subconjunto de objetivos de aprendizagem, a ajudar a suprir um subconjunto de deficiências de formação apontadas pela indústria e a contribuir para a preparação do estudante para o mercado de trabalho por representar uma experiência de mundo real.

A adoção de OSP a que se refere esta pesquisa compreende alguma forma de envolvimento dos estudantes com o mesmo, seja analisando o código fonte, projetando e implementando mudanças, testando, corrigindo erros, documentando, seja engajando-se de algum modo com a comunidade.

Ao longo da pesquisa, focamos no uso da abordagem em disciplinas de graduação. Primeiramente, porque almejávamos avaliar se a abordagem era adequada como exemplo prático na formação essencial do estudante, isto é, no desenvolvimento do conteúdo e das competências presentes nas disciplinas de engenharia de software. Segundo, porque pretendíamos verificar a influência de projetos de código aberto como experiência de mundo real e, para estudantes de pós-graduação, tal influência poderia não ser percebida, já que é grande a probabilidade destes terem tido alguma experiência prévia na indústria.

É importante ressaltar que não foi objetivo desta pesquisa comparar a abordagem a outras e assim identificar "a melhor solução". Mas sim, analisar "uma possível solução", uma vez que, na área de educação, o sucesso de qualquer solução depende do contexto onde a mesma é empregada. Buscamos entender e descrever o impacto da abordagem na aprendizagem dos estudantes, de maneira a prover informações para que professores da área possam avaliar se a mesma pode atender às suas necessidades.

O objetivo geral foi detalhado nos seguintes objetivos específicos:

Ob1. Identificar como projetos de código aberto têm sido incorporados como uma abordagem de ensino-aprendizagem de engenharia de software no ambiente acadêmico;

Ob2. Definir quais objetivos de aprendizagem podem ser cobertos por essa abordagem;

Ob3. Identificar os resultados reportados pelos estudos relacionados a esta abordagem;

Ob4. Avaliar a possibilidade do alcance de um subconjunto dos objetivos de aprendizagem, pelos estudantes, com a adoção da abordagem;

Ob5. Explorar as consequências para a aprendizagem de engenharia de software pelos estudantes quando projetos de código aberto são empregados no processo ensino-aprendizagem;

Ob6. Investigar se a adoção de projetos de código aberto permite que o estudante faça a conexão entre teoria e prática;

Ob7. Investigar a percepção do estudante sobre seu contato com o projeto de código aberto como uma experiência de mundo real;

Ob8. Investigar se e como o uso do projeto de código aberto permite ao estudante sentir-se preparado para o mercado de trabalho;

Ob9. Investigar se a abordagem ajuda a suprir algumas das lacunas de formação identificadas pela indústria.

Cabe destacar que, mesmo com o caráter exploratório do objetivo Ob5, com vistas a entender e descrever o impacto da abordagem na aprendizagem dos estudantes, decidimos investigar explicitamente algumas possíveis consequências, a exemplo dos objetivos Ob6, Ob7 e Ob8.

1.3 JUSTIFICATIVAS

Conforme mencionado na contextualização deste trabalho, a academia não tem conseguido suprir a demanda de profissionais capacitados para a indústria de modo adequado, principalmente considerando a necessidade de práticas que representem a realidade da indústria. Por outro lado, Lethbridge et al. (2007) destacam que a formação inadequada dos profissionais também contribui para os problemas de qualidade e orçamento existentes nos projetos de software, uma vez que muitos deles são causados por erro humano ou falta de competência. Os autores defendem que uma forma de aprimorar as práticas de desenvolvimento de software, obter melhores produtos de software e consequentemente, conquistar avanços na área de engenharia de software, seria através da melhoria da educação em engenharia de software.

Para melhorar a educação, é essencial desenvolver pesquisas sobre como educar melhor. De acordo com Moreno et al. (2012), como educar futuros engenheiros de software para que os mesmos executem suas atividades apropriadamente e eficientemente ainda é uma questão crucial, principalmente, ao refletir sobre a importância das práticas da engenharia para o mundo da tecnologia da informação. Radermacher e Walia (2013) ainda ressaltam a importância de considerar tanto a expectativa da academia quanto da indústria (quais práticas das indústria não estão sendo ensinadas – por exemplo), para determinar a qualidade da educação que os estudantes estão recebendo.

Todavia, a maioria dos departamentos de computação não reconhece (ou muitos dos pesquisadores são desafiados pelo fato de não serem reconhecidas) pesquisas em educação como pesquisas legítimas (LETHBRIDGE et al., 2007) (MEAD, 2009). Lethbridge et al. (2007) destacam que esta é uma questão cultural e que está parcialmente relacionada com a qualidade da pesquisa gerada e à falta de conhecimento do que constitui uma boa pesquisa em educação. Os autores ressaltam que grande parte dos estudos educacionais submetidos a algum tipo de revisão apresenta uma ou mais das seguintes falhas: a fundamentação da literatura é ignorada ou não é citada; hipóteses e métodos não são

descritos adequadamente e a avaliação dos resultados é inadequada. Revisando a literatura existente sobre educação em ciência da computação, Holmboe, McIver e George (2001) identificaram estudos importantes, porém com objetivos limitados a descrições de cursos, desenvolvimento de ferramentas e aprendizagem auxiliada por computador.

Para que as pesquisas na área de educação em engenharia de software sejam mais respeitadas, ou seja, tenham seu valor reconhecido, é imprescindível que a qualidade dos estudos seja refinada. Portanto, torna-se necessário que pesquisadores dediquem-se ao tema e que resultados obtidos sejam baseados em evidências.

Os educadores de engenharia de software ou ciência da computação usualmente não recebem qualquer formação na área pedagógica ou didática, já que não correspondem ao alvo destes cursos. O ideal seria que, paralelamente à sua área de especialização, esses educadores aprendessem sobre teorias de ensino e aprendizagem e como aplicá-las. Entretanto, comumente, os professores estão sobrecarregados com várias outras atividades além do dia-a-dia de sala de aula, como a orientação e execução de projetos de pesquisa, extensão e tarefas administrativas dentro das próprias instituições de ensino. Consequentemente, muitas vezes, não há tempo para o estudo de novas abordagens de ensino. Adicionalmente, os objetivos de aprendizagem e os recursos utilizados (e.g., linguagens de programação, ferramentas, entre outros) mudam demasiadamente mais rápido que em outras disciplinas (SIMON; PARRIS; SPACCO, 2013). Sendo assim, pesquisadores dedicados à educação em engenharia de software podem aprender mais sobre teorias de aprendizagem e com isso estruturar melhor as pesquisas sobre o tema, entender as contribuições de cada estudo e compreender onde concentrar os esforços para a melhoria. Ademais, podem refletir não somente sobre as suas práticas de ensino, mas sobre as de outros professores (HOLMBOE; MCIVER; GEORGE, 2001).

Considerando a necessidade de evidências concretas, o método de ensino a ser aplicado deve ser baseado em evidências reais e não em opiniões próprias (LETHBRIDGE et al., 2007). Todavia, segundo Holmboe, McIver e George (2001), muitos estudos apresentam seus métodos de ensino como o caminho para que os estudantes alcancem a excelência, contudo, apesar de algumas vezes apresentarem tentativas de justificar suas afirmações, frequentemente tais afirmações são resultado de argumentos e reflexões dos autores, sem que evidências sejam apresentadas.

De acordo com Navarro e Hoek (2007), se mais pesquisadores na área de educação em engenharia de software avaliassem suas abordagens apropriadamente, a comunidade seria provida com evidências mais convincentes sobre a eficácia de cada abordagem e, consequentemente, o compartilhamento e adoção destas abordagens seriam encorajadas. Os autores ainda complementam que isto promoveria uma compreensão geral de como a engenharia de software poderia ser ensinada e aprendida baseando-se em experiências efetivas. Como resultado, a área de educação em engenharia de software iria progredir e se tornar mais efetiva em preparar os estudantes para suas futuras carreiras.

Portanto, é necessário aperfeiçoar a educação em engenharia de software a fim de melhor capacitar os futuros profissionais de software. Este aperfeiçoamento é possível através do desenvolvimento de pesquisas voltadas para este tema. Devido à falta de tempo de muitos professores envolvidos em outras atividades, é importante que pesquisadores dediquem-se a esta área.

Preocupando-se com essas questões, o presente trabalho investigou a adoção de projetos de código aberto no processo de ensino-aprendizagem de engenharia de software, como experiência de mundo real para os estudantes. Tal estudo é relevante para pesquisadores e professores de engenharia de software interessados em melhorar suas práticas de ensino, a partir do uso de exemplos reais de projetos de software, sem que existam grandes problemas com relação à privacidade de código, perda de prazo, qualidade e desistência dos estudantes. A ocorrência de um painel discutindo "como usar projetos de código aberto na educação" (BISHOP et al., 2016) na conferência SIGCSE'16'[1] demonstra o interesse da comunidade científica sobre esta abordagem.

Os objetivos traçados para esta pesquisa permitiram obter uma visão de como os projetos estão sendo incorporados, exemplos de estratégias empregadas e os resultados dessa incorporação para a aprendizagem do estudante. Detalhamos os objetivos de aprendizagem em engenharia de software que poderiam ser alcançados por meio da abordagem, apresentamos evidências sobre o alcance de alguns desses objetivos para as áreas de conhecimento da engenharia de software investigadas e as consequências da abordagem para a aprendizagem dos estudantes, sob a perspectiva dos próprios estudantes. Tais resultados podem direcionar futuras pesquisas, inclusive de outros pesquisadores também interessados no tema, e proveem evidências para que educadores tomem suas decisões quanto à adoção ou não da abordagem.

1.4 CONTRIBUIÇÕES

Este trabalho resultou nas seguintes contribuições:

1. Uma visão geral e abrangente de como projetos de código aberto têm sido aplicados no processo de ensino-aprendizagem de engenharia de software. Esta visão é resultado do mapeamento sistemático realizado que sumariza e categoriza as diversas iniciativas existentes.

2. Identificação dos objetivos de aprendizagem em engenharia de software a partir da correlação entre corpo de conhecimento do SWEBOK (BOURQUE; FAIRLEY, 2014), tópicos e resultados de aprendizagem relacionados à engenharia de software do currículo de referência de ciência da computação (ACM AND IEEE, 2013) e as lacunas na formação dos estudantes recém-formados relatadas pela indústria de software.

3. Identificação de quais objetivos de aprendizagem em engenharia de software têm sido alvos da utilização de projetos de código aberto, segundo a literatura existente, e quais ainda podem ser favorecidos por esta utilização.

4. Síntese dos resultados obtidos pelos diversos trabalhos publicados relacionados à adoção de projetos de código aberto na educação em engenharia de software.

[1] Importante conferência anual sobre educação em computação promovida pelo *Special Interest Group on Computer Science Education* (www.sigcse.org/).

5. Evidências sobre a utilização de projetos de código aberto para a aprendizagem de evolução e manutenção de software, testes de software e requisitos de software, a partir da realização de três estudos de caso distintos.

6. Análise sobre a possibilidade de alcance de um subconjunto dos objetivos de aprendizagem por parte dos estudantes, com a adoção de projetos de código aberto na educação em engenharia de software.

7. Uma visão sobre as possíveis consequências para a aprendizagem dos estudantes com a adoção de projetos de código aberto na educação em engenharia de software.

8. Análise do suporte da adoção de projetos de código aberto na educação em engenharia de software à solução de deficiências na formação dos estudantes recém-formados apontadas pela indústria.

Outras contribuições indiretas foram:

- Propostas de incorporação de projetos de código aberto para a aprendizagem de evolução e manutenção de software, testes de software e requisitos de software.

- Visão sobre as possíveis dificuldades a serem enfrentadas pelos estudantes e professor com a aplicação da abordagem.

- Buscar a melhoria do processo ensino-aprendizagem através do envolvimento do estudante em projetos reais.

- Verificar as consequências do envolvimento do estudante em projetos reais para a sua aprendizagem.

- Ressaltar a importância da pesquisa em educação em engenharia de software para o avanço do processo ensino-aprendizagem nessa área.

- Colaborar com o aperfeiçoamento das pesquisas em educação em engenharia de software, seja pelo cuidado com a metodologia da pesquisa aplicada, seja pela apresentação de evidências.

- Alertar que o aperfeiçoamento da educação em engenharia de software pode acarretar na produção de softwares de melhor qualidade futuramente.

Algumas das contribuições, já resumimos em publicações:

1. NASCIMENTO, D. M. C. ; CHAVEZ, C. F. G. ; BITTENCOURT, R. A. . Learning Software Evolution: Curriculum, Approaches and an Experience Report. In: FEES 2013 - Fórum de Educação em Engenharia de Software, 2013, Brasília. Anais do XXVII Simpósio Brasileiro de Engenharia de Software, 2013[2].

[2]Elaboramos este artigo ainda na fase da definição dos objetivos da pesquisa. Influenciou-nos a considerar a utilização dos currículos de referência no estabelecimento dos objetivos de aprendizagem e na realização do primeiro estudo de caso

2. NASCIMENTO, D. M. C. ; COX, K. K. ; ALMEIDA, T. ; SAMPAIO, W. ; BITTENCOURT, R. A. ; SOUZA, R. R. G. ; CHAVEZ, C. F. G. . Using Open Source Projects in Software Engineering Education: A Systematic Mapping Study. In: FIE 2013 - 43rd Annual Frontiers In Education Conference, 2013, Oklahoma City. Proceedings of the 43rd Annual Frontiers In Education Conference, 2013.

3. NASCIMENTO, D. M. C.; BITTENCOURT, R. A. ; CHAVEZ, C. F. G. Open Source Projects in Software Engineering Education: A Mapping Study. Computer Science Education, v.25, p. 67-114, 2015.

Convém ressaltar que as demais contribuições permitem-nos elaborar outros artigos, que por exemplo reportem os resultados obtidos com a síntese das evidências das publicações relacionadas à adoção de projetos de código aberto na educação em engenharia de software, reportem os resultados obtidos com a adoção de projetos de código aberto em cada estudo de caso específico (para a aprendizagem de evolução e manutenção de software, testes de software e requisitos de software) e, por fim, que descrevam as consequências da adoção de projetos de código aberto como exemplo de projeto real para a aprendizagem dos estudantes na educação em engenharia de software.

Possíveis títulos para estes artigos seriam:

- 'Síntese dos resultados reportados sobre a adoção de projetos de código aberto na educação em engenharia de software';
- 'O uso de projetos de código aberto no ensino de evolução de software: qual a percepção dos estudantes e do professor?';
- 'Qual a aprendizagem dos estudantes ao implementar testes automatizados em um projeto de código aberto?';
- 'Aprendendo sobre requisitos de software a partir de um projeto de código aberto: uma abordagem diferente!';
- 'Consequências da adoção de projetos de código aberto como exemplo de projeto real para a aprendizagem dos estudantes na educação em engenharia de software'.

1.5 ORGANIZAÇÃO DO DOCUMENTO

Este documento está organizado em dez capítulos. Após a contextualização, definição dos objetivos, contribuições e justificativas para o trabalho, apresentados neste primeiro capítulo, o segundo capítulo discorre sobre a fundamentação teórica para adoção de projetos de código aberto na educação em engenharia de software. O Capítulo 3 trata dos objetivos de aprendizagem em engenharia de software que podem ser trabalhados com a adoção de projetos de código aberto. O Capítulo 4 descreve os procedimentos de investigação utilizados ao longo do trabalho. O Capítulo 5 resume como os projetos de código aberto têm sido incorporados na educação em engenharia de software e quais as evidências existentes. Os três capítulos seguintes descrevem o contexto e os resultados

encontrados nos estudos de caso realizados. O nono capítulo traz a triangulação entre todos os achados, identificando as interseções e particularidades de cada investigação, discutindo-os à luz dos objetivos traçados para a pesquisa e da literatura existente. Finalmente, o último capítulo apresenta as conclusões para o trabalho e a perspectiva de trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Ensinar e aprender são ações distintas, todavia, enquanto a aprendizagem pode ocorrer dissociada da operação de ensinar, o ensino não faz sentido sem que um aprendiz seja envolvido (MOON, 2004). O ensino ou instrução corresponde ao que pode ser feito, elaboração e uso de recursos, para que a aprendizagem seja facilitada (UDEN; BEAUMONT, 2006). Um dos deveres da educação formal é mediar, guiar e favorecer a aprendizagem. O professor é responsável por prover o estímulo e a qualidade desse estímulo depende da abordagem pedagógica utilizada (GENTRY, 1990). Portanto, compreender o que é aprendizagem, como ocorre, quais os fatores que podem influenciar o processo de aprendizagem, permite ao professor tomar ações que propiciem a aprendizagem (UDEN; BEAUMONT, 2006).

Ao longo do último século, psicólogos e educadores formularam diferentes teorias para explicar o processo de aprendizagem, porém eles não chegaram a um consenso (UDEN; BEAUMONT, 2006). Justamente porque corresponde a um processo complexo, uma única teoria não é suficiente para explica-lo por completo (DRISCOLL, 1993 apud UDEN; BEAUMONT, 2006, p. 4). Em verdade, a aprendizagem ocorre das mais variadas formas e em diferentes situações (ANZAI; SIMON, 1979). Contudo, como ressaltado por Lefrançois (2012, p. 10), "nenhuma teoria é totalmente incorreta", de modo que, por meio delas, pode-se simplificar, organizar os pensamentos e prever os resultados. Por conseguinte, as teorias de aprendizagem proveem a base para predizer os efeitos prováveis de determinadas ações.

A adoção de projetos de código aberto corresponde a um recurso didático a ser empregado na educação formal, a fim de favorecer a aprendizagem de engenharia de software. O objetivo desse capítulo é discutir quais fundamentos teóricos a respeito do processo de ensino-aprendizagem apoiam essa abordagem. Por conseguinte, a partir da definição de aprendizagem e da compreensão de um modelo simplificado do processo de aprendizagem, discorremos sobre como a abordagem estudada pode influenciar a aprendizagem. Cabe salientar que o capítulo não tem a intenção de debater sobre as concepções das diversas teorias de aprendizagem, quais as vantagens e desvantagens de cada uma.

2.1 DEFINIÇÃO DE APRENDIZAGEM

Alguns autores definem a aprendizagem como uma mudança relativamente permanente no potencial de comportamento (LEFRANÇOIS, 2012) (SCHUNK, 1991 apud UDEN; BEAUMONT, 2006, p. 3) ou no potencial de desempenho (DRISCOLL, 1993 apud UDEN; BEAUMONT, 2006, p. 3) de um indivíduo, como consequência de alguma forma de experiência ou interação com o ambiente, que não seja resultado de cansaço, drogas ou doença (LEFRANÇOIS, 2012) (SCHUNK, 1991 apud UDEN; BEAUMONT, 2006, p. 3).

Com a aprendizagem, existe uma mudança no nível de conhecimento ou habilidade do indivíduo (UDEN; BEAUMONT, 2006). Lefrançois (2012) destaca que a aprendizagem é um processo neurológico interno, que acontece em seres humanos e animais, como resultado da experiência e que a mudança no comportamento é apenas uma evidência de que a mesma ocorreu. O autor ainda explica que o uso do termo "potencial" na definição de aprendizagem deve-se à necessidade de uma oportunidade, para que a evidência da aprendizagem apresente-se, uma vez que, várias vezes a aprendizagem permanece latente.

Outro elemento importante na definição de aprendizagem, principalmente para o nosso projeto de pesquisa, é a "experiência". A aprendizagem resulta da experiência (UDEN; BEAUMONT, 2006) (LEFRANÇOIS, 2012). Essa experiência pode ser o contato com algo, a participação em algo, a exposição a algum evento interno ou externo (LEFRANÇOIS, 2012). Esse elemento bem como outros elementos importantes para a aprendizagem serão discutidos nas próximas seções.

2.2 A METÁFORA DO PROCESSAMENTO DA INFORMAÇÃO

Além da experiência, Lefrançois (2012) ressalta a importância da atenção e memória para a aprendizagem. O autor lembra que a atenção facilita a aprendizagem e que para a ocorrência de evidências da aprendizagem, é preciso que algo tenha sucedido na memória – alguma coisa precisa ser memorizada como consequência da experiência. Como a memória representa disponibilidade da informação, a disponibilidade implica na capacidade de recuperar habilidades ou informações previamente adquiridas (LEFRANÇOIS, 2012). Assim, o autor aponta que "estudar a memória é, na verdade, outra forma de estudar aprendizagem" (LEFRANÇOIS, 2012, p. 303).

De acordo com as teorias de aprendizagem cognitivas, a aprendizagem é descrita como uma mudança do conhecimento armazenado na memória (NEWBY et al., 1996 apud UDEN; BEAUMONT, 2006, p. 9). A aprendizagem ocorre quando a informação é capturada do ambiente, processada, armazenada na memória e evidenciada por meio de alguma capacidade apresentada (UDEN; BEAUMONT, 2006). Daí a utilização da metáfora do processamento de informação para compreender o funcionamento da memória, consequentemente o processo de aprendizagem.

Buscando compreender a imensa complexidade do funcionamento do cognitivo humano, a psicologia constantemente utiliza-se de metáforas (LEFRANÇOIS, 2012). Por meio de metáforas, é possível explicar algo complexo, com modelos mais fáceis de compreender. Várias metáforas têm sido sugeridas para descrever a memória ao longo dos anos, contudo, as características da memória de armazenamento e recuperação popu-

larizaram o uso da metáfora do computador (LEFRANÇOIS, 2012), mais especificamente, a metáfora do processamento da informação. Cabe ressaltar, entretanto, que essa metáfora representa apenas uma simplificação (para descrever o comportamento e facilitar o entendimento) e "não faz jus a algo tão rico e complexo como a memória humana" (LEFRANÇOIS, 2012, p. 303).

Após vários estudos, pesquisadores decidiram por abstrair a memória a partir de três componentes (ATKINSON; SHIFFRIN, 1968 apud LEFRANÇOIS, 2012, p. 307): (i) memória sensorial, também denominada por registro sensorial; (ii) memória de curto prazo (MCP), também conhecida como memória de trabalho ou temporária, e (iii) memória de longo prazo (MLP). Cada um desses componentes cumpre papel específico durante o processamento da informação (ver Figura 2.1).

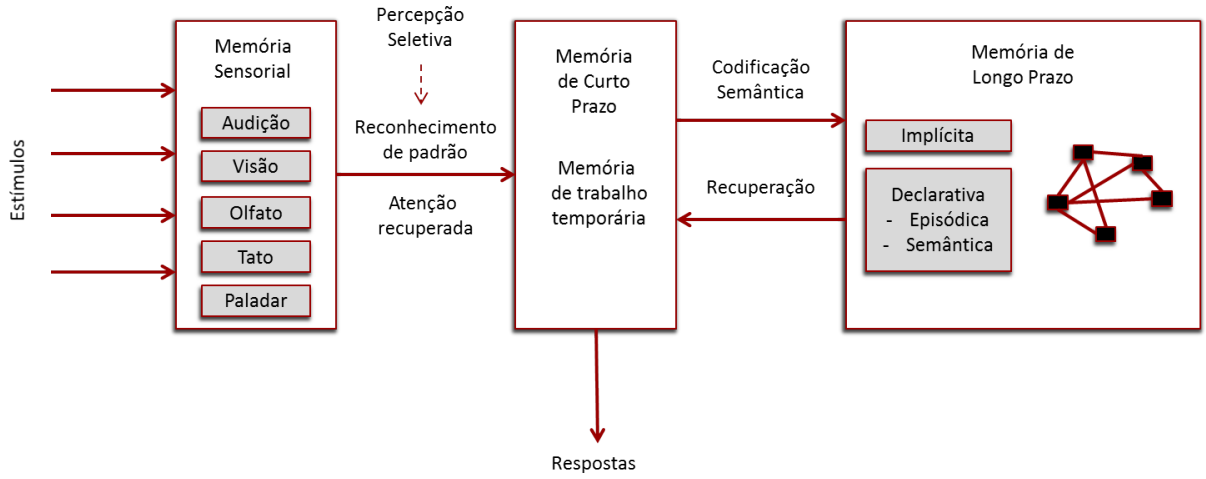

Figura 2.1 Modelo de processamento de informação adaptado de Driscoll (1993 apud UDEN; BEAUMONT, 2006, p. 7)

Inicialmente os estímulos são capturados pela memória sensorial. Existe uma memória sensorial associada a cada um dos sentidos (visão, audição, paladar, tato e olfato), de modo que, a memória sensorial trabalha inconscientemente (diz-se que a memória sensorial precede a atenção (LEFRANÇOIS, 2012)). Quando um padrão é reconhecido, o indivíduo torna-se consciente a respeito do estímulo (a atenção é ativada), então a informação é transferida para a memória de curto prazo. Esse processo é conhecido como percepção seletiva (UDEN; BEAUMONT, 2006). A MCP conserva a informação por um curto espaço de tempo (alguns segundos), de modo que na ausência de recapitulação, a informação será sobreposta e não estará mais disponível, isto é, não será mais lembrada ou recuperada. Lefrançois (2012) descreve a MCP como aquilo sobre o quê o indivíduo está consciente em um dado momento e é equivalente à amplitude de sua atenção, de maneira que, é facilmente descontinuada por eventos externos ou internos. Enfim, a informação é novamente transformada, pelo processo denominado de codificação semântica, e transferida para a memória de longo prazo (UDEN; BEAUMONT, 2006). Nessa etapa ocorre um processamento mais profundo, com a nova informação sendo integrada à in-

formação existente, envolvendo atividades como análise, reconhecimento de significado e organização (LEFRANÇOIS, 2012). Assim, durante esse processamento, informações também são recuperadas da MLP para a MCP e quando necessário, respostas são geradas para o meio externo (UDEN; BEAUMONT, 2006). É importante dizer que informações, para serem relembradas, precisam ser transferidas da MCP para a MLP (UDEN; BEAUMONT, 2006). A MLP corresponde ao que a pessoa consegue lembrar depois de certo espaço de tempo (relativamente permanente - pode durar minutos ou anos) e compreende o que pode ser retido nas experiências educacionais (LEFRANÇOIS, 2012).

Existem dois tipos de memória de longo prazo (LEFRANÇOIS, 2012): memória implícita ou não declarativa, e memória explícita ou declarativa. A memória implícita representa efeitos de aprendizagem não verbalizáveis, isto é, não pode ser explicada por meio de palavras. É inconsciente e geralmente corresponde às habilidades motoras, como por exemplo, andar de bicicleta. Já a memória explícita ou declarada corresponde ao que é consciente, representando informação que pode ser recuperada (expressas em palavras). A memória explícita pode ainda ser subdividida em memória episódica e semântica (LEFRANÇOIS, 2012) (UDEN; BEAUMONT, 2006). A memória episódica corresponde à memória relacionada a eventos específicos, naturalmente compreende os fatos vividos por cada indivíduo. Enquanto que a memória semântica representa todo o conhecimento sobre o mundo, o que pode ser lembrado, independentemente de como foi aprendido. A memória episódica apresenta-se como mais vulnerável ao esquecimento e à distorção (LEFRANÇOIS, 2012). Enquanto a memória semântica aparentemente não sofre influência da memória episódica, esta última parece depender da primeira (LEFRANÇOIS, 2012).

Atualmente, em geral, modelos abstraem a estrutura da MLP através de representações associativas, de maneira que os itens de informação armazenados na memória relacionam-se uns com os outros nas mais variadas formas (LEFRANÇOIS, 2012). Tais modelos correspondem principalmente a modelos cognitivos, como por exemplo, os sistemas de codificação em categorias de Bruner, os esquemas de Piaget e os modelos conexionistas (redes neurais) (LEFRANÇOIS, 2012). Moon (2004) ressalta que vários autores utilizam o termo 'estrutura cognitiva' para descrever o que o aprendiz sabe em um determinado momento.

De modo simplificado, podemos descrever duas perspectivas de como as informações são acrescentadas à estrutura da MLP, isto é, de como ocorre a aprendizagem.

Para explicar a primeira perspectiva, Moon (2004) faz uma analogia com a construção de uma parede de tijolos. Nessa perspectiva, o conhecimento é 'acumulado', como se uma parede de tijolos estivesse sendo construída. O conteúdo ensinado é absorvido pelo estudante e retido da mesma forma que foi ensinado, com apenas poucas modificações devido a memória. Posteriormente, se a aprendizagem foi efetiva, o estudante será capaz de apresentar o conteúdo, de forma similar ao apresentado pelo professor.

A segunda perspectiva segue a visão construtivista para a aprendizagem (MOON, 2004), denominada "construção do conhecimento" por Piaget e "construção do significado" por Bruner (LEFRANÇOIS, 2012). Nessa perspectiva, o conhecimento é construído a partir do conhecimento prévio existente. A cada nova experiência, o conhecimento prévio relacionado é recuperado, comparado à nova informação, ajustado e novamente

armazenado, não mais com o formato anterior, mas com algo diferente, modificado, à depender da compreensão gerada pelo aprendiz. Dessa maneira, cada indivíduo constrói o seu conhecimento, o significado das coisas lhe é peculiar, uma vez que apresenta uma interpretação individual da experiência (UDEN; BEAUMONT, 2006), a depender das experiências anteriores, crenças próprias e da sua capacidade cognitiva (estrutura mental existente (UDEN; BEAUMONT, 2006)).

Piaget descreve a perspectiva construtivista para a aprendizagem através do processo de assimilação e acomodação (LEFRANÇOIS, 2012). A assimilação envolve a recuperação do conhecimento prévio importante, para responder a situações correntes usando esse conhecimento, enquanto que, a acomodação representa a mudança na compreensão diante dessas situações (LEFRANÇOIS, 2012). Em outras palavras, o indivíduo, ao interagir com o ambiente, assimila os aspectos da experiência à estrutura cognitiva existente (objetos ou situações são assimilados ao esquema existente) e acomoda (ajusta) a estrutura às novas informações decorrentes da experiência (LEFRANÇOIS, 2012). Piaget denomina de "interiorização" o processo de formação de esquemas mentais, que representam os conceitos, eventos e atividades do mundo real, de modo que, a interiorização é o fundamento para a aprendizagem cognitiva (LEFRANÇOIS, 2012).

Rejeitando a perspectiva de 'acumulação do conhecimento', Moon (2004) destaca ainda três pontos importantes para a perspectiva construtivista, são eles: (i) não somente, o que está sendo aprendido pode gerar mudanças no que já se sabia, mas também, o que está sendo aprendido pode ser influenciado pelo que já se sabia; (ii) a estrutura cognitiva é flexível e pode modificar-se inclusive sem a adição de material de aprendizagem externo ao indivíduo e (iii) o estado da estrutura cognitiva do indivíduo influencia o que será selecionado e absorvido pelo mesmo, em determinado momento. Semelhantemente, Lefrançois (2012) aponta que o processo de assimilação e acomodação também é orientado pela estrutura cognitiva existente e é resultado das demandas do ambiente.

É importante destacar que outros fatores como condições emocionais, motivação, contexto e interação social também influenciam a aprendizagem. Na próxima seção, discutiremos elementos relevantes para a aprendizagem que são proporcionados pela adoção de projetos de código aberto.

2.3 ELEMENTOS RELEVANTES PARA A APRENDIZAGEM

Através dessa visão simplificada do processo de aprendizagem, podemos apontar três elementos relevantes para a aprendizagem, são eles: experiência, engajamento e significância.

Importância da experiência

Como vimos, o processo de aprendizagem inicia-se por meio de estímulos. Esses estímulos são gerados por cada experiência vivenciada pelo indivíduo. Consequentemente, podemos dizer que aprendizagem é resultado da experiência vivida pelo mesmo. Boud, Cohen e Walker (1993 apud MOON, 2004) frisam que a experiência é a base e o estímulo para toda a aprendizagem. Uma aula expositiva, a leitura do livro texto, a discussão de determinado conteúdo, atividades práticas, a visita a um museu são exemplos de experiências.

Moon (2004), distingue entre experiência externa e experiência interna. De acordo

com a autora, a primeira representa o que é externo, o que é vivenciado pelo aprendiz, seja um objeto, um conceito, uma imagem, etc. Por outro lado, a experiência interna significa o que o aprendiz recupera de sua estrutura cognitiva para a situação de aprendizagem atual, ou seja, o conjunto de experiências anteriores que são importantes para a situação presente. Por conseguinte, a aprendizagem ocorre por meio da comparação entre a experiência externa e a experiência interna corrente, sendo resultado de variações existentes entre as mesmas (MOON, 2004). Jarvis (1992 apud MOON, 2004) ainda ressalta que a experiência prévia é quem direciona como o aprendiz responde à experiência presente.

Enfim, a experiência do aprendiz é o que determina o que ele vai aprender, no entanto, o professor pode influenciar a aprendizagem, especificando as experiências externas pelas quais o aprendiz irá passar (MOON, 2004).

É relevante dizer que nem toda experiência traz bons resultados para a aprendizagem (DEWEY, 1938) (GENTRY, 1990) (MOON, 2004). A experiência precisa ser vívida, animada, interessante e deve estar conectada com experiências futuras, em especial com as situações de vida fora da escola (DEWEY, 1938).

Engajamento cognitivo

Vários autores para enfatizar a importância da participação ativa dos estudantes (SILBERMAN, 1996) ou da aprendizagem através do experimentar / fazer (GENTRY, 1990) (NIU, 2012) empregam as declarações de Confúcio:

"O que eu ouço, eu esqueço. O que eu vejo, eu lembro. O que eu faço, eu compreendo." (tradução nossa)

Gentry (1990, p. 9) acrescenta Sófocles:

"O indivíduo deve aprender fazendo, para você que pensa que sabe - você não tem certeza, até você experimentar." (tradução nossa)

Enquanto que a *Northern Illinois University* (NIU, 2012) agrega Benjamin Franklin:

"Diga-me e eu esqueço, ensine-me e eu lembro, envolva-me e eu aprenderei." (tradução nossa)

E John Dewey:

"Existe uma relação íntima e necessária entre o processo de experiência real e educação" (tradução nossa)

Engajamento cognitivo refere-se à qualidade do engajamento psicológico do estudante nas tarefas acadêmicas (DAVIS; SUMMERS; MILLER, 2012). Compreende a necessidade de processamento do conteúdo da lição por parte do aprendiz, de maneira que, tal engajamento pode ser superficial ou profundo (UDEN; BEAUMONT, 2006). Assistir a uma

aula expositiva, assistir a uma demonstração, descrever determinado conteúdo com as suas próprias palavras, dar exemplos, resolver exercícios, discutir ideias sobre o assunto, apresentar sua opinião usando argumentação, resolver problemas reais são exemplos de formas gradativas de engajamento. Podemos dizer que a aprendizagem é função do engajamento cognitivo do estudante, isto é, a aprendizagem aumenta em decorrência do aumento da qualidade do engajamento cognitivo (UDEN; BEAUMONT, 2006). Hannafin (1989 apud UDEN; BEAUMONT, 2006) indica que as estratégias de engajamento devem promover a manipulação da informação, ao invés da memorização. Enfim, Uden e Beaumont (2006) ressaltam que o engajamento em atividades cognitivas complexas gera uma aprendizagem útil e autêntica. Ademais, o engajamento por meio de atividades práticas possibilita a ocorrência de erros e enganos, que também são importantes para a aprendizagem.

Conteúdo significativo

O outro elemento importante para a aprendizagem está relacionado a quanto o conteúdo é significativo para o aprendiz.

Esse elemento participa do processo de aprendizagem desde o início, com a percepção ou seleção da informação que será processada. Como mencionado anteriormente, somente o que é significativo para o aprendiz é capturado. A informação precisa ser percebida ou selecionada para que a aprendizagem ocorra (UDEN; BEAUMONT, 2006).

No segundo momento, conforme a percepção construtivista exposta, a informação capturada é organizada e integrada à estrutura cognitiva corrente, de modo que a compreensão corresponde à integração da nova informação com o conhecimento prévio existente e esta compreensão resulta na construção de um novo significado.

De acordo com Lefrançois (2012), a compreensão influencia a memória de longo prazo e que quanto mais significativo for o conteúdo, mais facilmente ele será lembrado. Essa seria a explicação para a vulnerabilidade da memória episódica, enquanto que, quando um evento possui algum significado relacionado, é mais difícil que ele seja esquecido. A aprendizagem é efetiva quando o aprendiz consegue organizar a informação identificando relações lógicas no conteúdo (UDEN; BEAUMONT, 2006).

Podemos frisar ainda dois pontos com relação à construção do significado. O primeiro refere-se à importância da experiência também para a construção do significado. Autores defendem que experiências concretas são críticas para a aprendizagem significativa (MOON, 2004) (NIU, 2012). Por meio da experiência concreta, o aprendiz pode entender melhor, constatar a importância do conteúdo estudado, avaliar consequências, etc. No segundo ponto, destacamos que apesar da natureza da construção do significado ser individual, toda experiência ocorre dentro de um contexto social e por consequência, cada pessoa não constrói o significado sozinha, mas é fortemente influenciada pelo contexto social e cultural do ambiente de aprendizagem (MOON, 2004). Crenças e valores do indivíduo são geradas a partir do contexto social e cultural em que vive, sendo que essas crenças e valores influenciam a interpretação dos fatos, consequentemente a construção de significados.

Nessa seção, discutimos sobre três elementos importantes para a aprendizagem. A adoção de projetos de código aberto para a aprendizagem em engenharia de software

proporciona cada um deles, uma vez que representa uma experiência concreta equivalente às situações que serão vivenciadas pelo estudante no mercado de trabalho; permite o engajamento ativo do estudante, analisando o código, testando, modificando, documentando, entre outras atividades que podem ser realizadas; enfim, possibilita a partir da prática, a compreensão de conceitos e a visualização da importância dos princípios e técnicas de engenharia de software, ou seja, a construção de significados, que permitem uma aprendizagem mais efetiva.

Na próxima seção, discorreremos sobre teorias que fundamentam o emprego da abordagem.

2.4 TEORIAS QUE FUNDAMENTAM O EMPREGO DE OSP

Nessa seção, selecionamos algumas teorias que fundamentam a adoção de projetos de código aberto na educação de engenharia de software, são elas: teorias construtivistas, aprendizagem social e aprendizagem por experiência.

Teorias Construtivistas

As teorias construtivistas seguem a perspectiva construtivista explicada na Seção 2.2. Piaget, Vygotsky, Bruner, cada um à sua maneira, apresentam teorias onde a aquisição de conhecimento é um processo de desenvolvimento gradual. Uden e Beaumont (2006) resumiram as principais concepções construtivistas para a aprendizagem assim:

- Todo o conhecimento é construído e não transmitido;
- Conhecimento e significado resultam de atividades;
- O conhecimento está distribuído nas pessoas, ferramentas, outros artefatos e elementos culturais;
- Significado surge da interpretação e, portanto, múltiplas perspectivas são reconhecidas;
- Problemas, questões e tarefas autênticas induzem a construção do significado.

Segundo Driscoll (1993 apud UDEN; BEAUMONT, 2006), os objetivos do ensino seguindo a perspectiva construtivista são a solução de problemas, o pensamento crítico, o desenvolvimento da capacidade de discurso e o uso ativo do conhecimento por parte do aprendiz. Existe ainda uma grande ênfase para que a aprendizagem seja contextualizada e as atividades significativas (UDEN; BEAUMONT, 2006). Analisando a adoção de OSP, concluímos que a abordagem possibilita o atendimento desses objetivos.

Aprendizagem Social

Aprendizagem social significa toda a aprendizagem consequente da interação social, ou que, de alguma forma, envolva interação social (SALOMON; PERKINS, 1998 apud LEFRANÇOIS, 2012). Engloba a descoberta de quais comportamentos são esperados

e admissíveis em diferentes situações (LEFRANÇOIS, 2012). Por essa razão, são dependentes da cultura e podem variar de acordo com a idade ou até mesmo com o sexo (LEFRANÇOIS, 2012).

Tanto Vygotsky quanto Piaget atribuem à interação social um papel importante para a aprendizagem. Para Piaget, é através da interação social que as pessoas tornam-se cientes dos pensamentos e sentimentos alheios, constroem preceitos morais e elaboram processos de pensamento lógico próprios (LEFRANÇOIS, 2012). Já Vygotsky diz que graças principalmente à interação social, funções mentais elementares (tendências e comportamento naturais) transformam-se em funções mentais superiores (relativas ao pensamento, como imaginação e solução de problemas) (LEFRANÇOIS, 2012). Todavia, enquanto Piaget fundamenta o desenvolvimento à partir de forças internas, com o aperfeiçoamento das competências individuais, Vygotsky enfatiza as forças externas, em especial a cultura na qual o indivíduo está inserido, para a formação da consciência humana (LEFRANÇOIS, 2012). Para Vygotsky, a cultura exerce uma enorme influência sobre cada indivíduo, aponta o que deve ser aprendido e que competências são necessárias (LEFRANÇOIS, 2012).

Uma contribuição relevante das pesquisas de Vygotsky é a definição do conceito de zona de desenvolvimento proximal. Segundo o pesquisador, a criança consoante ao seu nível de desenvolvimento atual, consegue solucionar alguns tipos de problemas, porém, sob a orientação de um adulto ou com a colaboração de um companheiro, a mesma consegue resolver problemas ainda mais complexos (BARRA, 2014). A zona de desenvolvimento proximal corresponde à distância entre o que o indivíduo pode resolver de forma independente e o que pode resolver com auxílio dos outros, ou seja, o que é potencialmente atingível (BARRA, 2014). É como se fosse um potencial para o desenvolvimento - em contato com alguém com competências mais desenvolvidas, o aprendiz, de acordo com sua capacidade, pode desenvolver novas habilidades (LEFRANÇOIS, 2012). Por conseguinte, o conceito reforça a necessidade da interação estudante-professor e estudante-estudante para a aprendizagem.

Dewey (1938) também aponta a necessidade de interação entre pessoas maduras e pessoas imaturas, porém não com a filosofia tradicional de imposição de conhecimento, métodos e regras, mas através de colaboração, respeitando-se o princípio da aprendizagem através da experiência pessoal.

Considerando a questão da prática profissional, podemos dizer que profissionais mais experientes tendem a apresentar soluções mais eficientes para os problemas, uma vez que a experiência acumulada permite analisar as condições atuais, visualizar as possíveis consequências, inclusive considerando erros anteriores, e por fim, tomar decisões mais acertadas. Portanto, a interação/colaboração com profissionais experientes torna-se essencial para a aprendizagem.

Lefrançois (2012) aponta a aprendizagem por observação como outra forma de aprendizagem social. A aprendizagem por observação corresponde à aprendizagem baseada na imitação, estudada por Albert Bandura (LEFRANÇOIS, 2012). Bandura defende que grande parte do comportamento humano é imitativo: as pessoas aprendem o que é aceitável no que se refere a comportar-se, ou mesmo em questões humanas elementares como falar, comer e vestir-se, ao observar outras pessoas. Portanto, a aprendizagem

também provém de modelos, isto é, da imitação de modelos. O valor dos modelos deriva da sua capacidade informativa (LEFRANÇOIS, 2012). Os modelos informam não somente como fazer certo, mas propiciam visualizar as consequências de certas ações ou decisões, assim sendo, servem de inspiração.

Apesar da aprendizagem pela imitação evocar o modelo de condicionamento operante de Skinner, onde o comportamento depende da manipulação das consequências (LEFRANÇOIS, 2012), Bandura lembra que as pessoas são agentes de suas próprias ações, ou seja, não são somente submissas ao que observam, são capazes de examinar, perceber as implicações das relações entre causas e efeitos, antecipar consequências, refletir e reagir (LEFRANÇOIS, 2012).

Através da observação, o aprendiz pode apropriar-se de uma experiência alheia, refletir sobre as consequências dessa experiência, comparar com as suas próprias experiências e reagir de maneira mais adequada.

No caso do uso da abordagem com OSP, a aprendizagem social pode ocorrer das duas maneiras. Seja pela interação/colaboração com profissionais mais experientes, também participantes das comunidades dos projetos, seja pela observação e reflexão sobre as soluções projetadas por esses profissionais.

Aprendizagem Experiencial

Uma crítica bastante encontrada com relação à educação formal, principalmente nas universidades, diz que o conteúdo estudado na academia não é aplicado na vida real. Muitas vezes, este problema advém das atividades de aprendizagem realizadas pelos estudantes serem distantes da realidade, em outras palavras, das atividades que precisam ser realizadas na prática (SVINICKI; MCKEACHIE, 2011) (UDEN; BEAUMONT, 2006). Por conseguinte, os estudantes não conseguem generalizar, ou melhor, transferir o que foi aprendido para casos reais (SVINICKI; MCKEACHIE, 2011) (UDEN; BEAUMONT, 2006). Este problema tem sido descrito na literatura como *transfer problem* (UDEN; BEAUMONT, 2006), *inert knowledge* (UDEN; BEAUMONT, 2006) ou *situated learning* (LAVE; WENGER, 1991 apud SVINICKI; MCKEACHIE, 2011). Segundo o fenômeno denominado *situated learning*, a aprendizagem está fortemente relacionada ao contexto e não pode ser separada do mesmo facilmente (SVINICKI; MCKEACHIE, 2011). Consequentemente, é preciso que a aprendizagem ocorra em situações bem próximas ao mundo real, para que os estudantes sejam capazes de transferir o que aprenderam para o mundo real. Do mesmo modo, as atividades e habilidades exigidas dos estudantes devem refletir as que serão necessárias futuramente. Como solução para essas questões, Svinicki e McKeachie (2011) propõem o emprego da aprendizagem experiencial (*experiential learning*).

A aprendizagem experiencial representa situações em que o projeto pedagógico busca o engajamento direto dos estudantes em experiências bem próximas aos problemas do mundo real. Os estudantes cooperam e aprendem uns dos outros em uma abordagem semiestruturada e o professor é um facilitador, e não um direcionador do progresso do estudante (NIU, 2012).

Alguns aspectos são relevantes para melhor entender o que representa a aprendizagem experiencial. O primeiro deles refere-se ao aprender através da ação, do fazer, da descoberta, da exploração, quer dizer, de alguma forma experimentar diretamente algo

(NIU, 2012). Em outras palavras, corresponde à necessidade de engajamento mencionado anteriormente. O segundo, diz respeito à importância dada à reflexão. De acordo com Svinicki e McKeachie (2011), a reflexão sobre a experiência é elemento chave para a aprendizagem experiencial, uma vez que, aprender a partir de experiências do mundo real não é uma tarefa fácil e requer bastante esforço mental. Para que realmente exista a transferência do que foi aprendido, é essencial que exista uma reflexão sobre o que foi aprendido. O aprendiz precisa estar ciente de que ele estava aprendendo e o que ele estava aprendendo, ou seja, ele deve ter consciência que algum objetivo de aprendizagem estava por trás daquilo que estava sendo feito (SVINICKI; MCKEACHIE, 2011).

A pouca mediação representa outro aspecto significativo para a aprendizagem experiencial. Nessa abordagem, apenas o propósito geral da aprendizagem é estabelecido, de modo que, existem poucas direções estabelecidas pelo professor (MOON, 2004).

Mais um aspecto é a distinção entre aprendizagem experiencial (*experiential learning*) e aprendizagem proveniente da experiência (*learning from experience*). Enquanto a primeira representa a intenção formal de aprender a partir de uma experiência particular, em determinado período de tempo (MOON, 2004), a segunda representa a aprendizagem informal que ocorre no contexto dos eventos do dia-a-dia na vida do indivíduo (MOON, 2004). Assim, a aprendizagem experiencial ocorre quando experiências são selecionadas dentro de um projeto pedagógico e são apoiadas por reflexão, análise crítica e síntese (AEE, 2011).

Enfim, a aprendizagem experiencial é tanto uma filosofia quanto uma metodologia (NIU, 2012). Enquanto filosofia, também chamada de Educação Experiencial, inspira metodologias que buscam o engajamento de professor e estudantes em experiências diretas, focadas na reflexão, com o objetivo de incrementar o conhecimento, desenvolver habilidades, esclarecer valores e melhor capacitar os estudantes para que os mesmos possam contribuir com suas comunidades (AEE, 2011). Em virtude da forte conexão existente entre a educação e a experiência pessoal, Dewey (1938) já defendia que a filosofia da educação fosse de algum modo comprometida com a filosofia experimental e empírica. De acordo com Dewey (1938), para a educação alcançar o seu objetivo, tanto para o indivíduo, quanto para a sociedade, a mesma precisa estar baseada em experiências próximas à vida real.

Como metodologia, autores sugerem um conjunto de passos que permite aos estudantes desenvolver novas habilidades e conhecimento, por meio de uma experiência de aprendizagem onde os mesmos colocam "a mão na massa", refletem e procuram transferir para outra situação o que foi aprendido (NIU, 2012) (KOLB, 1984 apud MOON, 2004).

Restringindo-se à concepção filosófica, a essência da aprendizagem experiencial pode ser sintetizada nos pontos retirados de Svinicki e McKeachie (2011) e AEE (2011):

- A aprendizagem aplica situações, problemas, ações, equipamentos, ferramentas do mundo real, o máximo possível;

- As situações envolvem problemas complexos e mal definidos, cujas respostas não são simples e, ainda, muitas respostas podem ser possíveis;

- A situação deve envolver o estudante na solução de um problema que reflete os problemas com os quais o mesmo defrontar-se-á no mundo real futuramente;
- O papel do professor é definir a experiência, apresentar problemas, estabelecer limites, dar suporte aos estudantes, garantir a segurança física e emocional, e facilitar o processo de aprendizagem;
- O estudante é engajado intelectualmente, emocionalmente, socialmente e fisicamente;
- O estudante participa ativamente, apresentando questões, investigando, experimentando, solucionando problemas;
- O estudante precisa refletir sobre suas experiências anteriores e construir novos significados;
- Exige-se do estudante ter iniciativa, tomar decisões e ser responsável pelos resultados;
- O estudante reflete sobre a sua solução, como alcançou essa solução e recebe respostas sobre a qualidade da sua solução;
- Relacionamentos são desenvolvidos e estimulados – entre estudantes e entre estudante e profissionais;
- Como o resultado não é totalmente previsível, professor e estudante podem experimentar riscos, incertezas, sucesso ou falha;
- Acontecem oportunidades que permitem ao professor e estudantes explorar e examinar seus próprios valores;
- Inclui a possibilidade de aprender por meio de erros ou acertos.

A *Northern Illinois University* apresenta uma lista de oportunidades de aprendizagem experiencial em nível de graduação em várias disciplinas (NIU, 2012), inclusive considerando exemplos de outras universidades, entre elas: *Apprenticeship Experiences*, *Clinical Experiences*, *Cooperative Education Experiences*, *Fellowship Experiences*, *Field Work Experiences*, *Internship Experiences*, *Practicum Experiences*, *Service Learning Experiences*, *Student Teaching Experiences*, *Study Abroad Experiences*, *Volunteer Experiences*.

Como o projeto de código aberto representa um projeto de software real, do qual outros profissionais participam de seu desenvolvimento, que atende a usuários de verdade, com ambientes de desenvolvimento e produção similares aos encontrados na indústria de software, fica clara a possibilidade de sua utilização por parte do professor, como recurso para a aprendizagem experiencial.

Ao longo desse capítulo, a partir da compreensão da definição e do processo de aprendizagem (simplificado) pudemos visualizar como a adoção de OSP supre elementos essenciais à aprendizagem e sob quais teorias essa adoção está fundamentada. Agora, para

iniciarmos nossa discussão sobre a sua eficácia, é necessário discernir o que pode ser aprendido por meio dessa abordagem. Desse modo, no próximo capítulo, identificamos quais objetivos de aprendizagem em engenharia de software podem ser trabalhados por meio da adoção de OSP.

Capítulo 3

OBJETIVOS DE APRENDIZAGEM VS. OSP

Uma forma de avaliar a eficácia de uma estratégia de ensino-aprendizagem é identificar se a estratégia empregada permite alcançar os objetivos de aprendizagem inicialmente traçados.

Com o propósito de direcionar a investigação a partir de uma base de objetivos de aprendizagem que fosse independente da disciplina ou das disciplinas onde a abordagem de uso de projetos de código aberto estivesse sendo aplicada, decidimos primeiramente perscrutar quais objetivos poderiam ser trabalhados por meio da abordagem. Com este intuito, consideramos os objetivos de aprendizagem para a área de engenharia de software propostos pela visão acadêmica, as lacunas de formação relatadas pela indústria de software, as propostas de utilização existentes na literatura relacionada e uma reflexão pessoal, em virtude da experiência possuída, depois de vários anos ensinando disciplinas da área de engenharia de software.

Por conseguinte, neste capítulo, após uma pequena introdução sobre objetivos de aprendizagem, discutimos e analisamos a abrangência da abordagem sobre os objetivos de aprendizagem em engenharia de software sob a perspectiva acadêmica e da indústria, as indicações de uso de projetos de código aberto pela literatura relacionada e concluímos com um resumo da possível cobertura das áreas de conhecimento da engenharia de software, resultado da união dos três conjuntos e da reflexão realizada. Ressaltamos no entanto, que a intenção do conjunto de objetivos resultante era apenas de direcionar a busca pelas evidências e de facilitar a comparação entre os estudos, como apresentaremos nos capítulos seguintes.

3.1 OBJETIVOS DE APRENDIZAGEM

Objetivos de aprendizagem estabelecem o que o estudante deverá saber ou ser capaz de fazer, após sua participação em determinada atividade de ensino-aprendizagem (I-TECH, 2010). Ou seja, eles devem descrever o conhecimento ou as habilidades que os estudantes devem apresentar como resultado da atividade. A definição dos objetivos de aprendizagem permite que todos os envolvidos compartilhem uma compreensão homogênea do

que será alcançado por meio do processo de ensino-aprendizagem (I-TECH, 2010). Os estudantes têm uma visão clara do que estarão tratando, o que estarão fazendo e o que, ao final, espera-se deles (UNM, 2005), de modo que, podem direcionar seus esforços apropriadamente, monitorar seu próprio progresso (CMU, 2015) e assumir a responsabilidade sobre o mesmo (I-TECH, 2010). Por outro lado, a definição clara dos objetivos de aprendizagem possibilita ao professor selecionar e organizar o conteúdo, estabelecer as estratégias de ensino e avaliação apropriadas (CMU, 2015), finalmente, verificar a eficácia em si, do processo de ensino-aprendizagem empregado.

Gasparin (2010) salienta que os objetivos possuem duas dimensões: a aquisição teórica dos conhecimentos científicos e a necessidade de colocar em prática este conteúdo. Sendo assim, "os objetivos, em sua formulação implícita ou explícita, devem constituir um todo entre a aprendizagem dos conteúdos e a aprendizagem de seu uso social" (GASPARIN, 2010). Stroulia et al. (2011) destacam uma mudança de paradigma, onde os educadores devem estabelecer competências profissionais ao invés de apenas definir objetivos que cubram o corpo de conteúdo de determinada área de conhecimento.

A Taxonomia de Objetivos Educacionais, publicada por Bloom em 1956, auxilia no processo de elaboração de objetivos ao definir um esquema de classificação de objetivos em níveis de aprendizagem. A classificação permite ao educador estabelecer o nível de aprendizagem pretendido para os estudantes como resultado de determinada atividade de ensino-aprendizagem (KRATHWOHL, 2002). A taxonomia, além de possibilitar a padronização da linguagem no meio acadêmico (FERRAZ; BELHOT, 2010), reforça a mudança para a instrução baseada em competências (BONACI; MUSTATA; IENCIU, 2013).

De acordo com Krathwohl (2002), posteriormente, a proposta inicial de Bloom foi revisada, de maneira que a estrutura original foi dividida em duas dimensões: conhecimento e processo cognitivo. A dimensão conhecimento diz respeito ao tipo do conteúdo que se tem como alvo, definindo quatro categorias (ver Tabela 3.1). Já a dimensão processo cognitivo retrata as competências intelectuais a serem desenvolvidas, correspondendo aos níveis da taxonomia original, com algumas pequenas mudanças na nomenclatura e na troca da ordem das duas últimas categorias. As categorias continuaram organizadas em ordem crescente de complexidade, representando desde a simples recordação das informações, aplicação do conhecimento, à análise e síntese das informações. Todavia, após a revisão, estão organizadas como uma hierarquia cumulativa relaxada, de modo que, atingir uma categoria mais simples é geralmente pré-requisito para atingir a próxima categoria mais complexa, embora possam ocorrer algumas exceções (KRATHWOHL, 2002). A Tabela 3.2 sumariza os seis níveis da dimensão Processo Cognitivo.

3.2 OBJETIVOS DE APREND. EM ES SOB A PERSPECTIVA ACADÊMICA

O Guia do Corpo de Conhecimento de Engenharia de Software (SWEBOK – *Software Engineering Body of Knowledge Guide*) tem como objetivos (BOURQUE; FAIRLEY, 2014): (i) promover uma visão consistente e abrangente da engenharia de software; (ii) especificar o escopo e posicionar claramente a engenharia de software com respeito às outras áreas de conhecimento; (iii) caracterizar o conteúdo da área da engenharia de

Tabela 3.1 Dimensão Conhecimento da Taxinomia de Bloom Revisada (KRATHWOHL, 2002, p. 214).

Categoria	**Descrição**
Conhecimento Fatual	Conhecimentos básicos do assunto. Ex.: terminologia, elementos e detalhes específicos, características.
Conhecimento Conceitual	Relacionamento entre os elementos básicos dentro da estrutura que permite que eles funcionem juntos. Ex.: classificações e categorias; princípios e generalizações; teorias, modelos e estruturas.
Conhecimento Procedimental	Como fazer algo. Ex.: Habilidades específicas; técnicas e métodos; critérios para decisão de qual procedimento utilizar.
Conhecimento Metacognitivo	Consciência do conhecimento adquirido sobre determinado conteúdo, consciência sobre a própria capacidade cognitiva.

Tabela 3.2 Dimensão Processo Cognitivo da Taxinomia de Bloom Revisada (KRATHWOHL, 2002, p. 215).

Categoria	**Descrição**	**Subcategorias**
Lembrar	Recuperando da memória conhecimento importante.	Reconhecendo
		Recordando
Entender	Determinando o significado da instrução.	Interpretando
		Exemplificando
		Classificando
		Sumarizando
		Inferindo
		Comparando
		Explicando
Aplicar	Realizando uma ação ou usando um procedimento em determinada situação.	Executando
		Implementando
Analisar	Quebrando o material em suas partes constituintes e determinando como essas partes se relacionam entre si e com toda a estrutura ou propósito geral.	Diferenciando
		Organizando
		Atribuindo
Avaliar	Fazendo julgamentos baseados em critérios e padrões.	Checando
		Criticando
Criar	Colocando os elementos juntos para formar um todo coerente ou fazendo um produto original.	Gerando
		Planejando
		Produzindo

software; (iv) prover os fundamentos para o desenvolvimento de currículos, dentre outros. Com esses objetivos, o guia descreve e organiza todo o conteúdo relacionado à área, representando uma compreensão consensual e bem aceita pela comunidade, características descritas por Lethbridge et al. (2007) como fortemente necessárias, para direcionar todas as formas de esforços educacionais.

O SWEBOK organiza todo o assunto que a engenharia de software abrange em áreas de conhecimento (KA – *Knowledge Area*). Ao todo são 15 KAs, das quais 11 são essencialmente relacionadas à engenharia de software; três correspondem a fundamentos de computação, matemática e engenharia, sobre os quais a engenharia de software está ancorada; e por fim, a KA 'Prática Profissional de Engenharia de Software' que diz respeito ao conhecimento, habilidades, atitudes que os engenheiros de software devem possuir, para que realizem suas atividades de modo profissional, responsável e ético. Cada KA é decomposta hierarquicamente em subáreas, tópicos e sub-tópicos, breve e suficientemente descritos.

Especificamente, ao longo deste trabalho, consideramos as 11 áreas notadamente relacionadas à engenharia de software e mais a área 'Prática Profissional de Engenharia de Software', uma vez que tais KAs representam os conteúdos e habilidades que devem ser desenvolvidos em disciplinas de engenharia de software, foco da pesquisa.

Convém ressaltar que, cumprindo seus objetivos, o SWEBOK delimita o conteúdo e as práticas recomendadas para a área de engenharia de software. Contudo, não determina quais conteúdos ou práticas devem ser desenvolvidos em 'nível de graduação'.

Por outro lado, a *Association for Computing Machinery* (ACM) e a *Institute of Electrical and Electronics Engineers Computer Society* (IEEE-CS), trabalhando em conjunto, elaboraram e ao longo dos anos vêm atualizando currículos de referência para cinco cursos de graduação da área de computação[1]. O objetivo desses currículos é prover orientação para as instituições acadêmicas e de acreditação (reconhecimento de curso) sobre o que deve constituir os cursos, cada um em sua área específica. Ou seja, tais documentos esboçam os conteúdos e competências que devem ser desenvolvidas durante a graduação. De forma semelhante, no Brasil, o Ministério da Educação e a Sociedade Brasileira de Computação, separadamente, propõem diretrizes para a formulação de currículos na área da computação, contudo, nestes casos, apenas esboçam os conteúdos que devem ser cobertos, sem detalhar as competências que precisam ser desenvolvidas dentro do contexto desses conteúdos (MEC, 2012) (SBC, 2005).

Focando na identificação dos objetivos de aprendizagem em engenharia de software que podem ser trabalhados por meio do uso de OSP, a ideia inicial foi utilizar como base o currículo de referência para elaboração de cursos de graduação em engenharia de software (denominado de SE2014) (ACM AND IEEE, 2015) para especificar os objetivos de aprendizagem que adotaríamos como parâmetro para a análise. Todavia, nem todos os profissionais atuantes na indústria de software são provenientes de um bacharelado em engenharia de software. Muitos têm a formação em ciência da computação, engenharia da computação, sistemas de informação ou tecnologia da informação (MEYER, 2001) (RADERMACHER; WALIA, 2013) (LETHBRIDGE et al., 2007). Para essas situações,

[1]http://www.acm.org/education/curricula-recommendations

o conjunto de objetivos de aprendizagem especificados a partir do SE2014 seria muito extenso e por isso inapropriado para ser utilizado como parâmetro de análise. Sondando os demais currículos, o currículo de ciência de computação (denominado de CS2013) (ACM AND IEEE, 2013) apareceu como alternativa por apresentar uma cobertura razoável da área de engenharia de software e principalmente por corresponder a opção mais recente dentre as demais, uma vez que a questão de atualização em computação pode significar mudanças importantes.

O corpo de conhecimento do CS2013, de modo semelhante ao SWEBOK, também é estruturado em áreas de conhecimento (KA). Porém no caso do CS2013, essas áreas são decompostas em unidades de conhecimento (KU – *knowledge unit*) que, por sua vez, são detalhadas em tópicos e, o mais importante para o foco de nossa pesquisa, em resultados de aprendizagem, classificados de acordo com o nível de aprendizagem esperado para os estudantes. Dessa maneira, os resultados de aprendizagem estabelecem exatamente o que os estudantes deverão ser capazes de fazer considerando os conteúdos descritos pelos tópicos. Com a intenção de simplificar a classificação dos resultados de aprendizagem, o CS2013 definiu apenas três níveis de aprendizagem (ver Tabela 3.3). Cabe ressaltar que a definição de resultados de aprendizagem e esta simplificação dos níveis de aprendizagem, com relação à Taxinomia de Bloom, adequam-se justamente ao nosso propósito de formulação do conjunto inicial de objetivos de aprendizagem em engenharia de software. Os resultados estabelecem competências, ao invés de somente definir a abrangência de conteúdos da área de conhecimento (exatamente como devem ser os objetivos de aprendizagem, conforme mencionado na Seção 3.1), e os níveis de aprendizagem facilitam o julgamento do alcance dos objetivos estabelecidos.

Tabela 3.3 Níveis de Aprendizagem no CS2013.

Nível	**Competência desenvolvida**
Familiaridade	O aluno compreende o conceito e o que ele significa. O que o aluno sabe a respeito disso.
Uso	O aluno é capaz de usar ou aplicar o conceito de modo concreto. O aluno sabe como fazer.
Avaliação	O aluno é capaz de considerar o conceito segundo vários pontos de vista e/ou justificar a seleção de determinada abordagem para resolver o problema. Envolve a habilidade de selecionar a abordagem apropriada, dentre as alternativas. O aluno é capaz de explicar porque fez de determinado modo.

O CS2013 define a engenharia de software (SE) como uma de suas 18 KAs e desdobra-a em 10 unidades de conhecimento mais específicas. O documento ainda explicita a sua relação com mais cinco unidades de conhecimento de outras quatro KAs (*Software Development Fundamentals* – SDF, *Information Assurance and Security* – IAS, *Social Issues and Professional Practice* – SIPP e *Human-Computer Interaction* - HCI). Contudo, analisando detalhadamente o documento, sob a perspectiva do SWEBOK, encontramos outras unidades importantes em SIPP e HCI. Consideramos então, os resultados de aprendizagem propostos nessas unidades, como escopo inicial dos objetivos de aprendizagem em

engenharia de software.

Refletindo sobre a possibilidade de utilização de projetos de código aberto para o alcance de cada um desses objetivos e com uma visão aberta a quaisquer oportunidades, sugerimos um grau de impacto para essa utilização. A intenção do grau de impacto era tentar medir quanto a adoção da abordagem poderia favorecer o alcance do objetivo de aprendizagem. Desse modo, a escala utilizada foi a seguinte:

(3) A adoção de OSP pode favorecer satisfatoriamente o alcance do objetivo de aprendizagem;

(2) A adoção de OSP pode favorecer razoavelmente o alcance do objetivo de aprendizagem;

(1) A adoção de OSP pode favorecer um pouco o alcance do objetivo de aprendizagem;

(0) Não há relação entre o alcance do objetivo de aprendizagem e a adoção de OSP.

Para de algum modo justificar o grau de impacto estabelecido, sugerimos atividades a serem realizadas com a adoção de OSP, visando atender o objetivo de aprendizagem em questão. Nesse caso, a intenção não era explorar todas as possibilidades de utilização, mas sim justificar a adoção, apresentando pelo menos uma ou algumas possibilidades.

Um exemplo, dessa visão aberta às possibilidades e ações que podem ser realizadas, seria que apesar de acreditarmos que nenhum projeto de código aberto seja especificado por meio de linguagens formais, supomos que o professor pode apropriar-se de um módulo de algum OSP e criar a especificação formal para o mesmo ou solicitar que os estudantes o façam, de modo que, os respectivos objetivos de aprendizagem possam ser alcançados. Como nessa situação, um exemplo real (gerado a partir do OSP) passa a ser ser manipulado, consideramos o grau de impacto igual a três.

É importante dizer que, ao longo desse estudo, os objetivos de aprendizagem que tiveram ao menos o grau de impacto '1' foram considerados como objetivos de aprendizagem factíveis a serem trabalhados por meio da abordagem.

A Tabela 3.4 lista todas as unidades de conhecimento do CS2013 consideradas, o total de resultados de aprendizagem definidos para cada uma dessas unidades, bem como, o número de objetivos de aprendizagem factíveis de serem trabalhados com a abordagem OSP, de acordo com o grau de impacto atribuído, após a análise um a um dos resultados de aprendizagem dessas unidades. Os objetivos de aprendizagem definidos em razão dos resultados de aprendizagem estabelecidos pelo CS2013 encontram-se assinalados na coluna 'CS2013' da Tabela A.1 no Apêndice A, com CSLO (*Computer Science Learning Outcome*), juntamente com a sigla da respectiva KU. Também na Tabela A.1, apresentamos as justificativas para atribuição do grau de impacto para cada objetivo de aprendizagem, cujo grau foi igual ou superior a '1'.

A Tabela 3.4 permite ter a visão da possibilidade de cobertura da utilização da abordagem para suprir uma necessidade de aprendizagem razoável sobre engenharia de software sob a perspectiva acadêmica.

Tabela 3.4 Resultados de Aprendizagem (CS2013) x Objetivos Aprendizagem.

KA	**Unidades de Conhecimento**	**Resultados de Aprendizagem CS2013**	**Objetivos de Aprendizagem Factíveis**			**Objetivos de Aprendizagem não Factíveis**
			3	2	1	
SE	Engenharia de Requisitos (RE)	13	7		3	3
SE	Design (SD)	23	20	2	1	
SE	Construção (SC)	9	8	1		
SE	Verificação e Validação (V&V)	17	16	1		
SE	Evolução de Software (SE)	6	4		2	
SE	Processos de Software (SP)	14	4	2	5	3
SE	Gerenciamento de Projetos de Software (SPM)	25	2	6	17	
SE	Ambientes e Ferramentas (T&E)	6	8		1	
SE	Confiabilidade de Software (SR)	7	5	1	1	
SE	Métodos Formais (FM)	5	5			
SDF	Métodos de Desenvolvimento (SDF)	12	12			
IAS	Segurança na Engenharia de Software (SSE)	5	4		1	
SIPP	Comunicação Profissional (PC)	10	7		2	1
SIPP	Sustentabilidade (S)	8			1	7
HCI	Design e testes centrados no usuário (UCDT)	5	2			3
SIPP	Ética Profissional (PE)	15	1	2	2	10
SIPP	Propriedade Intelectual (IP)	12	10			2
SIPP	Economia na Computação (EC)	6			2	4
HCI	Fundamentos de Interação Humano-Computador (HCIF)	5	2	1		2
HCI	Projeto de Interação (DI)	4	3			1
HCI	Programação de Sistemas Interativos (PIS)	4	4			

Durante a análise, acrescentamos objetivos de aprendizagem para alguns tópicos das unidades de conhecimento presentes no CS2013, uma vez que, não encontramos resultados de aprendizagem associados aos mesmos, por exemplo, para o tópico *"Characteristics of maintainable software"*, definimos o objetivo de aprendizagem "Descrever as características apresentadas por um software fácil de manter" (LO119), enquanto que para o tópico *"Black-box and white-box testing techniques"*, definimos o objetivo de aprendizagem "Descrever e distinguir entre as diferentes técnicas de testes caixa-preta e caixa-branca" (LO98). Ao todo, estabelecemos 14 novos objetivos de aprendizagem. Estes objetivos encontram-se assinalados na coluna 'CS2013' da Tabela A.1 no Apêndice A, com CSCT (*Computer Science Curriculum Topic*), juntamente com a sigla da respectiva KU.

Realizamos o mesmo procedimento para algumas competências que os estudantes formados deverão demonstrar, também especificadas pelo CS2013, mas que não foram cobertas por nenhum resultado de aprendizagem. Exemplificando alguns objetivos definidos por essa razão: "Ser capaz de solucionar problemas" (LO266); "Ter vivenciado

pelo menos um projeto real" (LO269) e "Ser capaz de pensar o problema em vários níveis de detalhe e abstração" (LO273). Assim, adicionamos 11 objetivos de aprendizagem. Neste caso, os objetivos encontram-se assinalados na coluna 'CS2013' da Tabela A.1 no Apêndice A, com CSCG (*Computer Science Characteristics of Graduates*).

Visualizando os resultados apresentados na Tabela 3.4, para os 211 resultados de aprendizagem estabelecidos pelo CS2013, consideramos que apenas 36 objetivos de aprendizagem não seriam factíveis de serem alcançados a partir da abordagem OSP, isto é, a adoção da abordagem não influenciaria a obtenção de 17% dos resultados de aprendizagem. Considerando todas as unidades de conhecimento pertencentes à KA 'Engenharia de Software', apenas 4,8% dos resultados não seriam favorecidos, enquanto que, com exceção das unidades SPM e SP, a maioria dos resultados de aprendizagem, das demais unidades dessa KA, seria fortemente favorecida (grau de impacto igual a três). As demais áreas de conhecimento mais técnicas (SDF, IAS, HCI) também seriam na maioria das vezes fortemente favorecidas. Já em relação à área de 'Questões Sociais e Prática Profissional' (SIPP), algumas unidades seriam bem favorecidas (PC e IP), ao passo que as outras áreas não seriam tão favorecidas (S, PE e EC).

Comparando o número de resultados de aprendizagem com o somatório dos objetivos factíveis e não factíveis, existe uma diferença para a unidade de conhecimento T&E. Como não há uma correlação direta entre as unidades de conhecimento de engenharia de software do CS2013 e as áreas de conhecimento da versão atual do SWEBOK, para facilitar a futura classificação segundo o SWEBOK, decidimos desdobrar o resultado de aprendizagem "Demonstrar a capacidade de usar ferramentas de suporte ao desenvolvimento de um software de médio porte" da unidade citada, em quatro objetivos mais específicos relacionados às áreas de requisitos, design, testes e gerenciamento de configuração.

3.3 OBJETIVOS DE APREND. EM ES SOB A PERSPECTIVA DA INDÚSTRIA

Como mencionado na introdução deste documento, existem diversas lacunas ou deficiências na formação dos profissionais de computação que vão trabalhar na indústria de software. Obviamente que, se objetivos de aprendizagem fossem direcionados a suprir essas deficiências, a indústria seria beneficiada. Desse modo, buscamos correlacionar as lacunas relacionadas à engenharia de software com os objetivos de aprendizagem resultantes da análise do CS2013. Para os casos onde não havia correspondência acrescentamos novos objetivos de aprendizagem, por exemplo: "Comentar e documentar o código apropriadamente"(LO62); "Saber usar e configurar ferramentas necessárias ao ambiente de produção do software na indústria" (LO84) e "Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias" (LO241). É oportuno salientar que, como ponto de partida para a análise, utilizamos as deficiências reportadas pelos estudos de Radermacher e Walia (2013) e Radermacher, Walia e Knudson (2014), já que representam trabalhos recentes e visavam consolidar resultados anteriores. Como resultado dessa atividade, 23 deficiências foram associadas aos objetivos de aprendizagem existentes e 27 objetivos de aprendizagem foram acrescentados. Assim, os objetivos relacionados às deficiências apontadas pela indústria de software encontram-se assinalados na coluna 'Indústria' da Tabela A.1 no Apêndice A.

A Tabela 3.5 organiza as lacunas de formação de acordo com áreas de conhecimento estabelecidas pelo SWEBOK, identifica os objetivos de aprendizagem relacionados (LO – *learning objectives*) e o grau de impacto atribuído para o atendimento do objetivo. A partir dessa tabela podemos visualizar que, após a reflexão realizada (de maneira similar a explicada na seção 3.2), apenas duas deficiências não poderiam ser supridas pela abordagem de adoção de OSP. Julgamos as deficiências "Preparação e condução de entrevistas para a elicitação de requisitos" e "Habilidade de comunicação oral (uso de jargões e habilidades de escuta)" como objetivos de aprendizagem não factíveis, em virtude de estarem ligadas de algum modo à existência de comunicação oral entre desenvolvedor e usuário, que usualmente não ocorre em projetos de código aberto. Quanto à deficiência "Paixão por tecnologia ou pelo domínio com o qual está trabalhando" não conseguimos enxergá-la como um objetivo de aprendizagem em si, isto é, algo que pudesse ser trabalhado ao longo de uma disciplina. Mesmo assim, acreditamos que os estudantes possam ser de algum modo influenciados pela paixão que move os desenvolvedores que participam de projetos de código aberto. Por outro lado, consideramos que a grande maioria das demais deficiências pode ser superada pela adoção da abordagem (grau de impacto igual a três). Enfim, como a deficiência "gerenciamento de projetos" não foi suficientemente detalhada em um dos estudos utilizado como referência, decidimos associá-la aos vários objetivos de aprendizagem de gerenciamento de projetos definidos durante a análise do CS2013, que, por sua vez, têm grau de impacto de um a três (ver Tabela 3.4).

Ao longo da análise também verificamos a grande quantidade de deficiências assinaladas para a área "prática profissional" e que não foram apontadas deficiências para as áreas "modelos e métodos", "qualidade de software" e "economia da engenharia de software", outras KAs do SWEBOK. A ocorrência da primeira situação justifica ainda mais a necessidade de abordagens onde os estudantes tenham a vivência de projetos mais próximos do mundo real, ainda na academia. Na segunda situação, uma explicação aceitável, é que os estudos utilizados como referência (Radermacher e Walia (2013) e Radermacher, Walia e Knudson (2014)) documentaram apenas as deficiências com mais ocorrências. Portanto, outras deficiências existem, possivelmente também para essas áreas, contudo, não foram reportadas pelas referências utilizadas. No que diz respeito a este estudo, não julgamos que a ausência das deficiências menos frequentes prejudicou a análise da abordagem OSP, já que o propósito era complementar a visão acadêmica e isso foi feito com as deficiências mais importantes.

Tabela 3.5: Deficiências de Formação x Objetivos de Aprendizagem.

Deficiência	**LO**	**Grau**
Requisitos de Software		
Preparação e condução de entrevistas para a elicitação de requisitos	LO89	0
Elicitação de Requisitos	LO3	1
Especificação de Requisitos	LO6	3
Design de Software		
Design de Software	LO14	3
Desenvolvimento para plataforma móvel	LO75	3
Técnicas de modelagem	LO37	3
Nível de detalhamento dos modelos	LO38	3
Aplicação do paradigma OO	LO44	3
Construção de Software		
Comentar o código adequadamente	LO62	3

Deficiência	**LO**	**Grau**
Seguir padrões de codificação	LO49	3
Habilidades de programação: procedural, multithread, independência de linguagem, linguagens de quarta geração	LO63	3
Testes e localização do erro (*debugging*) no código	LO66	3
Configuração e uso de ferramentas similares ao ambiente de produção	LO84	1
Uso de ferramentas de análise de código	LO85	3
Uso de ambientes integrados de desenvolvimento	LO82	3
Testes de Software		
Construção de testes de unidades não redundantes	LO99	3
Processo de testes e correção de erros sem que haja a reintrodução de erros antigos	LO100	3
Uso de ferramentas de testes	LO107	3
Uso de ferramentas de localização de erros (*debugging*)	LO108	3
Uso de ferramentas de cobertura de testes	LO109	3
Exposição a ferramentas de integração contínua e testes de regressão	LO110	3
Manutenção de Software		
Manutenção de software	LO118	3
Gerência de Configuração de Software		
Uso de ferramentas de gerenciamento de configuração de software	LO133	3
Realização de merge entre códigos e/ou ramificação do código (*branching*)	LO134	3
Gerenciamento de Engenharia de Software		
Gerenciamento de projetos	LO135 – LO154	1-3
Planejamento de atividades	LO329	3
Estimativas de custos de projetos	LO138	2
Uso de ferramentas de gerenciamento de projetos	LO155	3
Uso de ferramenta de rastreamento de necessidades (*issue tracking*)	LO328	3
Processos de Engenharia de Software		
Conhecimento sobre os modelos (*frameworks*) de melhorias de processo de software	LO292	1
Prática Profissional em Engenharia de Software		
Comportamento ético	LO199	2
Comprometimento com a aprendizagem contínua	LO326	1
Compreensão das expectativas que recaem sobre o profissional da área	LO203	1
Motivação e disposição	LO238	1
Reconhecimento e aprendizagem com os próprios erros, aceitar respostas de terceiros	LO233	3
Gerenciamento do tempo	LO234	3
Habilidades de liderança	LO240	1
Experiência com trabalho em equipe	LO227	3
Participação em equipes multidisciplinares	LO228	3
Pensamento crítico, geração e síntese de novas ideias	LO241	3
Solução de problemas	LO266	3
Propor soluções alternativas	LO267	3
Pró-atividade	LO236	3
Paixão por tecnologia ou pelo domínio com o qual está trabalhando	LO239	1
Experiência com projetos reais	LO269	3
Habilidade de visualizar o projeto como um todo (visão do todo)	LO272	3
Habilidade de comunicação escrita para produção de documentação técnica	LO246	3
Habilidade de comunicação oral (uso de jargões e habilidades de escuta)	LO327	0
Habilidade de apresentação	LO311	1
Hesitação em solicitar ajuda	LO249	3
Articulação de argumentos e fornecimento de explicações claras	LO244	3

3.4 INDICAÇÕES DE USO DE OSP NA EDUCAÇÃO EM ENGENHARIA DE SOFTWARE

Observamos na literatura a existência de diversos estudos sobre o uso de OSP na educação formal em engenharia de software (NASCIMENTO; BITTENCOURT; CHAVEZ, 2015). Estes estudos indicam a utilização de projetos de código aberto para a aprendizagem

de vários tópicos dessa área de conhecimento. Essas indicações aparecem em forma de atividades que podem ser executadas, benefícios obtidos, ou mesmo, com a indicação direta de conteúdos ou tópicos a serem trabalhados por meio da abordagem.

Por outro ângulo, sob uma perspectiva de aprendizagem informal, Ghosh e Glott (2005) relataram a melhoria de um conjunto de habilidades quando desenvolvedores participam de projetos de código aberto.

Reunindo essas duas óticas, verificamos quais objetivos de aprendizagem, presentes até o momento em nosso conjunto, poderiam ser atendidos pela abordagem OSP por indicação da literatura existente. Novamente, para as situações onde não havia correspondência, alguns objetivos de aprendizagem foram acrescentados, a exemplo de: "Projetar código modularizado" (LO15); "Reusar código escrito por terceiros" (LO70) e "Opinar sobre diferentes estilos de programação baseado em experiência" (LO76). Assim, 78 indicações foram associadas aos objetivos existentes e 40 objetivos foram adicionados.

Os objetivos, cuja literatura relacionada indica o seu atendimento através da adoção de OSP, encontram-se assinalados na coluna 'Literatura' da Tabela A.1 no Apêndice A, com a respectiva referência onde a indicação foi encontrada.

A Tabela 3.6 resume a quantidade de estudos que indicam a utilização da abordagem para a aprendizagem de engenharia de software, classificados por área do SWEBOK, o número de objetivos de aprendizagem (LO - *Learning Objectives*) factíveis cobertos por essas indicações de acordo com o grau de impacto, e por fim, o número de objetivos de aprendizagem factíveis (por grau de impacto) e não factíveis, como resultado somente da reflexão realizada ao longo do estudo (semelhantemente à explicada na Seção 3.2).

Por meio da Tabela 3.6, observamos a possível abrangência da abordagem sob a perspectiva da literatura relacionada (LO Factíveis por Indicação) e sob a perspectiva desse estudo (LO Factíveis por Reflexão), lembrando que o grau de impacto é resultado apenas da reflexão efetuada. Por conseguinte, através dessa visão resumida, o leitor deste documento pode distinguir as indicações já existentes de emprego da abordagem proposta pela literatura, das que foram acrescentadas nesse trabalho.

Como dito no início dessa seção, os estudos indicam a abordagem OSP para a aprendizagem de diversos tópicos de engenharia de software. Durante a análise, identificamos casos onde um estudo estava associado a vários objetivos de aprendizagem e, em várias situações, vários estudos estavam associados a um mesmo objetivo de aprendizagem. Por essa razão, o 'número de estudos' não corresponde ao total de 'LOs Factíveis por Indicação' para cada linha da tabela. Como se pode ver, encontramos indicações de utilização de projetos de código aberto para todas as áreas do SWEBOK. As áreas com mais indicações foram: prática profissional em engenharia de software, design de software, construção de software e manutenção de software. Por outro lado, além de reforçarmos essas áreas enxergando vários objetivos factíveis novos para as mesmas, apontamos outros objetivos factíveis para as demais áreas. Desse modo, também destacamos áreas como qualidade de software, gerenciamento de engenharia de software, testes de software e economia de engenharia de software.

A aprendizagem por meio de um projeto prático é essencial para as áreas de requisitos, design, construção, testes, manutenção, gerência de configuração e modelos e métodos de engenharia de software, por isso consideramos um alto grau de impacto da utilização da

abordagem OSP para a grande maioria dos objetivos relacionados a estas áreas. Convém ressaltar que, os objetivos considerados como não factíveis para a área de requisitos estão relacionados a algumas práticas de elicitação de requisitos e prototipação, que não são comuns em projetos de código aberto. De modo semelhante, julgamos os objetivos da área de design ligados a algumas práticas de projeto de interface de usuário.

A vivência de um projeto também é relevante para a área de gerenciamento de engenharia de software, contudo, em virtude do processo de desenvolvimento e de negócio de projetos de código serem bastante diferentes dos praticados na indústria, os graus de impacto atribuídos ficaram mais distribuídos. Quanto às atividades de aprendizagem que podem ser executadas, Stamelos (2009) defende que esta área de conhecimento só pode ser completamente trabalhada, quando o projeto OSP for gerenciado pelos próprios estudantes, ou seja, quando os estudantes fazem parte do núcleo da comunidade OSP. De outro ponto de vista, acreditamos que os objetivos de aprendizagem referentes à área também podem ser trabalhados a partir da definição de um subprojeto de melhoria local, independente da comunidade, como por exemplo, a adição de uma nova funcionalidade proposta pelo professor.

No tocante à qualidade de software, ponderamos que apesar das poucas indicações encontradas, a área pode ser bastante trabalhada. Esta visão otimista justifica-se pela disponibilidade de código, de onde bons e maus exemplos podem ser extraídos e métricas podem ser aplicadas. Dessa maneira, a dimensão de qualidade do produto pode ser manipulada sobre projetos reais existentes, além da vivência das práticas de garantia qualidade realizadas pela comunidade do projeto.

Por fim, no caso específico da área da prática profissional em engenharia de software, encontramos muitas indicações, acrescentamos várias outras indicações e ainda restaram muitos objetivos não factíveis. Pela própria denominação da área fica fácil compreender a influência de utilização de um projeto real, particularmente projetos de código aberto, para a aprendizagem, ou melhor especificando nesse caso, para o desenvolvimento de competências e atitudes. Consequentemente, justificar a grande quantidade de objetivos de aprendizagem factíveis estabelecidos para a mesma. Entretanto, o grande número de objetivos não factíveis explica-se pelo fato de existirem objetivos relacionados às questões de sustentabilidade da computação e ética profissional, provenientes dos resultados de aprendizagem estabelecidos pelo CS2013, onde obrigatoriamente não há necessidade de envolvimento do estudante em um projeto de software para o seu entendimento.

3.5 OBJETIVOS DE APRENDIZAGEM EM ENGENHARIA DE SOFTWARE X OSP

Como última visão e resumo da possibilidade de cobertura da área de engenharia de software propiciada pela abordagem, elaboramos a Tabela 3.7. Esta tabela apresenta os objetivos de aprendizagem factíveis organizados por área/subárea de conhecimento do SWEBOK, sendo que, em alguns casos, os objetivos envolveram mais de uma subárea. Os objetivos encontram-se ainda classificados de acordo com a origem das referências utilizadas em sua formulação, seja o CS2013, a indústria e/ou os estudos relacionados. Dessa maneira, a tabela apresenta as interseções entre os conjuntos de origem, quando o mesmo

Tabela 3.6 Escopo de Indicações de Adoção de OSP.

Áreas de Conhecimento	**Número de Estudos**	**LO Factíveis / Indicação**				**LO Factíveis / Reflexão**				**LO não Factíveis**
		3	2	1	Total	3	2	1	Total	
Requisitos de Software	7	2		1	3	5		2	7	6
Design de Software	18	15	2		17	23	2	1	26	2
Construção do Software	15	13			13	13	2	1	16	
Teste de Software	7	11			11	13			13	
Manutenção de software	14	14		3	17	3			3	
Gerência de Configuração do Software	8	5			5	5			5	
Gerenciamento de Engenharia de Software	7	1		5	6	4	3	11	18	1
Processos de Engenharia de Software	9	6	1		7	1	1	7	9	3
Modelos e Métodos de Engenharia de Software	3	1			1	7			7	1
Qualidade de Software	3	4			4	23		1	24	
Prática profissional em Engenharia de Software	25	27	2	4	33	14	2	6	22	21
Economia da Engenharia de Software	2	1			1		4	10	14	4

objetivo de aprendizagem aparece classificado em mais de uma coluna, e as disjunções, quando o objetivo encontra-se em apenas um coluna. Para atender aos tópicos relevantes do SWEBOK, não atendidos pelo conjunto de objetivos de aprendizagem definidos até o momento e onde, ao nosso ver, poderia ser aplicada a abordagem, definimos mais 14 objetivos de aprendizagem (ver coluna 'Proposta Pessoal'). Os objetivos acrescentados, encontram-se detalhados na Tabela A.1 no Apêndice A e correspondem aos que tem a coluna 'P' assinalada.

A Tabela 3.7 mostra que o escopo de atendimento do corpo de conhecimento de engenharia de software é bastante amplo, inclusive visualizando-o de forma mais detalhada, isto é, considerando as subáreas de conhecimento do SWEBOK. Apenas três subáreas não tiveram nenhum objetivo de aprendizagem factível associado, são elas: "fechamento do projeto", "ciclo de vida econômico", "métodos de análises econômicos". Este não atendimento é bastante compreensível, uma vez que, projetos de código aberto seguem o processo de desenvolvimento evolutivo, onde usualmente não existe uma perspectiva de 'fechamento do projeto' e, são construídos essencialmente por esforço voluntário, no qual a questão econômica não é tão importante. Convém ressaltar que tais conteúdos representam ainda conhecimentos avançados e não essenciais para a formação básica dos estudantes.

Tabela 3.7: Objetivos de Aprendizagem Factíveis por Subárea do SWEBOK.

SWEBOK Subárea	**CS2013**	**Indústria**	**Trabalhos Relacionados**	**Proposta Pessoal**
Requisitos de Software				

SWEBOK Área/Subárea	**CS2013**	**Indústria**	**Trabalhos Relacionados**	**Proposta Pessoal**
Fundamentos sobre requisitos	LO1			
Processo de requisitos	LO2, LO4			
Elicitação de requisitos	LO3		LO3	
Análise	LO3	LO3	LO3	
Especificação	LO5, LO6	LO6	LO6	
Considerações práticas	LO7			LO8
Validação de requisitos	LO9		LO9	
Ferramentas de manipulação de requisitos	LO10			
Design de Software				
Fundamentos do Design	LO12, LO14, LO13	LO14	LO52, LO12, LO14, LO15	
Questões chaves do design	LO17, LO19, LO20, LO283, LO284			LO16, LO18
Arquitetura e estrutura do software	LO21, LO22, LO23, LO24, LO25, LO26, LO27, LO28, LO29, LO30, LO31, LO32	LO75	LO21, LO22, LO25, LO30	
Projeto de interface com o usuário	LO53, LO56, LO57, LO33, LO320, LO279		LO33, LO35	
Notação (Modelagem) do design	LO36, LO37, LO38	LO37, LO38	LO37, LO38	
Análise e avaliação da qualidade do design	LO40, LO41, LO42		LO40, LO41	
Estratégias e Métodos de Design	LO43, LO44, LO46, LO282	LO44	LO43, LO44, LO46	
Ferramentas de design	LO47			
Construção de Software				
Fundamentos da construção	LO48, LO49, LO50	LO49, LO62	LO48, LO49, LO50	
Gerência da construção	LO60		LO61	
Considerações práticas	LO64, LO66, LO67, LO72, LO73, LO74, LO285, LO286	LO63, LO66	LO63, LO65, LO66, LO70, LO71	
Tecnologias de Construção	LO77, LO78, LO79, LO80, LO81		LO76	
Ferramentas de construção	LO82, LO83	LO82, LO84, LO85	LO82, LO83, LO86	
Testes de Software				
Fundamentos de testes	LO92, LO93, LO94		LO90, LO91	
Níveis de teste	LO34, LO95, LO96, LO97	LO99	LO95, LO96, LO99	
Técnicas de teste	LO98, LO101, LO102, LO103	LO99, LO100	LO98, LO99, LO101, LO102, LO103	
Processos de Teste	LO104, LO105		LO104, LO105	
Métricas relacionadas a testes				LO87, LO88
Ferramentas de Teste	LO106, LO107	LO107, LO108, LO109, LO110		
Manutenção de Software				
Fundamentos de Manutenção	LO112, LO113	LO118	LO112, LO113, LO114, LO115, LO116, LO117, LO118	LO111
Questões chave	LO119, LO120		LO120	
Processo de Manutenção	LO68, LO69, LO287		LO68, LO69	
Técnicas de Manutenção	LO122, LO123, LO318, LO319, LO125		LO122, LO123, LO318, LO319, LO124, LO125	
Ferramentas de suporte à manutenção			LO126	
Gerência de Configuração de Software				

SWEBOK Área/Subárea	**CS2013**	**Indústria**	**Trabalhos Relacionados**	**Proposta Pessoal**
Gerenciamento	LO128, LO129		LO127, LO128	
Identificação da configuração	LO130			
Controle de configuração			LO131	
Acompanhamento de situação de configuração				LO58
Auditoria de configuração				LO59
Gerenciamento e Distribuição de Releases			LO132	
Ferramentas de Gerência de Configuração	LO133	LO133, LO134	LO133	
Gerenciamento de Engenharia de Software				
Iniciação e Definição do escopo			LO137	LO135, LO136
Planejamento do projeto	LO138, LO139, LO140, LO141, LO142, LO143, LO144, LO145, LO288	LO138, LO329	LO142	
Execução do plano	LO146, LO147		LO147, LO149, LO150	LO148
Revisão e Avaliação	LO151		LO153	
Medição da engenharia de Software	LO152, LO154			
Fechamento do projeto				
Ferramentas de apoio ao gerenciamento de projetos	LO155	LO155, LO328		
Processos de Engenharia de Software				
Definição do Processo de Software	LO156		LO157	
Ciclo de Vida de Software	LO158, LO162, LO163, LO164, LO290, LO291		LO158, LO159, LO160, LO161, LO162	
Avaliação e Melhoria de Processo	LO165, LO292	LO292		
Medição de Software	LO166, LO167			
Ferramentas de suporte ao Processo de Engenharia de Software			LO168	
Modelos e Métodos de Engenharia de Software				
Modelagem			LO169	
Tipos de modelos			LO38	
Análise dos modelos	LO171			LO172
Métodos de Engenharia de Software	LO321, LO322, LO323, LO324, LO325		LO160	
Qualidade de Software				
Fundamentos de Qualidade	LO173, LO174, LO175, LO176		LO178, LO179	LO177
Processos de Gerenciamento da qualidade	LO174, LO180, LO181, LO182, LO183, LO184, LO185, LO186, LO187, LO188, LO54, LO298, LO55		LO174	
Considerações práticas	LO189, LO190, LO191, LO193, LO194, LO297		LO196	
Ferramentas de Suporte à qualidade	LO197, LO198			
Prática Profissional em Engenharia de Software				
Profissionalismo	LO200, LO201, LO202, LO326, LO203, LO205, LO207, LO208, LO209, LO210, LO211, LO213, LO215, LO216, LO217, LO218, LO219, LO220, LO226	LO199, LO326, LO203	LO199, LO326, LO203, LO208, LO209, LO210, LO215, LO216, LO217, LO218, LO219	

SWEBOK Área/Subárea	**CS2013**	**Indústria**	**Trabalhos Relacionados**	**Proposta Pessoal**
Dinâmica de Grupo e Psicologia	LO146, LO147, LO227, LO229, LO230, LO266, LO267, LO269, LO270, LO271, LO272, LO273, LO309, LO310	LO227, LO228, LO233, LO234, LO236, LO238, LO240, LO241, LO266, LO267, LO269, LO272	LO147, LO227, LO228, LO229, LO230, LO231, LO232, LO233, LO234, LO235, LO237, LO238, LO240, LO266, LO268, LO269, LO272, LO273	LO242
Habilidades de Comunicação	LO243, LO246, LO247, LO248, LO250, LO251, LO311	LO244, LO246, LO249	LO243, LO244, LO245, LO246, LO250	
Economia de Engenharia de Software				
Fundamentos de economia da engenharia de software			LO252	
Ciclo de vida econômico				
Riscos e Incertezas	LO254, LO255, LO256, LO257, LO258, LO259, LO260, LO261, LO262, LO263, LO264, LO265			
Métodos de Análises Econômicos				
Considerações práticas	LO312, LO316			

Ao final dessa análise de cobertura, a união das perspectivas descritas resultou em 320 objetivos de aprendizagem estabelecidos, dos quais, após reflexão abrangente sobre as possibilidades de utilização da abordagem, 282 foram considerados factíveis e apenas 38 foram considerados como não factíveis. Dos 282 objetivos de aprendizagem factíveis, 118 foram provenientes de indicações da literatura relacionada à abordagem, o que representa 42% dos objetivos factíveis, e 164 foram acrescentados por essa investigação, ou seja, 58%. A Figura 3.1 ilustra o resultado desse estudo. O Apêndice A elenca todos os objetivos de aprendizagem, especificando para cada um, as referências utilizadas para a sua formulação, o nível de aprendizagem pretendido, o grau de impacto da adoção da abordagem para o alcance do objetivo de aprendizagem e algumas atividades possíveis de serem realizadas com o emprego de OSP, sugeridas no intuito de justificar o grau de impacto atribuído.

Com vistas à investigação da eficácia da abordagem com OSP, foco da pesquisa, o conjunto de objetivos de aprendizagem factíveis, resultado da análise realizada, permitiu o direcionamento das atividades seguintes. Buscamos comprovar, por meio de evidências geradas por nossas próprias experiências e reportadas por trabalhos relacionados, se realmente a abordagem favorecia o alcance de um subconjunto destes objetivos. Todavia antes de apresentarmos os resultados obtidos, descrevemos os procedimentos de investigação empregados ao longo de toda a pesquisa, no próximo capítulo.

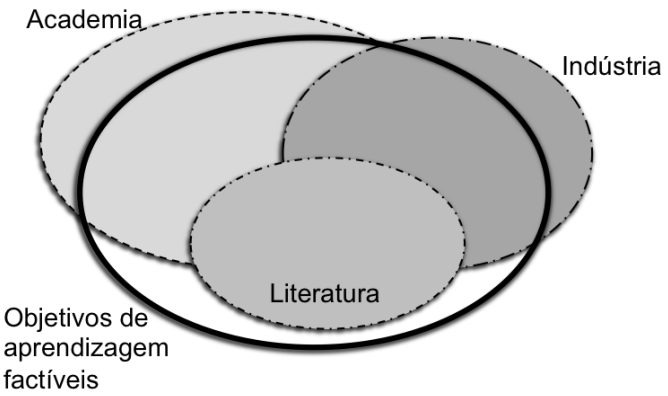

Figura 3.1 Visão dos objetivos de aprendizagem factíveis.

Capítulo

4

PROCEDIMENTOS DE INVESTIGAÇÃO

Como pesquisadores da área de computação e, consequentemente, representantes das ciências exatas, temos a tendência pela concepção positivista na elaboração de nossos projetos de pesquisa. Sob essa visão, buscamos objetividade, mensurabilidade, previsibilidade, controlabilidade, a construção de leis e regras de comportamento, isto é, o alvo é quantificar nossos resultados. Contudo, a sala de aula é bastante diferente de um laboratório, no qual criamos um ambiente estável, com variáveis controladas e isolado de interferências externas. Na sala de aula, ao invés de elementos inanimados, lidamos com pessoas, cada uma com a sua individualidade, capacidade cognitiva, experiência prévia, contexto social externo, etc. Medidas podem não ser sensíveis às diferenças de gênero, situação econômica, ou mesmo, à diversidade individual (CRESWELL, 2014). Segundo Creswell (2014), nivelar todos os participantes em uma média estatística desconsidera a singularidade de cada um.

Para a pesquisa em educação, é importante refletir como cada um dos indivíduos, participantes do processo, descreve e interpreta a sua experiência com relação ao objeto em estudo da pesquisa, qual foi a sua reação e em qual contexto apresentaram essa reação. Ou seja, uma abordagem qualitativa pode ser considerada.

Borrego, Douglas e Amelink (2009) argumentam que é preciso expandir o repertório das pesquisas em educação em engenharia de modo a considerar métodos alternativos. Por outro lado, em seu estudo, após a análise dos trabalhos apresentados em uma conferência internacional voltada para o tema, relatam que existe uma forte preferência por métodos quantitativos, principalmente por estudos que relatem resultados provenientes do modelo experimental aplicado em sala de aula. Os autores ainda destacam que mesmo participantes que lamentam a falta de compreensão e aceitação por parte dos revisores com relação aos estudos qualitativos, enaltecem os trabalhos quantitativos.

Longe de discutir concepções filosóficas ou definir se métodos qualitativos ou quantitativos são melhores para aplicar na pesquisa em educação, uma vez que este não é o foco de nosso estudo, escolhemos utilizar a concepção pragmática (CRESWELL, 2010) para este trabalho. Creswell (2010) destaca que o pragmatismo não está vinculado a

nenhum sistema filosófico e de realidade. Na concepção pragmática, os pesquisadores preocupam-se com a solução dos problemas, de modo que se utilizam de todas as abordagens disponíveis para melhor entender a questão em estudo (CRESWELL, 2010). Em vez de se restringirem a apenas um método específico de coletar e analisar os dados (qualitativo ou quantitativo), empregam várias abordagens (CRESWELL, 2010). Paralelamente, autores defendem que a metodologia ou projeto de pesquisa a ser empregado depende da proposta de pesquisa (COHEN; MANION; MORRISON, 2007), isto é, deve ser dirigida pelas questões de pesquisa estabelecidas (BORREGO; DOUGLAS; AMELINK, 2009). Seguindo este raciocínio, a Tabela 4.1 resume as estratégias de investigação utilizadas a fim de atingir os objetivos traçados para responder as questões de pesquisa concebidas para este trabalho.

No decorrer deste capítulo detalharemos as estratégias empregadas ao longo da pesquisa, bem como as questões éticas e ameaças à validade envolvidas.

4.1 MAPEAMENTO SISTEMÁTICO

O objetivo do mapeamento sistemático é proporcionar uma visão geral e abrangente dos estudos existentes sobre determinado tema (KITCHENHAM; CHARTERS, 2007). Após um processo sistemático de identificação e seleção de trabalhos, uma estrutura de classificação para a área de interesse é criada, por meio de uma análise textual desses estudos, de modo que os resultados apresentam a frequência de publicações por categoria desse esquema (PETERSEN et al., 2008). Através desta estratégia de pesquisa exploratória, a cobertura de uma área de pesquisa pode ser determinada (PETERSEN et al., 2008), lacunas podem ser identificadas, onde novos estudos primários ou revisões sistemáticas podem ser necessários (BUDGEN et al., 2008). Ou seja, representa uma tática bastante interessante para estudantes de doutorado, que precisam conhecer o cenário geral da área na qual estarão trabalhando (KITCHENHAM; CHARTERS, 2007). Por estas razões, iniciamos nosso estudo realizando um mapeamento sistemático e, assim, identificamos como projetos de código aberto estão sendo incorporados como uma abordagem de ensino-aprendizagem de engenharia de software no ambiente acadêmico (Ob1).

A Figura 4.1 resume o processo empregado para a realização do mapeamento. A partir da definição das questões de pesquisa, detalhamos o escopo da revisão, estabelecendo a *string* padrão para a pesquisa e as bases de dados digitais que iriam ser consultadas. Após adaptarmos a *string* padrão à sintaxe exigida pelas bases, executamos todas as pesquisas, o que resultou em um total de 2204 publicações. Na etapa de seleção dos estudos, inicialmente eliminamos as publicações duplicadas, depois aplicamos os critérios de inclusão e exclusão pré-definidos, para que apenas as publicações de fato relacionadas ao objeto da pesquisa fossem consideradas. Tendo como base as referências existentes nos artigos selecionados, investigamos outras publicações também relevantes para o propósito da pesquisa, que porventura não haviam sido identificadas pelo processo de busca (processo conhecido como *snow-balling* (BUDGEN et al., 2008)). Ao final trabalhamos com 72 artigos. Com a leitura destes artigos e, direcionados pelas questões de pesquisa, identificamos nove facetas que consideramos importantes para a classificação dos estudos. Por último, extraímos as informações necessárias e classificamos os trabalhos para a construção dos

Tabela 4.1 Procedimentos de investigação empregados ao longo da pesquisa.

Questão de Pesquisa	**Objetivo Específico**	**Estratégia de Pesquisa**
RQ1. Como projetos de código aberto podem ser incorporados como uma abordagem de ensino-aprendizagem de engenharia de software no ambiente acadêmico?	Ob1. Identificar como projetos de código aberto têm sido incorporados como uma abordagem de ensino-aprendizagem de engenharia de software no ambiente acadêmico.	Mapeamento sistemático de literatura
RQ2. Quais objetivos de aprendizagem de engenharia de software podem ser cobertos através dessa abordagem? Em que nível podem ser atingidos?	Ob2. Definir quais objetivos de aprendizagem podem ser cobertos por essa abordagem;	Revisão de literatura
	Ob3. Identificar os resultados reportados pelos estudos relacionados a esta abordagem;	Síntese de evidências
	Ob4. Avaliar a possibilidade do alcance de um subconjunto de objetivos de aprendizagem, pelos estudantes, com a adoção da abordagem.	Estudo de caso e síntese de evidências
RQ3. Quais as consequências para a aprendizagem de engenharia de software, por parte dos estudantes, quando projetos de código aberto são empregados no processo ensino-aprendizagem?	Ob5. Explorar as consequências para a aprendizagem de engenharia de software pelos estudantes quando projetos de código aberto são empregados no processo ensino-aprendizagem;	Estudo de caso e síntese de evidências
	Ob6. Investigar se a adoção de projetos de código aberto permite que o estudante faça a conexão entre teoria e prática;	Estudo de caso
	Ob7. Investigar a percepção do estudante sobre seu contato com o projeto de código aberto, como uma experiência de mundo real;	Estudo de caso
	Ob8. Investigar se e como o uso do projeto de código aberto permite ao estudante sentir-se preparado para o mercado de trabalho.	Estudo de caso
RQ4. Que lacunas identificadas pela indústria na formação dos estudantes a abordagem ajuda a suprir?	Ob9. Investigar se abordagem ajuda a suprir algumas das lacunas de formação identificadas pela indústria.	Comparação dos resultados obtidos com as necessidades da indústria reportadas pela literatura existente.

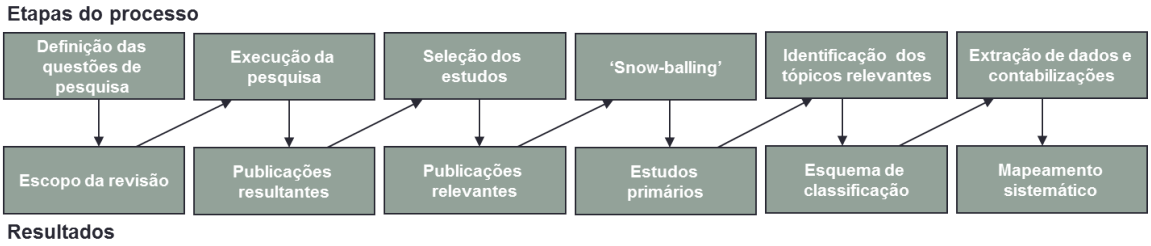

Figura 4.1 Processo do mapeamento sistemático, adaptado de Petersen et al. (2008).

mapas.

As nove facetas identificadas no decorrer do processo e utilizadas para as construções dos mapas foram:

1. Objetivo da aplicação da abordagem (aprender sobre engenharia de software ou especificamente sobre projetos de código aberto);

2. Área da engenharia de software para qual a abordagem foi empregada;

3. Abordagem pedagógica aplicada;

4. Tipo de pesquisa adotado;

5. Perspectiva da avaliação da aprendizagem do estudante;

6. Instrumento de avaliação da aprendizagem do estudante;

7. Como a abordagem foi incorporada ao currículo (ao longo de disciplinas, trabalho de conclusão de curso ou atividades extras);

8. Nível de controle do professor sobre as atividades que o estudante realizava no projeto;

9. Responsabilidade da escolha do projeto a ser trabalhado pelo estudante.

O Capítulo 5 apresenta o resumo dos resultados encontrados. Todavia, para a descrição detalhada do processo e dos resultados obtidos, consultar Nascimento, Bittencourt e Chavez (2015).

A realização do mapeamento foi essencial para definirmos as demais questões de pesquisa e objetivos para o trabalho. Como a maioria dos estudos recuperados tinha como objetivo propor uma forma de aplicar a abordagem ou relatar as lições aprendidas com a sua aplicação, onde nem sempre resultados eram apresentados, resolvemos focalizar a continuidade de nossos estudos na procura por evidências sobre a eficácia da abordagem para a aprendizagem dos estudantes.

4.2 REVISÃO DE LITERATURA E DEFINIÇÃO DOS OBJETIVOS DE APRENDIZAGEM FACTÍVEIS

Como um dos componentes para o julgamento da eficácia da abordagem, utilizamos a análise da possibilidade do alcance de objetivos de aprendizagem. Contudo, para obtermos uma visão abrangente e independente da disciplina onde a abordagem de uso de projetos de código aberto estava sendo aplicada, decidimos primeiramente estabelecer um conjunto de objetivos de aprendizagem em engenharia de software factíveis de serem tratados por esta abordagem (Ob2). Com esta finalidade, realizamos uma revisão de literatura.

Dada a relevância da prática para a engenharia de software, conforme discutimos na contextualização deste trabalho (Capítulo 1), concluímos que o escopo inicial desses objetivos deveria estar fundamentado tanto pela visão acadêmica quanto pela visão da indústria de software. Assim, inicialmente buscamos identificar quais documentos existentes seriam apropriados para este fim. Selecionados esses documentos, correlacionamos os objetivos provenientes da visão acadêmica, visão da indústria e estudos relacionados à abordagem de uso de projetos de código aberto.

Para a justificativa sobre a possibilidade de cobertura de cada objetivo estabelecido, apontamos formas de utilização da abordagem, a partir da experiência pessoal de mais de 10 anos ensinando disciplinas de engenharia de software e da indicação dos estudos relacionados. De acordo com as possiblidades de utilização, acrescentamos a nossa interpretação sobre o impacto que a adoção da abordagem poderia trazer ao alcance do objetivo de aprendizagem em questão. Com vistas a verificar se nossa análise havia sido demasiadamente otimista, discutimos as possibilidades apontadas de utilização da abordagem, bem como, o grau do impacto sugerido, com um professor da área de engenharia de software não envolvido com os demais resultados desta pesquisa. Nos casos onde houve divergência no julgamento do grau de impacto, adotamos a posição pessimista, considerando o menor valor.

Os passos executados, os documentos empregados e os resultados obtidos encontram-se detalhados no Capítulo 3. Convém ressaltar que o conjunto definido teve como propósito essencial servir de base para indicar quais objetivos de aprendizagem poderiam ser investigados nos demais estudos que estaríamos realizando, e permitir a comparação entre os resultados destes estudos. Por sua vez, a análise de cobertura e o grau de impacto ofereceram apenas uma visão inicial da potencialidade da abordagem, que deveria ser comprovada no decorrer da pesquisa.

4.3 SÍNTESE DE EVIDÊNCIAS

Dado que traduzimos o propósito geral de investigar a eficácia da abordagem em entender e descrever a aprendizagem ocorrida a partir do uso de projetos de código aberto, procuramos ampliar a abrangência de nossa investigação, considerando também as evidências reportadas pelos trabalhos selecionados no processo do mapeamento sistemático. A intenção era gerar uma visão descritiva e interpretativa do impacto da abordagem obtido nos estudos relacionados, para assim confirmar, complementar ou mesmo refutar os re-

sultados conseguidos como consequência das nossas próprias experiências com a adoção da abordagem.

Em função da limitação de tempo para a realização de uma revisão sistemática de literatura[1], método que seria adequado para este fim, optamos por considerar os estudos selecionados no processo do mapeamento sistemático. Isto porque mapeamento e revisão sistemáticos representam estudos secundários similares, que utilizam o mesmo processo cuidadoso para a identificação e seleção de artigos relacionados a determinado tema. Entretanto, enquanto que o mapeamento sistemático é geralmente guiado por questões de pesquisa gerais ou múltiplas questões, recupera um número grande de estudos de maneira a obter uma cobertura abrangente e a fase de análise busca a sumarização e totalização dos dados (KITCHENHAM; CHARTERS, 2007), a revisão sistemática procura examinar os estudos primários, criteriosamente selecionados, em profundidade, aplicando meta-análise e/ou sínteses narrativas, a fim de gerar conclusões sobre uma questão de pesquisa particular (KITCHENHAM; CHARTERS, 2007). Ou seja, enquanto o mapeamento pretende prover uma visão geral e abrangente, a revisão tem a intenção de identificar as melhores práticas baseada em evidências empíricas (PETERSEN et al., 2008).

Portanto, retornamos ao escopo dos 72 artigos selecionados no processo de mapeamento e complementamos com a síntese, compreendendo as seguintes atividades: (i) restrição dos estudos, (ii) definição da estratégia para a síntese das evidências e (iii) execução da síntese propriamente dita. As atividades, bem como os resultados obtidos, encontram-se detalhados em 5.2.

Ao realizarmos a síntese das evidências, identificamos e categorizamos os resultados reportados pelos estudos relacionados (Ob3). A categorização das evidências demonstrou ainda outras consequências para a aprendizagem dos estudantes (Ob5). Além de sintetizarmos os resultados, correlacionamo-los aos objetivos de aprendizagem em ES e, assim, detectamos quais destes poderiam ser alcançados por meio da abordagem (Tabela 5.19), a depender das informações fornecidas nos artigos (Ob4). Semelhantemente, procedemos para avaliar se a abordagem ajuda a suprir as deficiências de formação indicadas pela indústria de software (Ob9) (Tabela 5.21).

4.4 ESTUDOS DE CASO

Estabelecidos os objetivos de aprendizagem, para analisarmos a eficácia da abordagem, foi necessário aplicá-la em alguma disciplina de engenharia de software. Todavia, como os objetivos de aprendizagem ficam restritos ao programa da disciplina em si, e ainda precisávamos explorar as consequências para a aprendizagem de engenharia de software quando projetos de código aberto são utilizados como exemplos de projetos de software reais, decidimos aplicar a abordagem em mais de uma disciplina. Nossa intenção foi entender e descrever o impacto da abordagem na aprendizagem dos estudantes, de maneira

[1]Revisão sistemática de literatura ou simplesmente revisão sistemática corresponde a um tipo de estudo secundário, que aplica uma metodologia rigorosa para identificar, analisar e interpretar todas as evidências disponíveis a respeito de uma questão de pesquisa específica, de maneira que o resultado seja imparcial e o máximo possível repetível (KITCHENHAM; CHARTERS, 2007).

a prover informações para que professores da área pudessem avaliar se esta encaixar-se-ia em suas necessidades. Deste modo, quanto mais abrangente a visão gerada, especialmente considerando os objetivos de aprendizagem, mais interessantes seriam os resultados.

A intenção era investigar os objetivos de aprendizagem alcançados (Ob4), as consequências para a aprendizagem (Ob5), particularmente, se a adoção da abordagem permite que o estudante faça a conexão do que é visto na teoria com a prática (Ob6), se o estudante enxerga seu contato com o OSP como uma experiência de mundo real (Ob7), se, com o uso do OSP, o estudante sente-se mais preparado para o mercado de trabalho (Ob8) e, finalmente, se a abordagem ajuda a suprir as deficiências de formação indicadas pela indústria de software (Ob9).

Segundo Creswell (2014), estudo de caso é uma abordagem de pesquisa qualitativa onde o investigador explora, durante um determinado período de tempo, um ou mais sistemas delimitados, reais e contemporâneos, através de uma coleta de dados detalhada, envolvendo múltiplas fontes de informação. Por outro lado, Yin (2010) defende que o estudo de caso é um método de investigação empírica que vai além de uma opção de pesquisa qualitativa, já que pode utilizar tanto evidências quantitativas quanto qualitativas, e que nem sempre precisa incluir a evidência observacional direta, essencial em outras formas de pesquisa qualitativa. Merriam (2009) também admite que estudos de casos incluam análises quantitativas. Enfim, Yin (2010) acrescenta às questões de contemporaneidade, realidade, delimitação de escopo e tempo, características dos estudos de caso, que estes são adequados quando os limites entre o contexto e o fenômeno não são nitidamente evidentes e há muito mais variáveis de interesse do que pontos de dados. Tais características corroboraram para a escolha desse método de pesquisa, uma vez que estas situações estavam presentes no projeto da pesquisa, principalmente porque a aprendizagem e as consequências da adoção da abordagem não podiam ser dissociadas do contexto onde a abordagem foi aplicada.

Neste trabalho, realizamos três estudos de caso. A Tabela 4.2 provê uma visão geral dos estudos de caso efetuados. A posteriori, descreveremos detalhadamente cada um dos estudos. As áreas de conhecimento foco da aplicação da abordagem foram selecionadas de acordo com o programa das disciplinas, onde oportunamente pudemos realizar os estudos de caso. A primeira disciplina foi ofertada pelo orientador desta pesquisa na Universidade Estadual de Feira de Santana, enquanto que as duas outras representam disciplinas ofertadas na Universidade Federal de Sergipe, instituição à qual esta pesquisadora está vinculada, e onde o professor responsável solicitamente concordou com a execução dos estudos. Como a participação na pesquisa foi voluntária, obtivemos uma participação reduzida no primeiro estudo (aproximadamente um terço dos estudantes) e razoável nos outros dois (aproximadamente metade dos estudantes) nas atividades de coleta de dados. Observamos alguma heterogeneidade quanto à obrigatoriedade na realização das atividades e à maturidade acadêmica apresentada pelos estudantes. Finalmente, como, no período que realizamos o estudo EC1, ainda estávamos na fase de estreitamento dos objetivos da pesquisa, este assumiu apenas um caráter exploratório. Convém destacar que este estudo, em conjunto com o mapeamento sistemático, foram essenciais para a formulação das questões de pesquisa. Nos estudos seguintes (EC2 e EC3), com objetivos definidos, assumimos uma estratégia mais investigativa, sem contudo descartar o caráter

exploratório, uma vez que este era necessário para atender o propósito de entender e interpretar as consequências da adoção da abordagem.

Tabela 4.2 Visão geral dos estudos de caso realizados.

Id.	**Área de Conhecimento Foco**	**Local**	**Período**	**Número estudantes particip.**	**Realização das atividades**	**Maturidade Acadêmica**	**Caráter do estudo de caso**
EC1	Evolução de Software	UEFS	2013.2	9	Obrigatória	Variada	Exploratório
EC2	Testes de Software	UFS	2015.2	15	Opcional	Final do curso	Investigativo e exploratório
EC3	Requisitos de Software	UFS	2015.2	15	Obrigatória	Meio do curso	Investigativo e exploratório

O caráter dos estudos influenciou a escolha e aplicação dos instrumentos de coleta de dados. Enquanto, no primeiro, escolhemos utilizar entrevistas, observações das aulas e questionário, na tentativa de visualizar todo o processo, nos outros, descartamos a utilização das observações das aulas uma vez que se demonstraram pouco producentes no sentido da relação custo/benefício, acrescentamos a análise de documentos e a aplicação de questionários pré- e pós-intervenção, além da realização das entrevistas.

Para a análise dos dados, empregamos estatísticas descritivas e testes estatísticos adequados à distribuição dos dados, para os dados quantitativos, e o processo indutivo-dedutivo descrito por Merriam (2009), para os dados qualitativos. Seguindo este processo, em um primeiro momento, usamos o método de codificação aberta, buscando descobrir informações que pudessem ser relevantes para a pesquisa. Simultaneamente, após certo volume de códigos criados, passamos a fazer a codificação axial, agrupando os códigos inter-relacionados e criando uma hierarquia de temas. À medida que a codificação prosseguia, buscamos verificar códigos existentes que poderiam ser utilizados ou criamos novos códigos, quando necessário. Com o término da codificação de todos os dados, revisamos os códigos resultantes, refinamos a hierarquia de temas e geramos memorandos para os temas-chave. Durante todo o processo, buscamos identificar relacionamentos entre códigos e temas criados. Ao longo de todo o texto, o termo 'categorias' também refere-se aos códigos criados.

Cabe salientar que realizamos a análise e interpretação dos dados individualmente para cada estudo. O contexto de aplicação da abordagem, a metodologia usada na coleta e análise dos dados, bem como os resultados obtidos em cada estudo de caso encontram-se detalhados em capítulos específicos: EC1 no Capítulo 6, EC2 no Capítulo 7 e EC3 no Capítulo 8. Somente ao final dos trabalhos, comparamos e discutimos os achados entre os estudos, como pode ser visto no Capítulo 9.

4.5 JULGAMENTO DO ALCANCE DOS OBJETIVOS DE APRENDIZAGEM

De acordo com o objetivo Ob4 traçado para a pesquisa, foi preciso avaliar a possibilidade de alcance de um subconjunto de objetivos de aprendizagem proporcionado pelo uso de projetos de código aberto. Usualmente, quando falamos em avaliação de aprendizagem, imediatamente imaginamos como será o teste que iremos aplicar. Entretanto, a aplicação de testes para a avaliação da aprendizagem com o intuito da pesquisa apresentava alguns problemas:

- Representaria uma solução intrusiva, porque estaríamos impondo mais um mecanismo de avaliação, uma vez que o professor já estava avaliando os estudantes por meio de provas;
- Representaria uma nova atividade para os estudantes que demandaria tempo além da própria realização das atividades no projeto;
- Para que a intrusão não fosse tão incisiva, o teste deveria ser respondido rapidamente e em horário extraclasse, de modo a não atrapalhar o andamento da disciplina;
- Testes com questões de múltipla escolha, que possibilitam respostas rápidas, não avaliam habilidades relacionadas à engenharia de software tais como implementar casos de teste ou elaborar diagramas;
- A avaliação de habilidades corresponde a uma atividade complexa. Era preciso elaborar questões que julgassem o alcance de cada objetivo de aprendizagem que também não representam objetivos simples de serem medidos;
- Poderia ocorrer a falta de comprometimento dos estudantes na solução das atividades propostas, por não existir nenhuma nota associada e ainda por ser realizado extraclasse.

Moody e Sindre (2003), baseados em pesquisas que mostram que a percepção dos estudantes sobre sua aprendizagem é fortemente relacionada às notas obtidas nos testes, apontam esta percepção como forma alternativa para medir a eficácia de estratégias didáticas empregadas. Por sua vez, Savi (2011) relata um conjunto de pesquisas que indicam um nível apenas moderado de correlação entre a autoavaliação dos estudantes e as notas obtidas nos testes, após a correção dos professores. Todavia, ao mesmo tempo, indica estudos que relatam que também há variação entre as notas atribuídas por diferentes professores ao mesmo trabalho de um estudante. Consequentemente, o autor conclui que, se existem variações na avaliação entre professores, é aceitável que haja variação considerando as autoavaliações dos estudantes.

Após estas considerações, nos estudos EC2 e EC3, optamos por utilizar tanto a percepção do estudante quanto a percepção do professor para avaliar o alcance dos objetivos de aprendizagem proporcionado pela adoção de projetos de código aberto. Capturamos a percepção dos estudantes sobre a sua aprendizagem através de três dimensões. A primeira

foi obtida através do *nível de concordância* [2] *do estudante sobre a sua competência em relação a cada objetivo de aprendizagem* sendo tratado no estudo, com questões idênticas nos questionários aplicados antes e depois das atividades realizadas no OSP, de modo que pudéssemos aplicar testes estatísticos para identificar alguma melhoria nesta competência. A segunda representou o *nível de concordância* [3] *sobre a contribuição da realização das atividades no projeto para o alcance de cada competência expressa em cada objetivo de aprendizagem*, questionada após o término do projeto. Por último, a terceira considerou *o relato dos estudantes quando questionados ao longo das entrevistas, inicialmente se existiu aprendizagem e, posteriormente, em caso afirmativo, qual foi a aprendizagem proporcionada pela execução das atividades.* No que diz respeito à percepção do professor, avaliamos os produtos gerados pelos estudantes ao realizarem as atividades solicitadas no OSP.

Para termos um julgamento final a respeito do alcance de cada objetivo de aprendizagem proporcionado pela abordagem, foi necessário especificarmos um conjunto de heurísticas, priorizando as dimensões e critérios a serem aplicados. O propósito de estabelecermos estas heurísticas foi prover consistência e transparência para o nosso julgamento, bem como possibilitar a repetibilidade do processo.

Estabelecemos a seguinte ordem de prioridade de avaliação para as dimensões citadas:

1. Melhora significativa, através de testes estatísticos, para a competência descrita no objetivo de aprendizagem;

2. Avaliação dos produtos gerados pelos estudantes;

3. Percentual de concordância sobre a contribuição da realização das atividades no projeto para o alcance da competência expressa no objetivo de aprendizagem;

4. Existência de relato sobre aprendizagem correspondente ao objetivo de aprendizagem.

Na análise realizada, era possível que uma ou mais dimensões, ou mesmo todas, estivessem presentes como resultado para cada objetivo de aprendizagem, em virtude do caráter investigativo e exploratório empregado nos estudos EC2 e EC3. Portanto, de acordo com os critérios de cada dimensão e a combinação existente entre essas dimensões, consideramos os seguintes julgamentos finais possíveis:

(S) Sim, o objetivo de aprendizagem foi alcançado através da abordagem;

(P) Parcial, o objetivo de aprendizagem foi parcialmente alcançado através da abordagem;

(N) Não, o objetivo de aprendizagem não foi alcançado através da abordagem;

[2] Utilizamos a escala Likert de quatro pontos: 'discordo plenamente', 'discordo', 'concordo' e 'concordo plenamente'.

[3] Utilizando a mesma escala.

Precisávamos ser mais pragmáticos no julgamento da dimensão correspondente ao percentual de concordância sobre a contribuição da realização das atividades no projeto para o alcance da competência expressa no objetivo de aprendizagem, uma vez que o resultado poderia ser qualquer valor entre zero e 100%. Assim, optamos por definir faixas, de certo modo rigorosas, para considerarmos que a abordagem contribuiu realmente, parcialmente ou não contribuiu para tal alcance. As faixas definidas foram as seguintes:

Entre 80 e 100% O resultado confirma a contribuição da abordagem, devido à concordância de uma maioria absoluta qualificada dos estudantes;

Entre 60 e 79% O resultado confirma apenas parcialmente a contribuição da abordagem, devido à concordância de uma maioria simples dos estudantes;

Menor ou igual a 59% O resultado não confirma a contribuição da abordagem, uma vez que menos de 60% dos estudantes reconheceram que houve tal contribuição.

Dois fatores influenciaram no estabelecimento dos valores limites para estas faixas, resumidos em 80 e 60%. O primeiro refere-se à quantidade de participantes nos estudos ter sido pequena[4], de modo que valores altos de concordância (maiores que 80%) dificilmente seriam obtidos. O segundo representa o fator psicológico de possível comportamento aquiescente dos estudantes ao responderem questionários.

Ray (1983) explica o comportamento aquiescente, como a atitude de alguns indivíduos de concordar com quase todas as proposições que lhe são colocadas, seja com o objetivo de agradar, seja por serem indiferentes à tarefa de responder questionários. Billiet e Mcclendon (1998), apesar de concordarem com a existência deste comportamento, apontam outros pesquisadores que negam a importância dessa conduta de concordância em pesquisas de personalidade e teorias de medição. Gee (2015), investigando o comportamento dos estudantes ao responderem questionários de avaliação de seus cursos cujas questões utilizavam a escala Likert, concluiu que a familiaridade e a regularidade com que este tipo de avaliação é aplicado gera algumas vezes apatia, complacência, falta de compreensão e engajamento com as questões. Contudo, o autor também identificou como explicação para este 'desprezo' por parte dos estudantes, a falta de *feedback* ou ações aparentes de melhorias nos cursos sendo avaliados.

Sem entrar no mérito da discussão da ocorrência ou não da aquiescência, uma vez que este não era o foco da nossa pesquisa, nos estudos EC2 e EC3, ao esclarecermos os objetivos da pesquisa e, consequentemente, dos questionários, procuramos conscientizar os estudantes da relevância de sua participação e da franqueza em suas respostas, para o sucesso do estudo de validação ou não do uso de projetos de código aberto na educação em engenharia de software. Mesmo assim, preferimos assumir uma atitude pessimista, ao mesmo tempo rigorosa, estabelecendo o limite mínimo de 60% de concordância para considerarmos a contribuição da abordagem como pelo menos parcial.

A Tabela 4.3 resume as heurísticas estabelecidas. Nos casos particulares de ter ocorrido a melhora significativa da aprendizagem obtida por testes estatísticos, ou a avaliação

[4]Consideramos apenas 14 respostas no estudo EC2 e nove respostas no Estudo EC3 para este tipo de questão, conforme explicado nos capítulos onde os estudos são detalhados.

dos trabalhos ter sido satisfatória, ou se os trabalhos apresentassem poucos problemas, consideramos que a abordagem havia proporcionado o alcance do objetivo de aprendizagem (Alcance do objetivo de aprendizagem igual a 'S'). Quando o trabalho não estava satisfatório, utilizamos a análise das demais dimensões, de maneira a verificar, de acordo com a combinação obtida, se ao menos a abordagem proporcionara o alcance parcial do objetivo de aprendizagem. Quando o objetivo não foi diretamente investigado por meio dos trabalhos, as duas últimas dimensões foram essenciais. Quando contabilizamos o percentual de concordância dos estudantes sobre a contribuição da abordagem para o alcance do objetivo de aprendizagem em questão e este foi maior ou igual que 80%, consideramos que a abordagem havia proporcionado o seu alcance (Alcance do objetivo de aprendizagem igual a 'S'). Quando este estava entre 60 e 80%, representando a maioria simples, e ainda existiam relatos espontâneos dos estudantes sobre aprendizagem correspondente ao objetivo de aprendizagem, também consideramos que a abordagem havia proporcionado o alcance do objetivo de aprendizagem (Alcance do objetivo de aprendizagem igual a 'S'). Quando o percentual de concordância estava entre 60 e 80% e não existiam relatos dos estudantes, consideramos que a abordagem proporcionara apenas o alcance parcial do objetivo (Alcance do objetivo de aprendizagem igual a 'P'). Quando o percentual era menor que 60%, se existissem relatos, consideramos que a abordagem proporcionara o alcance parcial (Alcance do objetivo de aprendizagem igual a 'P'). Se não existissem relatos, consideramos que a abordagem não proporcionou o alcance do objetivo (Alcance do objetivo de aprendizagem igual a 'N'). Por fim, para os objetivos que não foram contabilizados por percentuais de concordância, a única dimensão que consideramos foi o relato dos estudantes. Julgamo-los segundo a existência ou não desses relatos. Para maior clareza, descrevemos as heurísticas aplicadas também no formato de um algoritmo, ilustrado na Figura 4.2.

Convém ressaltar que essas heurísticas bem como os limites estabelecidos podem ser calibrados em trabalhos futuros.

4.6 COMPARAÇÃO DE RESULTADOS

Para verificarmos se a adoção de projetos de código aberto como exemplo de projeto de software a ser trabalhado pelos estudantes ajuda a suprir algumas das lacunas de formação apontadas pela indústria (Ob9), comparamos se os resultados positivos encontrados com a realização das demais investigações, especialmente os objetivos de aprendizagem alcançados, estavam relacionados a estas deficiências.

Obviamente esse método é demasiadamente simplório, já que não captura a percepção de membros da indústria sobre melhorias em relação a tais deficiências. Todavia representa um alerta, para que pesquisas na área da educação em engenharia de software tenham também este propósito. Podendo inclusive servir de reflexão preliminar a fim de direcionar para investigações mais adequadas no futuro.

Tabela 4.3 Heurística utilizada para o julgamento da contribuição da adoção de projetos de código aberto para o alcance dos objetivos de aprendizagem.

Critérios	**Resultado obtido em cada dimensão**														
Houve melhora significativa	V	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Trabalhos satisfatórios ou com poucos problemas	-	V	-	-	-	-	-	-	-	-	-	-	-	-	-
Trabalhos não satisfatórios ou com muitos problemas	-	-	V	V	V	V	V	V	-	-	-	-	-	-	-
Não houve julgamento do objetivo no trabalho	-	-	-	-	-	-	-	-	V	V	V	V	V	V	V
Concordância ≥80%	-	-	V	-	-	-	-	-	V	-	-	-	-	-	-
Concordância < 80% E Concordância ≥ 60%	-	-	-	V	-	-	-	-	-	V	V	-	-	-	-
Concordância < 60%	-	-	-	-	-	V	V	-	-	-	-	V	V	-	-
Não houve contabilização da concordância	-	-	-	-	V	-	-	V	-	-	-	-	-	V	V
Existe relato	-	-	-	-	V	V	-	-	-	V	-	V	-	V	-
Não existe relato	-	-	-	-	-	-	V	V	-	-	V	-	V	-	V
Alcance do Objetivo	**S**	**S**	**P**	**P**	**P**	**N**	**N**	**N**	**S**	**S**	**P**	**P**	**N**	**S**	**N**

(V) indica a ocorrência do resultado, (S) que o objetivo foi alcançado, (P) que o objetivo foi parcialmente alcançado e (N) que o objetivo não foi alcançado.

4.7 QUESTÕES ÉTICAS

Como qualquer outro estudo que envolve coletar dados das pessoas e sobre as pessoas, questões éticas precisaram ser tratadas em várias etapas da pesquisa (CRESWELL, 2010). Ao longo do projeto procuramos seguir os quatro princípios presentes em Wohlin et al. (2012). São eles: valor científico, benefícios que sobrepõem os riscos, consentimento informado e confidencialidade.

A preocupação com o valor científico ocorreu desde a elaboração do projeto de pesquisa até a sua conclusão, com o relato dos resultados, uma vez que procuramos seguir as diretrizes definidas pela literatura para os métodos de investigação propostos. O contexto da pesquisa abrangeu a melhoria da educação em engenharia de software., que conforme discutido na introdução deste documento, seus resultados, além de proverem informações para os demais pesquisadores sobre o tema, propiciam evidências científicas sobre a adoção de projetos de código aberto para este fim, aos educadores interessados.

Optamos por não utilizar grupos de controle, de modo que todos os estudantes foram submetidos à abordagem. A dificuldade de realização das atividades no OSP foi o principal risco identificado para os estudantes. Contudo, procuramos atenuá-lo provendo suporte a sua execução, bem como um roteiro para a configuração do ambiente que seria necessário, além de exemplos de como as atividades deveriam ser realizadas. Na avaliação dos trabalhos, ponderamos entre as dificuldades enfrentadas e as soluções apresentadas junto ao professor responsável pela atribuição das notas, para que os estudantes não saíssem prejudicados. De fato, após receberem o resultado da avaliação, nenhum estudante fez qualquer reinvindicação.

No início dos estudos de caso, explicamos os objetivos da pesquisa e obtivemos o consentimento de participação voluntária dos estudantes através da assinatura do termo de consentimento livre e esclarecido. No texto do termo, além de detalharmos os objetivos da pesquisa e persuadirmos o estudante sobre a relevância de sua participação, explicamos como seria esta participação, que a mesma seria voluntária e que, a qualquer momento, ele poderia declinar, sem que lhe fosse imputada qualquer tipo de penalidade. Ainda esclarecemos que as informações prestadas seriam confidenciais e que não seriam utilizadas para fins de sua avaliação ao longo da disciplina.

Poucos estudantes optaram por não participar dos estudos, de maneira que não consideramos os resultados de seus trabalhos. Quanto à coleta de dados sobre a percepção do estudante sobre a abordagem didática empregada, direcionamo-la somente aos estudantes que aceitaram participar.

Quanto à confidencialidade, foram tomados dois cuidados. O primeiro está relacionado à privacidade dos dados, de maneira que apenas o grupo de pesquisa envolvido teve acesso aos dados brutos. O segundo diz respeito ao anonimato dos participantes. Ainda durante a fase de transcrição das entrevistas realizadas, atribuímos códigos aos estudantes entrevistados. O mesmo aconteceu para os questionários, uma vez que estes precisaram ser nomeados para que pudéssemos realizar a análise sobre a aprendizagem pré- e pós-intervenção. Deste modo, nenhum estudante foi verdadeiramente identificado na fase de análise. O relato dos resultados em qualquer documento utilizou o formato generalizado e nas situações em que foi necessário explicitar algum extrato de texto individual, aplicamos

os códigos criados. Por exemplo, SE1, ST1, SR1, para estudantes participantes nos estudos onde a abordagem foi adotada para a aprendizagem sobre evolução, testes e requisitos, respectivamente.

Ainda como questão ética, a redação final da pesquisa consistiu em relatos precisos e verídicos, evitando-se suprimir qualquer informação que fosse importante ao estudo.

4.8 CONSIDERAÇÕES SOBRE A VALIDADE

Como mencionamos no início deste capítulo, empregamos a concepção pragmática para a definição dos métodos de investigação que utilizaríamos ao longo da pesquisa. Por conseguinte, com o propósito de identificar e entender as possíveis consequências para a aprendizagem quando projetos de código aberto são utilizados como exemplos de projetos de software reais a serem trabalhados pelos estudantes, coletamos muitos dados qualitativos. Tratando-se da validade de estudos qualitativos, Merriam (2009) defende que termos como credibilidade, consistência e transferibilidade são mais adequados do que validade interna, confiabilidade e validade externa. Segundo a autora, a validade interna lida com a questão de como as descobertas da pesquisa retratam a realidade, contudo a realidade em pesquisas qualitativas é holística, multidimensional e muda a todo o instante. Portanto a validade é algo relativo que deve ser avaliado em relação às propostas e circunstâncias da pesquisa (MAXWELL, 2005 apud MERRIAM, 2009, p. 214). O pesquisador qualitativo não pode capturar uma 'verdade absoluta', mas pode utilizar uma série de estratégias para aumentar a 'credibilidade' dos seus achados (MERRIAM, 2009). No que concerne à confiabilidade, Merriam (2009) argumenta que o comportamento humano nunca é estático, logo, a replicação de um estudo qualitativo não resultará nos mesmos resultados, além de que, várias interpretações são possíveis para os mesmos dados. Todavia, isto não representa que um estudo específico não tenha seus méritos. Na verdade, é preciso que os resultados estejam 'consistentes' com os dados coletados (MERRIAM, 2009). Por último, sobre a questão das descobertas de um estudo poderem ser generalizadas (validade externa), como os achados nos estudos qualitativos refletem situações peculiares de um contexto particular, Merriam (2009) justifica que estes achados podem ser apenas extrapolados ou transferidos para condições semelhantes. Consoante a autora, a pessoa interessada é que vai decidir se os achados podem ou não ser aplicados para a sua situação particular.

A fim de obtermos credibilidade, consistência e transferibilidade para nossos resultados, utilizamos algumas estratégias sugeridas por Merriam (2009) e Creswell (2010). Foram elas:

- Cuidados com os instrumentos de coleta de dados utilizados - Discutimos prévia e amplamente com o orientador sobre as perguntas básicas a serem realizadas nas entrevistas, bem como sobre as questões e formatos dos questionários, para que estes instrumentos estivessem alinhados aos objetivos da pesquisa. Validamos ainda a compreensão das questões presentes nos questionários com dois estudantes que já haviam cursado as disciplinas.

- Cuidados na preparação e análise dos dados - As transcrições das entrevistas foram revisadas por uma pessoa diferente da responsável pela execução da atividade, para a detecção de erros. Os códigos gerados durante o processo de codificação aberta e axial foram constantemente revisados e comparados aos dados, a fim de verificar a existência de desvios, isto é, a perda de representatividade devido a alguma mudança no significado do código no decorrer do processo.

- Transparência da percepção do pesquisador sobre o objeto em estudo - Quanto à nossa crença sobre a adoção de projetos de código aberto na educação em engenharia de software, relatamos que acreditamos na contribuição benéfica para a formação profissional do estudante, em virtude da experiência de mundo real proporcionada.

- Informações negativas ou discrepantes também foram reportadas - Expressamos ao longo do texto, várias situações negativas da abordagem (testes estatísticos não significativos, problemas nos trabalhos realizados pelos estudantes, relatos negativos dos estudantes, objetivos de aprendizagem não alcançados, deficiências, onde não houve contribuição da abordagem para o seu suprimento, além das dificuldades enfrentadas pelos estudantes e professor).

- Triangulação de informações - Aplicamos a triangulação de resultados provenientes de diferentes métodos e fontes de informação. Quanto aos métodos, cruzamos as informações geradas entre três estudos de caso distintos e trabalhos relacionados. Lidamos com dados quantitativos e qualitativos obtidos de diversas fontes de informações - estudantes, professor e avaliação dos produtos gerados pelos estudantes.

- Variação máxima - Buscando visualizar a abrangência da abordagem, empregamos esta estratégia em áreas de conhecimento distintas da engenharia de software. Na coleta de dados individualizada, procuramos entrevistar estudantes que pertenciam a cada uma das equipes formadas para a execução das atividades, bem como estudantes que apresentavam níveis diferentes de experiência prévia com projetos reais. Ainda como fonte de variação, realizamos os estudos de caso com estudantes de níveis de maturidade acadêmica desiguais.

- Descrição rica e densa do contexto - Descrevemos cada estudo de caso detalhando o cenário, população envolvida, como a abordagem foi aplicada, instrumentos utilizados na coleta e método empregado para a análise dos dados. Tais informações servem para enriquecer a interpretação dos resultados, possibilitando que sejam transferidos para outras situações.

- Trilha de auditoria - No decorrer do texto, procuramos expor toda a linha de raciocínio empregada para chegarmos aos resultados encontrados. Apresentamos os extratos que nos levaram a criar as categorias; explicamos as razões de estabelecermos relacionamentos entre os temas identificados; expusemos os critérios aplicados para todos os julgamentos realizados e, finalmente, discutimos como chegamos às conclusões para o estudo.

- Revisão por pares - Realizamos a revisão por pares com a participação do orientador do projeto durante todo o processo de codificação e análise dos resultados. Tudo foi discutido, argumentos foram dados e decisões foram tomadas.

Contudo, ainda podemos relatar algumas questões que ameaçam a validade de nossos estudos.

- Viés do pesquisador - Intrínseca às pesquisas que envolvem a análise qualitativa, em virtude da sempre presente interpretação pessoal. Por mais que estejamos engajados para que ela não ocorra, esta influência sempre existirá.

- Direcionar a abordagem para o alcance dos objetivos de aprendizagem - Como o nosso propósito era investigar se a abordagem seria eficaz de acordo com o alcance de determinados objetivos de aprendizagem, direcionamos as atividades a serem realizadas pelos estudantes, para que eles pudessem alcançar alguns destes objetivos, inclusive acrescentando outros conteúdos, não vistos anteriormente pelos estudantes.

- A avaliação dos resultados do trabalhos realizados pelos estudantes não isolou a influência da adoção do OSP - Não avaliamos o conhecimento apresentado pelos estudantes antes da adoção, em virtude dos problemas envolvidos, discutidos em 4.5.

- A avaliação dos resultados do trabalhos realizados pelos estudantes considerou apenas a nossa interpretação - Dada a profundidade da análise necessária para a pesquisa, nenhum outro pesquisador envolveu-se com essa atividade. Em contrapartida, procuramos ser realmente criteriosos a esse respeito, como pode ser visto nas seções específicas sobre o alcance dos objetivos de aprendizagem dos estudos EC2 e EC3 (ver Subseções 7.2.3 e 8.2.3, respectivamente).

- Amostras pequenas - Poucos estudantes responderam a ambos os questionários (pré- e pós-intervenção), de modo que as amostras terminaram ficando ainda menores para a realização dos testes estatísticos;

- Possibilidade do comportamento aquiescente do estudante - Conforme discutido em 4.5.

- Não houve auditoria externa - Ainda não submetemos os resultados dos estudos, com exceção do mapeamento sistemático, na forma de artigos científicos, a revistas ou a conferências relacionadas à educação em computação ou engenharia.

Especificamente sobre a síntese das evidências reportadas pelos trabalhos relacionados, relatamos outras ameaças:

- Heterogeneidade de contexto dos estudos - Apesar de nos preocuparmos com o contexto dos estudos, ao realizarmos a síntese, alguma situação específica pode ter escapado de nossa análise.

- Acesso limitado aos dados reportados - Devido à limitação de espaço, comum em artigos publicados em conferências, os autores apresentam apenas os resultados que julgam relevantes e no formato que acham interessante. Por conseguinte, pudemos aproveitar na síntese somente os valores reportados e, algumas vezes, precisamos extraí-los de modo aproximado dos gráficos disponibilizados.

- Heterogeneidade dos dados reportados - Como os estudos apresentavam os resultados utilizando diferentes formatos e métricas, precisamos estabelecer heurísticas para uma interpretação comum. Contudo esta interpretação comum pode levar a perdas para determinados resultados e ganhos para outros.

- Variação de qualidade entre os estudos - Como a nossa intenção era ter uma visão abrangente sobre as possíveis consequências para a aprendizagem dos estudantes com a aplicação da abordagem, não fomos muito rigorosos quanto aos critérios de qualidade para a restrição dos artigos a serem utilizados.

- Falta de experiência da pesquisadora - Segundo Cruzes et al. (2015), independente do método empregado para a síntese, a experiência do pesquisador irá influenciar as conclusões obtidas.

Procuramos atenuar essas ameaças fornecendo transparência ao processo executado, através da apresentação das evidências reportadas pelos estudos associadas às categorias e interpretações resultantes, além de descrevermos todo o processo utilizado na interpretação destas evidências.

Finalmente, retornando à questão de generalização dos resultados de todo o trabalho, lembramos que é impossível assumir que todos os estudantes reagirão da mesma maneira, com relação à aprendizagem, quando submetidos a determinado método didático (MOON, 2004). Como vimos no Capítulo 2, o estado da estrutura cognitiva de cada indivíduo influencia o que será selecionado e assimilado pelo mesmo em determinado momento, bem como fatores como condições emocionais, motivação, contexto e interação social existente. Portanto, os resultados encontrados não podem ser generalizados. Porém, a partir da descrição do contexto e interações ocorridas, interessados podem transferi-los para outras situações particulares.

Nos próximos capítulos apresentamos os resultados obtidos em cada investigação: mapeamento sistemático e síntese de evidências, EC1, EC2 e EC3.

```
Se houve melhora significativa para a aprendizagem
    alcance = Sim
senão
    Se os trabalhos eram satisfatórios ou apresentavam poucos problemas
       alcance = Sim
    senão
       Se (os trabalhos não estavam satisfatórios ou apresentavam muitos problemas) E
          (concordância >= 60)
           alcance = parcial
       senão
           Se (os trabalhos não estavam satisfatórios ou apresentavam muitos problemas) E
              (não houve contabilização das concordâncias) E (existem relatos)
               alcance = Parcial
           senão
               Se (os trabalhos não estavam satisfatórios ou apresentavam muitos problemas) E
                  (concordância < 60)
                   alcance = Não
               senão
                   Se (os trabalhos não estavam satisfatórios ou apresentavam muitos problemas) E
                      (não houve contabilização das concordâncias) E (não existem relatos)
                      alcance = Não
                   senão
                      Se (não houve julgamento do objetivo no trabalho) E (concordância >= 80%)
                         alcance = Sim
                      senão
                          Se (não houve julgamento do objetivo no trabalho) E (concordância < 80%) E
                             (concordância >= 60%) E (existem relatos)
                              alcance = Sim
                          senão
                              Se (não houve julgamento do objetivo no trabalho) E (concordância < 80%) E
                                 (concordância >= 60%) E ( não existem relatos)
                                 alcance = Parcial
                              senão
                                 Se (não houve julgamento do objetivo no trabalho) E (concordância < 60%) E
                                    (existem relatos)
                                     alcance = Parcial
                                 senão
                                    Se (não houve julgamento do objetivo no trabalho) E (concordância < 60%) E
                                       (não existem relatos)
                                        alcance = Não
                                    senão
                                        Se (não houve julgamento do objetivo no trabalho) E (não houve contabilização
                                           da concordância) E (existem relatos)
                                           alcance = Sim
                                        senão
                                           Se (não houve julgamento do objetivo no trabalho) E (não houve
                                              contabilização da concordância)  E (não existem relatos)
                                               alcance = Não
                                           fim-se
                                        fim-se
                                    fim-se
                                 fim-se
                              fim-se
                          fim-se
                      fim-se
                   fim-se
               fim-se
           fim-se
       fim-se
    fim-se
fim-se
```

Figura 4.2 Algoritmo de aplicação das heurísticas para o julgamento da contribuição da adoção de projetos de código aberto para o alcance dos objetivos de aprendizagem.

Capítulo 5

COMO OSP TÊM SIDO INCORPORADOS NA EES E QUAIS AS EVIDÊNCIAS EXISTENTES

O ponto de partida dessa pesquisa foi a investigação de como projetos de código aberto têm sido utilizados na educação em engenharia de software (EES). Para atender a este objetivo, realizamos um mapeamento sistemático da literatura conforme descrito em 4.1. A intenção era ter uma visão geral sobre os estudos já publicados que relacionavam a educação em engenharia de software com o emprego de projetos de código aberto e assim direcionar nossa pesquisa. A abordagem a ser investigada representava a utilização do OSP não como ferramenta de apoio ao desenvolvimento de software ou de configuração de laboratórios, mas como projeto de software, cujo código pode ser visualizado, analisado, modificado, testado ou mesmo documentado pelos estudantes.

A partir do mapeamento pudemos identificar as comunidades interessadas na abordagem, com qual propósito e como a abordagem tem sido aplicada. Posteriormente, com o estabelecimento dos objetivos finais para a pesquisa, decidimos retornar aos estudos selecionados e perscrutar em mais detalhe as evidências reportadas pelos mesmos.

Neste capítulo apresentamos um resumo dos resultados do mapeamento, complementamos tais informações com a síntese das evidências reportadas e, por fim, encerramos com a integração dos resultados desta síntese com os objetivos traçados para a pesquisa.

5.1 QUEM, PARA QUE E COMO OSP TÊM SIDO UTILIZADOS NA EES

Os resultados obtidos com o mapeamento sistemático de literatura mostram que existem comunidades importantes interessadas nesse tema, especialmente nos Estados Unidos, Canadá e Europa. Estas comunidades têm produzido iniciativas interessantes e diversas ao longo dos anos.

Entre as publicações identificadas, propostas de solução são o tipo de produção mais comum, seguidas por relatos de experiência. Uma observação é que poucos estudos expressam algum rigor metodológico de pesquisa em sua condução. Vários estudos não especificam a forma de avaliação da abordagem com relação à aprendizagem dos estudantes. Dentre os trabalhos que realizam essa avaliação, a maioria capta a percepção

do próprio estudante sobre a sua aprendizagem por meio de questionários. Artefatos, relatórios e apresentações são os principais instrumentos usados pelos professores para avaliar a aprendizagem. Contudo, raramente estes resultados são discutidos no sentido de avaliar a eficácia da abordagem.

A grande maioria dos estudos tem adotado projetos de código aberto para a aprendizagem de conceitos e princípios de engenharia de software. Porém, a aprendizagem sobre projetos de código aberto em si também tem sido o propósito de alguns estudos, enquanto que outros estudos apresentam as duas finalidades. A maioria dos estudos enfoca a aprendizagem de engenharia de software em geral, sendo encontrados poucos trabalhos em áreas específicas como *design*/arquitetura, desenvolvimento/construção ou evolução de software.

Raramente os estudos apresentam a teoria pedagógica na qual estão fundamentados. A maioria segue o método tradicional de ensino, no qual os estudantes assistem a aulas teóricas e devem desenvolver/participar de um projeto como atividade prática.

A abordagem de uso de OSP tem sido incorporada aos currículos dos cursos ao longo das disciplinas, como trabalhos de conclusão de curso, como atividades extras, ou mesmo como uma combinação dessas possibilidades. A primeira opção representa a forma mais encontrada nos estudos.

Outros dois aspectos detectados no mapeamento ajudam a descrever como a abordagem tem sido empregada. O primeiro diz respeito à escolha do software a ser trabalhado. Os estudos relatam situações em que o professor define o projeto com o qual os estudantes irão trabalhar, disponibiliza uma lista de projetos a partir da qual o estudante poderá escolher uma das alternativas ou, finalmente, deixa a escolha de forma livre, isto é, o estudante poderá escolher qualquer OSP. A situação mais comum foi a primeira, ou seja, o projeto é pré-definido pelo professor.

O segundo aspecto representa o nível de controle do professor com relação ao projeto e atividades que serão realizadas pelos estudantes. As seguintes alternativas foram detectadas: (i) 'nenhum controle', quando os estudantes realizam as atividades de acordo com as necessidades da comunidade do projeto, a aceitação da contribuição dos estudantes por parte da comunidade tem peso importante na avaliação final, e o professor apenas acompanha as atividades dos estudantes; (ii) 'definição interna e aprovação externa', quando há bastante interação do professor com a comunidade, de maneira que, por exemplo, uma nova funcionalidade desenvolvida no ambiente acadêmico é submetida e aprovada pela comunidade externa; (iii) 'controle interno', quando faz-se um *branch* do código do projeto, o professor define as atividades a serem realizadas e as avalia, a interação com a comunidade, se existir, é muito pequena, uma vez que não é obrigatória, e as contribuições também não são necessariamente submetidas para a comunidade, e, por fim, (iv) 'controle total', quando o projeto é originado dentro da instituição de ensino e o professor faz parte do núcleo da comunidade. Neste aspecto, encontramos um balanceamento entre as alternativas de 'controle interno' e 'nenhum controle'.

Para maiores detalhes, inclusive a visão geral dos objetivos, contribuições e métodos aplicados nos 72 artigos que utilizam OSP na educação em engenharia de software selecionados pelo processo sistemático, ver Nascimento, Bittencourt e Chavez (2015).

5.2 SÍNTESE DAS EVIDÊNCIAS REPORTADAS PELOS ESTUDOS SELECIONADOS

Após estabelecermos que iríamos investigar quais objetivos de aprendizagem poderiam ser alcançados e quais as consequências para a aprendizagem em engenharia de software, quando projetos de código aberto são empregados como exemplo de projeto real a ser trabalhado pelos estudantes, resolvemos retornar aos trabalhos selecionados no processo do mapeamento sistemático e apurarmos tais questões. Sendo assim, realizamos uma síntese dos resultados apresentados por esses estudos. O processo de realização da síntese compreendeu as seguintes atividades: (i) restrição dos estudos, (ii) definição da estratégia para a síntese das evidências e (iii) a execução da síntese propriamente dita. Cada uma dessas atividades será detalhada a seguir.

5.2.1 Restrição dos estudos

Como o foco principal de muitos estudos identificados no mapeamento era somente expor o método de incorporação do OSP empregado, e poucos estudos expressavam algum rigor metodológico, optamos por não sermos muito restritivos com relação à qualidade. Desse modo, estabelecemos que o estudo, para ser selecionado para essa análise, deveria ao menos explicitar o método/instrumento utilizado para a coleta ou análise de dados, ou mesmo, apresentar algum relato dos estudantes. O nosso propósito era, com vistas a nossa procura por evidências, excluir estudos que reportassem resultados que fossem apenas reflexões de seus autores.

A Tabela 5.1 enumera os 23 estudos que atenderam ao critério estabelecido. Para cada estudo, levantamos o número de estudantes participantes, os métodos/instrumentos utilizados para a coleta ou análise de dados empregados, o número de opções utilizadas nas escalas ordinais, quando presentes nos questionários e, por fim, o número de estudantes que responderam ao(s) questionário(s), quando houver. A variação do número de opções das escalas ordinais utilizadas nos questionários dos estudos demonstra a heterogeneidade de apresentação das evidências. Como descreveremos mais adiante, esta informação será importante para a interpretação destas evidências.

Considerando as recomendações de Kitchenham e Charters (2007) de não incluir dados reportados por múltiplas publicações e de que somente a publicação mais completa seja utilizada para que a síntese não se torne tendenciosa, faremos algumas observações sobre os estudos selecionados.

Os trabalhos de Ellis et al. (2007), Ellis et al. (2007), Morelli e Lanerolle (2009), Hislop, Ellis e Morelli (2009), Ellis, Hislop e Morelli (2011) e Ellis et al. (2012) representam estudos realizados dentro do projeto HFOSS[1] (*Humanitarian Free and Open Source Software*), cujo objetivo é construir uma comunidade, envolvendo departamentos acadêmicos de computação, empresas de tecnologia da informação e instituições sem fins lucrativos, dedicada a construir e usar softwares de código aberto que atendam a necessidades humanitárias. Conforme investigamos, enquanto os três primeiros estudos apresentam coletas de dados distintas, o estudo de Ellis et al. (2012) engloba os dados quantitativos de His-

[1] http://hfoss.org/

Tabela 5.1 Artigos provenientes do mapeamento sistemático que apresentam evidências sobre o uso de OSP.

Estudo	Número de participantes	Método e/ou Instrumento	Número de opções	Quantidade de respostas
(ELLIS et al., 2007)	3	Relatos dos estudantes e instrutores	NA	NA
(ELLIS et al., 2007)	9	Questionário	5	5
(HISLOP; ELLIS; MORELLI, 2009)	30	Entrevista estruturada e questionário	5	27
(ELLIS; HISLOP; MORELLI, 2011)	19	Questionário	5	19
(ELLIS et al., 2012)	158 (ao longo de vários anos)	Questionário	5	133
(MORELLI; LANEROLLE, 2009)	13	Questionário 1 (questões fechadas) e Questionário 2 (questões abertas)	3	Q1 teve 8 respostas. Q2 teve 4 respostas.
(SOWE; STAMELOS; DELIGIANNIS, 2006)	13	Questionário e análise dos trabalhos executados	NE	10
(SOWE; STAMELOS, 2007)	13	Dois questionários (antes e depois) e análise dos trabalhos executados	2	11
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	268 (ao longo de vários anos)	Questionário	2	268
(MCCARTNEY; GOKHALE; SMITH, 2012)	36	Dois questionários (antes e depois). Análise temática das questões abertas.	NA	QA teve 34 respostas, QD teve 35 respostas.
(BUCHTA et al., 2006)	NE	Questionário e entrevista	5	NE
(PETRENKO et al., 2007)	NE	Questionário e entrevista	NE	NE
(RAJ; KAZEMIAN, 2006)	34	Questionário, avaliações dos trabalhos e observações do professor	5	22
(REKHA et al., 2009)	NE	Questionário	3	55
(CARRINGTON, 2003)	264	Análise dos trabalhos executados e questionário	5	149
(MARMORSTEIN, 2011)	NE	Questionário	4	NE
(XING, 2010)	NE	Questionário	5	NE
(KUSSMAUL, 2009)	NE	Relatos dos estudantes	NA	NA
(LUNDELL; PERSSON; LINGS, 2007)	12	Relatos dos estudantes	NA	NA
(CHEN et al., 2008)	> 200	Questionário com questões abertas	NA	NA
(QIAN; FU, 2008)	NE	Questionário	NE	NE
(KILAMO, 2010)	22	Questionário	5	15
(KROGSTIE, 2008)	1 equipe	Observações, análises de documentos e entrevistas.	NA	NA

(NA) representa que a informação não se aplica para o caso particular e (NE) representa que o estudo não especificou a informação.

lop, Ellis e Morelli (2009) e Ellis, Hislop e Morelli (2011). Por essa razão, no caso destes três últimos, tendo em vista os resultados quantitativos, utilizamos em nossa síntese de evidências apenas os resultados de Ellis et al. (2012). Como o estudo de Hislop, Ellis e Morelli (2009) além de aplicar o questionário realizou uma entrevista estruturada, estes resultados também foram considerados. Os resultados dos demais estudos Ellis et al. (2007), Ellis et al. (2007) e Morelli e Lanerolle (2009) foram utilizados normalmente. Convém ressaltar que procuramos isolar as evidências que diziam respeito somente ao uso de projetos de código aberto da questão humanitária e do processo particularmente aplicado no projeto HFOSS.

Situação semelhante ocorreu para os trabalhos de Sowe, Stamelos e Deligiannis (2006), Sowe e Stamelos (2007) e Papadopoulos, Stamelos e Meiszner (2012), que foram realizados pelo mesmo grupo de pesquisa. Este grupo introduziu o uso de projetos de código aberto em uma disciplina introdutória de engenharia de software. Porém, enquanto que o estudo de Sowe e Stamelos (2007) englobava os resultados de Sowe, Stamelos e Deligiannis (2006), onde os estudantes assumiam apenas o papel de testadores, o estudo de Papadopoulos, Stamelos e Meiszner (2012) envolvia dados totalmente distintos, onde os estudantes podiam escolher entre testar o software, desenvolver uma nova funcionalidade ou documentá-lo. Utilizamos em nossa análise apenas os resultados de Sowe e Stamelos (2007) e Papadopoulos, Stamelos e Meiszner (2012).

Os estudos de Buchta et al. (2006) e Petrenko et al. (2007) buscaram a utilização de projetos de código aberto para a aprendizagem de evolução de software. Contudo, dado a ênfase ao processo de mudança incremental utilizado, poucos resultados puderam ser aproveitados para nossa síntese. Sendo assim, apesar do estudo de Petrenko et al. (2007) estender o estudo de Buchta et al. (2006), conseguimos aproveitar apenas pouquíssimas evidências provenientes deste último.

Finalmente, decidimos excluir os trabalhos de Kilamo (2010) e Krogstie (2008) da síntese dos resultados. O primeiro porque os resultados reportados referiam-se somente à avaliação do jogo criado para a aprendizagem sobre o funcionamento de comunidades de projetos de código aberto (KILAMO, 2010). O segundo porque, apesar dos resultados referirem-se à interação de um dos membros da equipe formada, no papel de interlocutor, com a comunidade do projeto de código aberto, não encontramos nenhuma informação relevante para a análise que estávamos executando (KROGSTIE, 2008). Neste estudo, os estudantes precisavam entender e usar componentes de código aberto no desenvolvimento de seus próprios projetos.

Ao final, chegamos a 18 artigos cujas evidências seriam sintetizadas. A Tabela 5.2 lista estes artigos. Além da identificação de cada estudo, utilizamos as demais colunas para contextualizarmos os resultados reportados em relação à proposta didática aplicada em cada estudo. Desse modo, transcrevemos dados já reportados no mapeamento, que são: (i) objetivo da abordagem (aprender conceitos e princípios da engenharia de software (ES), especificamente sobre projetos de código aberto (OSP) ou ambos); (ii) área de conhecimento da engenharia de software particularmente abordada, ou sobre engenharia de software em geral; (iii) forma de incorporação da abordagem no currículo; (iv) nível de controle do professor sobre o projeto e atividades que foram executadas pelos estudantes e, por fim, (v) como o OSP a ser trabalhado foi selecionado.

Tabela 5.2 Artigos cujas evidências foram sintetizadas.

Estudo	Objetivo	Área da ES	Incorporação no Currículo	Controle	Escolha do OSP
(ELLIS et al., 2007)	Ambos	Geral	Atividade extra	Definição interna e aprovação externa	Projeto pré-definido
(ELLIS et al., 2007)	Ambos	Geral	Disciplina	Definição interna e aprovação externa	Projeto pré-definido
(HISLOP; ELLIS; MORELLI, 2009)	ES	Geral	Disciplina e atividade extra	Não especificado	Não especificado
(ELLIS et al., 2012)	ES	Geral	Disciplina e atividade extra	Não especificado	Não especificado
(MORELLI; LANEROLLE, 2009)	OSP	Geral	Disciplina	Definição interna e aprovação externa	Projeto pré-definido
(SOWE; STAMELOS, 2007)	ES	Teste de software	Disciplina	Nenhum controle	Livre escolha
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	ES	Geral	Disciplina	Nenhum controle	Livre escolha
(MCCARTNEY; GOKHALE; SMITH, 2012)	ES	Evolução e manutenção de software	Disciplina	Controle interno	Livre escolha
(BUCHTA et al., 2006)	ES	Evolução e manutenção de software	Disciplina	Controle interno	Projeto pré-definido
(RAJ; KAZEMIAN, 2006)	ES	Desenvolvimento e construção de software	Disciplina	Definição interna e aprovação externa	Seleção da lista fornecida
(REKHA et al., 2009)	ES	Desenvolvimento e construção de software	Disciplina, atividade extra e projeto de final de curso	Não especificado	Não especificado
(CARRINGTON, 2003)	Ambos	Design, arquitetura e teste de software	Disciplina	Controle interno	Seleção da lista fornecida
(MARMORSTEIN, 2011)	ES	Geral	Disciplina	Nenhum controle	Livre escolha
(XING, 2010)	ES	Geral	Disciplina	Controle interno	Seleção da lista fornecida
(KUSSMAUL, 2009)	ES	Geral	Disciplina e projeto de final de curso	Controle interno	Seleção da lista fornecida
(LUNDELL; PERSSON; LINGS, 2007)	OSP	Geral	Disciplina	Nenhum controle	Livre escolha
(CHEN et al., 2008)	Ambos	Geral	Atividade extra	Controle total	Projeto pré-definido
(QIAN; FU, 2008)	ES	Processo de software, design e arquitetura	Disciplina	Controle interno	Livre escolha

Tabela 5.3 Escala estabelecida para interpretação das evidências.

Escala	Código	Interpretação
Confirma	C	A evidência comprova a ocorrência da categoria.
Confirma Parcialmente	CP	A evidência comprova parcialmente a ocorrência da categoria.
Neutra	N	A evidência não comprova a ocorrência da categoria, mas também não garante que ela não ocorreu.
Não confirma	NC	A evidência não comprova a ocorrência da categoria.

5.2.2 Definição da estratégia para síntese das evidências

No decorrer da releitura dos artigos selecionados, verificamos que as evidências reportadas encontravam-se tanto no formato de relatos qualitativos quanto no de estatísticas descritivas, estas últimas no formato de percentagens, médias ou testes estatísticos. Tendo em vista nosso intuito de esmiuçar as consequências da aplicação da abordagem de uso de OSP para a aprendizagem, qualquer achado seria relevante. Portanto, resolvemos traçar uma estratégia para nossa síntese voltada para entender e descrever quais seriam essas consequências. Achamos que o processo indutivo-dedutivo (MERRIAM, 2009) aplicado em nossos estudos de caso também seria adequado nesse caso. Além de que, percebemos que esse processo é semelhante à análise temática, uma das estratégias sugeridas para a síntese de evidências provenientes de múltiplas publicações (DIXON-WOODS et al., 2005) (CRUZES; DYBÅ, 2011), quando a análise é direcionada pelos dados, isto é, os temas são identificados a partir dos dados (*data driven*) (DIXON-WOODS et al., 2005).

Seguindo o processo estabelecido, categorizamos as evidências reportadas, estudo por estudo, e as agrupamos em temas de modo que, a cada nova evidência de outro artigo sendo analisado, as categorias e temas eram ajustados. Os temas e categorias encontrados serão descritos na próxima subseção.

Julgamos relevante tentar prover maior objetividade à síntese, dado o número de artigos envolvidos e as múltiplas formas de apresentação das evidências, mencionadas anteriormente. Assim, decidimos traçar uma escala ordinal para a intepretação dessas evidências. A Tabela 5.3 descreve a escala estabelecida. Cabe ressaltar que mesmo com a escala, a interpretação ainda representa alguma subjetividade, principalmente porque estamos tratando com percepções qualitativas. Ademais, os dados quantitativos também influenciaram o surgimento dos níveis intermediários, como explicaremos a seguir.

Particularmente para a interpretação dos dados quantitativos, também devido as diferentes métricas empregadas, precisamos conceber algumas heurísticas. Os objetivos dessas heurísticas eram prover um método consistente para a nossa interpretação, propiciar transparência e permitir a repetibilidade do processo.

Compomos as heurísticas através de três elementos. O primeiro representava a métrica empregada pelo estudo ao reportar a evidência. O segundo correspondia ao critério a ser avaliado. Finalmente, o terceiro representava a interpretação para a evidência. Como na Tabela 5.3, definimos a escala para a interpretação das evidências e as métricas empregadas dependem dos estudos onde os resultados estão reportados, o desafio crucial tornou-se definir os critérios a serem utilizados.

Como podemos visualizar na Tabela 5.1, o principal instrumento utilizado para a coleta de dados foi o questionário. Porém, entre os estudos, as questões usavam escalas

ordinais diferentes para capturar a percepção dos estudantes sobre o tópico questionado. As escalas com número ímpar de opções de reposta apresentavam um ponto neutro, ou seja, o estudante poderia optar, por exemplo, por não concordar, nem discordar com determinada afirmação. Por outro lado, nas escalas com o número par, o estudante tinha sempre que escolher entre posições positivas ou negativas, isto é, ou ele concordava ou discordava com determinada afirmação. Este fato, bem como a quantidade de participantes nas pesquisas ser geralmente pequena (turmas pequenas e/ou participação voluntária) e a possibilidade do comportamento aquiescente do estudante, discutida em 4.5, influenciaram a definição destes critérios.

Devido ao possível comportamento aquiescente dos estudantes, decidimos ser exigentes na interpretação das evidências. Portanto, para considerarmos que a evidência 'confirmava' a ocorrência de determinada categoria (ver Tabela 5.3), era preciso que o resultado representasse, de fato, a percepção de uma maioria qualificada dos estudantes.

Para as escalas com número de opções de resposta par, nas quais os estudantes não possuem uma opção de escolha neutra, criamos uma faixa intermediária que representasse que a evidência demonstrava alguma influência ou alteração, mas que não era percebida pela maioria absoluta dos estudantes. Tal faixa corresponde ao nível de interpretação 'confirma parcialmente' na Tabela 5.3. Quando percentagens eram utilizadas para expressar esses resultados, delimitamos tal faixa entre 60 e 80%. Isso quer dizer que, para escalas com número de opções de resposta par, a maioria absoluta qualificada corresponderia a valores iguais ou superiores a 80% (a evidência confirmaria a ocorrência da categoria), a maioria absoluta simples com valores entre 60 (inclusive) e 80% (a evidência confirmaria apenas parcialmente a ocorrência da categoria), e valores abaixo de 60% como se os participantes não reconhecessem a influência ou efeito da abordagem na questão sendo avaliada (a evidência não confirmaria a ocorrência da categoria). Utilizamos o valor de 80% para limitar a interpretação para a maioria absoluta qualificada, uma vez que valores superiores dificilmente são obtidos, quando existe um pequeno número de participantes, como dito, situação comum em pesquisas que envolvam estudantes. Esta heurística bem como as demais encontram-se resumidas na Tabela 5.4.

Para ilustrar a interpretação de quando o número de opções de resposta for par, extraímos um exemplo de evidência retirada do artigo de Marmorstein (2011). Para a afirmação: "Este projeto ensinou-me alguma coisa sobre manter e depurar software" (tradução nossa), 30% dos estudantes concordaram plenamente, 40% concordaram, 30% discordaram e nenhum estudante discordou plenamente. Com 70% de concordância (lado positivo), estando o resultado entre 60 e 80%, utilizando a heurística estabelecida, a interpretação para essa evidência é que a aprendizagem foi apenas parcialmente confirmada.

Para as escalas ordinais com número de opções de resposta ímpar, apesar dos estudantes possuírem a opção neutra como possibilidade de escolha, continuamos sendo exigentes na consideração da maioria qualificada. Desta forma, o lado positivo não bastava ser superior à opção neutra e ao lado negativo, ele precisava representar ao menos 60% das escolhas dos estudantes. Por outro lado, para considerarmos que a evidência não confirmava a questão sendo avaliada, as escolhas negativas também tinham que ser iguais ou superiores a 60%. Sendo assim, quando houvesse uma distribuição equivalente entre os lados positivo, negativo e a opção neutra, consideramos que os estudantes demonstra-

ram um sentimento de 'neutralidade' quanto à influência ou efeito sobre a questão sendo avaliada. Ver resumo da heurística na Tabela 5.4.

Para ilustrar a interpretação de quando o número de opções de resposta for ímpar, extraímos um exemplo de evidência retirada do artigo de Carrington (2003). Para a afirmação: "A minha ferramenta Java[2] é fácil de compreender" (tradução nossa), 7% dos estudantes concordaram plenamente, 30% concordaram, 32% optaram pela opção neutra, 22% discordaram 9% discordaram plenamente. Com 37% de concordância (lado positivo) e 31% de discordância (lado negativo), apesar do maior número de concordâncias, este valor ainda foi inferior a 60%, sendo assim, segundo a heurística definida, a evidência representa uma situação de neutralidade.

Convém destacar que utilizamos termos distintos para as faixas intermediárias, 'confirma parcialmente' e ' neutra' na escala ordinal estabelecida para interpretação das evidências (Tabela 5.3) exatamente por representarem percepções diferentes. Não podemos dizer, por exemplo, que quando as escolhas positivas dos estudantes estão entre 60 e 80%, que os estudantes tiveram uma concepção neutra quanto à questão sendo avaliada. Enquanto que, quando nem as escolhas positivas nem as escolhas negativas alcançaram 60%, também não podemos afirmar que os estudantes perceberam uma influência de concordância ou discordância parcial sobre a questão sendo avaliada.

Finalmente, alguns estudos apresentaram seus resultados no formato de médias. Coincidentemente, todos os estudos nesta situação utilizaram escalas ordinais de cinco pontos. Com exceção do estudo de Carrington (2003), que utilizou a tática inversa, os demais calcularam as médias considerando o valor negativo mais baixo (por exemplo: 'discordo plenamente') como um, e o valor positivo mais alto (por exemplo: 'concordo plenamente') como cinco. Nestes casos, estabelecemos a faixa intermediária de neutralidade para os valores das médias entre três e três e meio (ver Tabela 5.4). Para o estudo de Carrington (2003), para não gerar confusão na interpretação das evidências, ao invés de utilizarmos a média reportada, preferimos extrair os percentuais de escolha dos estudantes para cada opção de resposta dos gráficos disponibilizados.

Concluímos que toda a estratégia definida, reunindo características interpretativas (identificação das categorias e temas a partir dos dados, julgamento dos relatos qualitativos e significação dos resultados quantitativos), integrativas (agregar os resultados por categoria e tema identificado) e descritivas (identificação dos estudos, evidência e forma que a evidência foi apresentada) atendeu ao propósito estabelecido para a investigação, mencionado previamente. Na próxima seção, apresentaremos a síntese das evidências reportadas nos estudos selecionados.

5.2.3 Síntese das evidências

Ao realizarmos a síntese, agregamos as categorias identificadas a partir das evidências reportadas em seis temas: (i) aprendizagem proporcionada; (ii) associação teoria e prática; (iii) percepções sobre o uso do projeto de código aberto como recurso didático; (iv) dificuldades provenientes da aplicação da abordagem; (v) motivação e (vi) satisfação.

A seguir apresentamos as categorias encontradas de acordo com esses temas, seguindo

[2] "A minha ferramenta Java" refere-se ao software OSP utilizado pelo estudante.

Tabela 5.4 Heurísticas para interpretação dos resultados quantitativos.

Medida quantitativa	**Critério**	**Interpretação para a evidência**
Média (para questões com número de opções de resposta ímpar)	Maior ou igual a 3,5	Confirma
	Entre 3,0 (inclusive) e 3,5	Neutro
	Menor que 3,0	Não confirma
Percentagem para duas opções de resposta: sim ou não	Respostas 'Sim' maior ou igual a 80%	Confirma
	Respostas 'Sim' entre 60% (inclusive) e 80%	Confirma parcialmente
	Respostas 'Sim' menor que 60%	Não confirma
Percentagem para escala ordinal com número ímpar de opções de resposta: lado positivo, posição neutra e lado negativo.	Percentagem para lado positivo maior ou igual a 60%	Confirma
	Nem o lado positivo, nem o negativo tem mais que 60%	Neutro
	Percentagem para lado negativo maior ou igual a 60%	Não confirma
Percentagem para escala ordinal com número par de opções de resposta: apenas lado positivo e lado negativo.	Percentagem para o lado positivo maior ou igual a 80%	Confirma
	Percentagem para o lado positivo entre 60% (inclusive) e 80%	Confirma parcialmente
	Percentagem para o lado positivo menor que 60%	Não confirma

um formato padrão. Tabelas agrupam as categorias, descrevendo para cada categoria, as evidências que originaram a sua definição, o artigo e o formato no qual cada evidência foi reportada, por fim, a intepretação para a evidência, segundo as heurísticas definidas (Tabela 5.4). Ao expormos as evidências e as heurísticas empregadas, provemos transparência à nossa investigação.

Uma observação importante é que, em alguns artigos, os resultados quantitativos estavam apresentados apenas em gráficos. Portanto, nestas situações, os valores retratados neste levantamento são valores aproximados extraídos da visualização desses gráficos.

Aprendizagem proporcionada

No que concerne à aprendizagem proporcionada, os estudos reportaram evidências desde a aprendizagem de modo geral até a experiência profissional propiciada. Em virtude dessa abrangência, detectamos categorias intermediárias que inclusive norteiam a apresentação das evidências sobre o tema, com uma tabela para cada. As categorias intermediárias capturadas foram: aprendizagem de modo geral (Tabela 5.5), aprendizagem conceitual (Tabela 5.6), desenvolvimento de habilidades técnicas (Tabela 5.7), desenvolvimento de habilidades gerais (Tabela 5.8), mudanças de atitudes (Tabela 5.9) e experiência profissional (Tabela 5.10). Cada tabela, seguindo o padrão estabelecido descrito no início desta subseção, resume a categorização das evidências e a sua interpretação quanto à aprendizagem proporcionada.

De modo geral as evidências retratadas na Tabela 5.5 comprovam que a abordagem proporciona alguma aprendizagem, particularmente considerando o escopo da engenharia

Tabela 5.5 Evidências reportadas sobre a aprendizagem de modo geral.

Categoria		
Artigo	Formato e evidência encontrada	Interpretação
Aprendizagem de modo geral		
(BUCHTA et al., 2006)	Estatística descritiva: *"How much did you learn in this course?"* Média igual a 3,8.	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"I learned something from this project."* 100% concordaram.	Confirma
Aprendizagem sobre engenharia de software de modo geral		
(ELLIS et al., 2007)	Relato qualitativo: *"In answer to the question about how much students felt they learned about software engineering, "a fair amount" was the common answer."*	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"The quality subthemes (reliability, modularity, evolvability and robustness), and the Activities subthemes (planning, lifecycle, requirements, testing) reflect a deeper and more formal understanding of software engineering. The increased specificity and use of more formal vocabulary suggest that the students have matured within the discipline."*	Confirma

de software.

Do ponto de vista conceitual, a abordagem propiciou 'conhecimento' / 'entendimento', dentre os níveis da taxonomia de Bloom (Tabela 3.2) ou 'familiaridade', dentre os níveis de aprendizagem propostos pelo CS2013 (Tabela 3.3), com relação: (i) às fases que compreendem o desenvolvimento de software; (ii) ao design de software; (iii) à construção de software; (iv) aos benefícios e desvantagens de projetos de código aberto; (v) ao impacto da complexidade do projeto nas abordagens usadas para desenvolver software; (vi) ao impacto do tamanho do projeto nas abordagens usadas para desenvolver software e, por fim, (vii) às questões de qualidade do software (ver Tabela 5.6).

Tabela 5.6: Evidências reportadas sobre a aprendizagem conceitual.

Categoria		
Artigo	Formato e evidência encontrada	Interpretação
Conhecer as fases que compreendem o desenvolvimento de um software real		
(ELLIS et al., 2007)	Relato qualitativo: *"Students indicated that the primary knowledge gained from the course ranged from (...) to software development process..."*	Confirma
(ELLIS et al., 2012)	Teste estatístico com resultado significativo: *"I can list the high-level phases that comprise a software project in a real-world environment."* *p-value* = 0,012	Confirma
Compreender sobre design de software		
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is helping me understand software design and construction."* 66% concordaram	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about designing software"* 93% concordaram	Confirma
(XING, 2010)	Estatística descritiva: *"Helping me understand software design and construction."* Média igual a 4,9	Confirma
Compreender sobre construção de software		
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is helping me understand software design and construction."* 66% concordaram	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about implementing software"* 86% concordaram	Confirma
(XING, 2010)	Estatística descritiva: *"Helping me understand software design and construction."* Média igual a 4,9	Confirma
Descrever benefícios e desvantagens de projetos de código aberto		

(ELLIS et al., 2012)	este estatístico com resultado significativo: *"I can describe the drawbacks and benefits of FOSS to society."* *p-value* = 0,029	Confirma
(ELLIS et al., 2012)	Teste estatístico com resultado significativo: *"I can describe the drawbacks and benefits of FOSS to business."* *p-value* = 0,045	Confirma
(MORELLI; LANE-ROLLE, 2009)	Relato qualitativo: *"I learned that [Open-source software development] is like helping each other to build software."*	Confirma
(KUSSMAUL, 2009)	Relato qualitativo: *"I got some good exposure to open source communities and the ways that they work. (. . .) the contrast between [two different] communities allowed me to see just how important that momentum is for other developers."*	Confirma
Descrever o impacto da complexidade do projeto nas abordagens usadas para desenvolver software		
(ELLIS et al., 2012)	Teste estatístico com resultado significativo: *"I can describe the impact of project complexity on the approaches used to develop software."* *p-value* = 0,042	Confirma
Descrever o impacto do tamanho do projeto nas abordagens usadas para desenvolver software		
(ELLIS et al., 2012)	Teste estatístico com resultado significativo: *"I can describe the impact of project size on the approaches used to develop software."* *p-value* = 0,012	Confirma
Identificar questões de qualidade do software		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to Identify software quality issues."* 91% concordaram	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"Under Software Quality they wrote of the specific qualities they expect in software: reliability, modularity, evolvability and robustness."*	Confirma

No que diz respeito à competência de 'saber fazer', correspondente ao nível de 'aplicação' da taxonomia de Bloom (Tabela 3.2) ou de 'uso', entre os níveis de aprendizagem propostos pelo CS2013 (Tabela 3.3), as evidências reportadas nas publicações demonstram o desenvolvimento das seguintes competências (Tabela 5.7): (i) elaborar diagramas; (ii) testar software; (iii) programar; (iv) executar engenharia reversa e (v) usar ferramentas.

As evidências ainda apontam alguma aprendizagem com relação às habilidades de 'depurar código' e 'modificar software de maior porte', enquanto que duas evidências não confirmam o desenvolvimento da habilidade de 'corrigir *bugs* do software'. Apesar de estas habilidades serem fortemente relacionadas, os resultados foram extraídos de estudos distintos. Apenas o estudo de Marmorstein (2011) associa a modificação do software à depuração do código, com confirmação parcial da aprendizagem. Isto nos leva a crer que a aprendizagem relacionada à modificação de software ocorrida está mais relacionada à adição ou modificação de funcionalidades do que à correção de erros. Entretanto, cabe salientar que houve ao menos uma evidência que confirma parcialmente o desenvolvimento desta última competência.

Tabela 5.7: Evidências reportadas sobre o desenvolvimento de habilidades técnicas.

Categoria		
Artigo	Formato e evidência encontrada	Interpretação
Elaborar diagramas		
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"How to make Use Case Diagrams, Sequence Diagrams and Class Diagrams."*	Confirma
Programar		

(REKHA et al., 2009)	Relato qualitativo: *"Participants mentioned that (. . .) Working with FOSS forums helped them to learn the art of programming."*	Confirma
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is helping me improve my programming skills."* 30% concordaram e 25% discordaram.	Neutro
(XING, 2010)	Estatística descritiva: *"Helping me improve my programming skills."* Média igual a 4,7	Confirma
Testar software		
(SOWE; STAMELOS, 2007)	Resultado das atividades realizadas: os autores demonstraram que os resultados foram satisfatórios.	Confirma
Depurar código		
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about maintaining and debugging software."* 70% concordaram	Confirma parcialmente
Corrigir bugs do software		
(SOWE; STAMELOS, 2007)	Resultado das atividades realizadas: os autores demonstraram que os resultados não foram suficientes.	Confirma parcialmente
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Can you fix the bugs that you found?"* 44% confirmaram	Não confirma
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Can you fix the bugs that other people found?"* 33% confirmaram	Não confirma
Executar engenharia reversa		
(MCCARTNEY; GOKHALE; SMITH, 2012)	Resultado das atividades realizadas: *"We observed that each team used tools to extract their project designs, and was able to add and demonstrate functionality to their project. This offers some support for the idea that they gained sufficient comprehension from reverse engineering."*	Confirma
Modificar software de maior porte		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to develop or modify modules of large software systems."* 95% concordaram	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about maintaining and debugging software."* 70% concordaram	Confirma parcialmente
(ELLIS et al., 2012)	Estatística descritiva: *"I am confident that I can maintain an HFOSS project."* Média igual a 3,4	Neutro
Usar ferramentas de modelagem de design de software		
(ELLIS et al., 2007)	Relato qualitativo: *"Students indicated that the primary knowledge gained from the course ranged from tools to software design. . . "*	Confirma
Usar ferramentas de desenvolvimento de software		
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about software development tools."* 100% concordaram	Confirma
Usar ferramentas		
(ELLIS et al., 2012)	Teste estatístico com resultado significativo: *"I can use all tools and techniques employed in my HFOSS project."* p-*value* = 0,018	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"One student answered the question "What are the main things that you learned in this course?" with: Use of software engineering tools."*	Confirma

Os estudos também investigaram o desenvolvimento de habilidades gerais (ver Tabela 5.8). As evidências confirmam o progresso em habilidades como: (i) auto-organizar o próprio trabalho; (ii) ser capaz de solucionar problemas; (iii) ser capaz de propor outras soluções e (iv) saber elaborar documentação útil para terceiros.

Também podemos dizer que houve o aperfeiçoamento das habilidades de 'entender código desenvolvido por terceiros' e 'entender documentação elaborada por terceiros',

apesar da ocorrência de uma percepção de confirmação parcial e neutralidade para cada habilidade, respectivamente.

Para a habilidade de 'compreender qual deve ser o comportamento ético do profissional de computação', obtivemos duas interpretações opostas, inclusive com evidências extraídas do mesmo artigo (ELLIS et al., 2012). Todavia, como existe pelo menos uma evidência positiva, podemos dizer que o desenvolvimento dessa competência foi parcial.

Quanto ao desenvolvimento da habilidade de 'interagir com a comunidade do projeto', as evidências mostram que a abordagem pode promover o seu desenvolvimento. Contudo, cabe uma ressalva quanto à participação nos fóruns de discussão. Talvez o pouco tempo de experiência com o projeto, possivelmente dentro de um único período acadêmico, não dê aos estudantes a confiança necessária para participarem de discussões.

No que concerne às categorias de 'trabalhar com profissionais externos' e 'trabalhar em equipe', estas estão estreitamente ligadas, principalmente considerando projetos de código aberto. Porém, a depender da forma de aplicação da abordagem dentro do curso, o professor pode determinar que os estudantes trabalhem individualmente ou em equipe em um mesmo OSP. Assim, procuramos distinguir, quando possível, as evidências que representam a equipe com a integração com profissionais externos ou não. Para compreender a categorização das evidências dos estudos de Ellis et al. (2007) e Chen et al. (2008) nesta perspectiva, precisamos acrescentar algumas informações de contexto. No estudo de Ellis et al. (2007), os estudantes interagem tanto com instituições com finalidade humanitária quanto com estudantes de outras universidades que também participam do projeto. Por isso categorizamos as evidências reportadas em 'trabalhar com profissionais externos'. Já no estudo de Chen et al. (2008), a comunidade do projeto é a própria universidade, e o OSP representa um dos exemplos identificados no mapeamento onde existe o controle total do projeto dentro da academia. Como, ao longo do texto, não detectamos a interação com desenvolvedores externos, categorizamos a evidência como 'trabalhar em equipe'. Voltando à interpretação dos resultados relatados, as evidências mostram que os estudantes aprenderam a trabalhar com outros estudantes, mas as experiências não foram tão positivas com relação a trabalhar com profissionais externos.

Tabela 5.8: Evidências reportadas sobre o desenvolvimento de habilidades gerais.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Auto-organizar o próprio trabalho		
(KUSSMAUL, 2009)	Relato qualitativo: *"All in all, I'm quite glad I took this course because (. . .) taught me useful organizational skills for completing any sort of project or assignment I may have."*	Confirma
Ser capaz de solucionar problemas		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Figured out approaches leading to successful outcomes despite initial problems with OSS."* 96% concordaram	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"(...)they took special care in getting the design right and solving unanticipated problems that arose."*	Confirma
Ser capaz de propor outras soluções		

(CARRINGTON, 2003)	Relato qualitativo: *"Some students bund it difficult to imagine that they might be able to extend or improve the work of other people: "It's hard for us to say that our design could possibly be better than the original programmers."*	Confirma
Entender código desenvolvido por terceiros		
(ELLIS et al., 2007)	Relato qualitativo: *"The Wesleyan team reported that they learned about how to understand the project by reading code and project documents."*	Confirma
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to read and analyze large programs written by others."* 100% concordaram	Confirma
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Did you understand the code that other people submitted on your FLOSS project?"* 74% confirmaram	Confirma parcialmente
Entender documentação elaborada por terceiros		
(ELLIS et al., 2007)	Relato qualitativo: *"The Wesleyan team reported that they learned about how to understand the project by reading code and project documents."*	Confirma
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to read external documentation."* 59% concordaram e nenhum discordou.	Neutro
Saber elaborar documentação útil para terceiros		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to write useful documentation for other professionals."* 73% concordaram e nenhum discordou.	Confirma
Compreender qual deve ser o comportamento ético do profissional de computação		
(ELLIS et al., 2012)	Estatística descritiva: *"I can identify when peers in an HFOSS project are behaving in an unprofessional manner."* Média igual a 3,8	Confirma
(ELLIS et al., 2012)	Teste estatístico com resultado significativo negativo: *"Participation in an HFOSS project has improved my understanding of how to behave like a computing professional."* *p-value* = 0,006	Não confirma
Interagir com a comunidade do projeto		
(REKHA et al., 2009)	Relato qualitativo: *"Participants mentioned that (. . .) Working with FOSS forums gave them the necessary communication skills to interact with industry experts."*	Confirma
(ELLIS et al., 2012)	Estatística descritiva: *"I can participate in an HFOSS development team's interactions."* Média igual a 3,9.	Confirma
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Did you exchange messages with the project developers?"* 63% confirmaram.	Confirma parcialmente
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Did you participate in discussions in project forums?"* 34% confirmaram.	Não confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about communicating with other developers."* 94% concordaram.	Confirma
Trabalhar com profissionais externos		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to Work with external software developers."* 19% concordaram e 19% discordaram	Neutro
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about working in a large team."* 59% discordaram	Não confirma
(ELLIS et al., 2012)	Teste estatístico com resultado significativo negativo: *"I have gained some confidence in collaborating with professionals from a variety of locations and cultures."* *p-value* = 0,002	Não confirma

(ELLIS et al., 2007)	Relato qualitativo: *"Students indicated that the primary knowledge gained from the course ranged from (. . .) to working in a team." "The Connecticut College students said that the most important things they learned were how to work in teams and . . . "*	Confirma
(MORELLI; LANE-ROLLE, 2009)	Relato qualitativo: *"I now have a better understanding of what it is like to work with and contribute to a team of people, even when I may never meet them in person."*	Confirma
Trabalhar em equipe		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Improved ability to Work in software development teams."* 86% concordaram	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"My partner and I worked well together."* 80% concordaram	Confirma
(CHEN et al., 2008)	Relato qualitativo: *"The best part about it were the people. (. . .) The diversity of the group brought together people with different perspectives and different skill sets into a greater whole."*	Confirma

Evidências também mostram que a abordagem de uso de OSP foi fundamental para a conscientização dos estudantes quanto à relevância de diversos aspectos da engenharia de software, essencial para a mudança de atitudes. Os estudantes reconheceram a importância da modelagem, da documentação, do código autodocumentado, da manutenção e evolução de software como parte importante do ciclo de vida do software; que existem melhores práticas a serem seguidas para a programação; tiveram uma percepção mais profissional de que seus trabalhos encaixam-se em um contexto maior e, por fim, da importância dos projetos de código aberto para a sociedade e, consequentemente, de aprender sobre projetos de código aberto. A Tabela 5.9 exibe as evidências reportadas pelos diversos estudos que geraram a constatação dessas categorias.

Tabela 5.9: Evidências reportadas sobre mudanças de atitudes.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Reconhecer a importância da modelagem		
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"the importance in software artifacts, how they are helpful for understanding the code and help with future implementations."*	Confirma
Conscientização sobre a importância da documentação		
(ELLIS et al., 2007)	Relato qualitativo: *"The Connecticut College students said that the most important things they learned were (. . .) and the importance of documentation."*	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"This project taught me something about the importance of good documentation."* 100% concordaram	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"It is important to write good documentation for all of the code that is produced. Not only did good documentation serve to refresh one's own memory, it was a necessary reference for all the software experts who would be using the code, the sort of people we don't want to disappoint."*	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"Documentation and organization of the Software Lifecycle are absolutely critical to successful software development."*	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"I should document my work and work in a methodical way so that other can understand my work."*	Confirma

(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"I really learned the importance of documentation, because if something is not well documented, then it is nearly impossible to add extensions to it."*	Confirma
Conscientização sobre a importância do código autodocumentado		
(ELLIS et al., 2007)	Relato qualitativo: *"Writing smart, self-documenting code is an unstated project requirement. Because we were coding for a broader and more demanding audience, the clever or shoddy solutions that ordinarily pass muster in a CS class are simply not welcome in the OS environment. In order for our contributions to be accepted, it was necessary that they possess the right functionality and achieve it the right way. Further, thrashing through dense or poorly documented preexisting code was a learning experience in itself. We were forced to see how the code worked, often times experimentally."*	Confirma
Conscientização sobre melhores práticas de programação		
(ELLIS et al., 2007)	Relato qualitativo: *"At the same time, we would be learning how not to write our own programs."*	Confirma
Conscientização da importância da manutenção e evolução do software		
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"Software maintenance and planning are the most important part of the development process."*	Confirma
(MCCARTNEY; GOKHALE; SMITH, 2012)	Relato qualitativo: *"One team asked advice when it was faced with two choices about how to implement a modification (...) The question revealed that they knew that it was critical to avoid destructuring the code."*	Confirma
Conscientização sobre a importância dos projetos de código aberto		
(MORELLI; LANE-ROLLE, 2009)	Relato qualitativo: *"Before I thought that [FOSS] was only relevant to computer scientists and no one else. But that is not true. The open source movement is much more relevant to the greater humanitarian good."*	Confirma
(MORELLI; LANE-ROLLE, 2009)	Relato qualitativo: *"Open-source software development allows for low- or no-cost highly customizable software products that can be used to support many causes who have limited financial means."*	Confirma
Conscientização da importância de aprender sobre projetos de código aberto		
(LUNDELL; PERSSON; LINGS, 2007)	Relato qualitativo: *"The open source phenomenon will surely stay for a long time to come and it is only positive to have gained a more direct understanding of how these work and function."*	Confirma
Conscientização de que seu trabalho encaixa-se em um contexto maior		
(MORELLI; LANE-ROLLE, 2009)	Relato qualitativo: *"It shows how my input to the open source community can have a greater impact on the open source community with the available source on the web."*	Confirma
(KUSSMAUL, 2009)	Relato qualitativo: *"Having to work with others on most projects was also something new to me ... I found it hard to keep a balance between needing to know what was going on and just assuming things would all end up fine – I often felt unsure about whether or not I was doing enough for the group, or whether the project would be completed ... I certainly learned that I need to keep a better focus on what everyone is doing."*	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"It is necessary to know where our contribution is needed, and what is needed of our contribution."*	Confirma

Como última categoria intermediária associada à aprendizagem, os artigos relatam que a abordagem de uso de OSP proporcionou alguma experiência profissional ao estudante (Tabela 5.10), seja pela 'vivência de um projeto real', seja por uma 'visão mais profissional' sobre o projeto de software. Vários estudos também exibiram evidências que a abordagem propiciou a 'capacitação para o mercado'. Apenas um trabalho reportou

uma evidência negativa relacionada ao 'alto nível de experiência proporcionada sobre o assunto' (ELLIS et al., 2007), em contrapartida, o mesmo estudo apresentou evidências que confirmam a experiência de mundo real provida (categorizada em 'vivência de um projeto real') e a 'capacitação para o mercado'. Finalmente, evidências ainda demonstram que a abordagem contribuiu para o 'desenvolvimento da autoconfiança' dos estudantes.

Tabela 5.10: Evidências reportadas sobre a experiência profissional proporcionada.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Vivência de um projeto real		
(ELLIS et al., 2007)	Relato qualitativo: *"The benefits that students mention most are those related to real-world experience and..."*	Confirma
(RAJ; KAZEMIAN, 2006)	Relato qualitativo: *"Many students stated that they appreciated the real-world nature of the OSS experience because they were painfully aware via coops and other job experiences that most programmers do not have the luxury to design software from scratch."*	Confirma
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is giving me real-world software experience."* 67% concordaram	Confirma
(XING, 2010)	Estatística descritiva: *"Giving me real-world software experience."* Média igual a 4,9	Confirma
Visão profissional		
(KUSSMAUL, 2009)	Relato qualitativo: *"This acted as a way to teach me how to work on a project for a customer. [We] were the contractors and the Librarians were our clients. We were consultants, providing progress reports and working on new acceptable features, trying to meet the clients needs. [We] were lucky to have a way to experience working with people who had a vested interested in our production."*	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"Working with the Sahana project, we needed to find our own place in a real world organizational network. This may mean changing deadlines, changing specifications, database updates -anything, really. Unlike an independent project, which is purely one-dimensional (receive the programming assignment and do it), an OSS project is multi-dimensional."*	Confirma
Alto nível de experiência proporcionada sobre o assunto		
(ELLIS et al., 2007)	Estatística descritiva: *"I have a high level of experience in the course subject matter."* Média igual a 2,5	Não confirma
Desenvolvimento da autoconfiança		
(REKHA et al., 2009)	Relato qualitativo: *"Participants mentioned that (...) They were not feeling insecure about job opportunities because they already had live project experiences."*	Confirma
(REKHA et al., 2009)	Estatística descritiva: *"Increase in confidence levels after FOSS usage."* Muito (45%), moderadamente (53%) e não estou certo (2%)	Neutro
(ELLIS et al., 2012)	Estatística descritiva: *"I am comfortable that I could participate in the planning and development of a real-world software project."* Média igual a 3,6	Confirma
(ELLIS et al., 2012)	Estatística descritiva: *"I am confident that I can maintain an HFOSS project."* Média igual a 3,4	Neutro
Capacitação para o Mercado		
(ELLIS et al., 2007)	Estatística descritiva: *"The subject matter of this course is highly relevant to my future career plans."* Média igual a 3,75	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"The benefits that students mention most are those (...) and marketability."*	Confirma
(RAJ; KAZEMIAN, 2006)	Relato qualitativo: *"(...) several students stated that their marketability had improved due to the external recognition of their software development skills."*	Confirma

(REKHA et al., 2009)	Relato qualitativo: *" (. . .) were convinced that it would help them improve their placement prospects."*	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva: *"The project gave me experience that will help me find a job."* 64% concordaram	Confirma parcialmente
(KUSSMAUL, 2009)	Relato qualitativo: *"All in all, I'm quite glad I took this course because it not only taught me about how to actually work as a programmer, but taught me useful organizational skills for completing any sort of project or assignment I may have."*	Confirma
(CHEN et al., 2008)	Relato qualitativo: *"The experience prepared me for a career in software development in ways that my CS classes never could."*	Confirma

Associação Teoria e Prática

Sob a perspectiva voltada para o processo de aprendizagem, evidências mostram que 'o uso de projetos reais permite colocar em prática toda a teoria estudada' e ainda mais importante, com 'a prática em projetos reais é possível enxergar as razões de aplicar a teoria', de maneira que o estudante vivencia a necessidade de aplicar determinado método, ou seja, a utilização do método não provém apenas de recomendação teórica. A Tabela 5.11 exibe as evidências reportadas que geraram a descoberta destas categorias.

Tabela 5.11: Evidências reportadas sobre a associação teoria e prática.

Categoria		
Artigo	Formato e evidência encontrada	Interpretação
O uso de projetos reais permite colocar em prática toda a teoria estudada		
(CHEN et al., 2008)	Relato qualitativo: *"It pulled together all of the theoretical concepts from the various CS classes in providing my first practical application of my degree. Everything I learned in class was also present in GamesCrafters."*	Confirma
Com a prática em projetos reais é possível enxergar as razões de aplicar a teoria		
(HISLOP; ELLIS; MORELLI, 2009)	Relato qualitativo: *"The instructor thought that students got real-world experience with development tools. 'Students can see the tools in use and have a reason to use them as opposed to pure discussion.' "*	Confirma

O uso do projeto de código aberto como recurso didático

Alguns estudos reportaram evidências sobre benefícios proporcionados pela utilização de projetos de código aberto como recurso didático. Participantes destes estudos reconheceram o uso do OSP como uma 'experiência de mundo real', que pela sua característica de 'liberdade', permite que o seu código seja utilizado e modificado da maneira que for necessário e, por fim, por representar um 'software existente', permite que os estudantes trabalhem de modo semelhante ao meio profissional, onde nem sempre os programadores trabalham com projetos desde o seu início. A Tabela 5.12 expõe as evidências reportadas que permitiram a identificação desses benefícios.

Dificuldades provenientes da aplicação da abordagem

Se, por um lado, a abordagem de uso de OSP proporciona os benefícios que acabamos de relatar, por outro lado ela acarreta vários tipos de obstáculos. Os estudos relataram evidências sobre a ocorrência de diversas dificuldades (Tabela 5.13), entre as quais: (i) menos orientação por parte do professor, uma vez que existe menos conhecimento sobre o OSP; (ii) demanda dos estudantes por respostas rápidas do professor; (iii) exigência de

Tabela 5.12 Evidências reportadas sobre os benefícios proporcionados pelo uso de projetos de código aberto.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Provê uma experiência de mundo real		
(ELLIS et al., 2007)	Relato qualitativo: *"The benefits that students mention most are those related to real-world experience."*	Confirma
(RAJ; KAZEMIAN, 2006)	Relato qualitativo: *"Many students stated that they appreciated the real-world nature of the OSS experience . . . "*	Confirma
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is giving me real-world software experience."* 67% concordaram.	Confirma
(XING, 2010)	Estatística descritiva: *"Giving me real-world software experience."* Média igual a 4.9	Confirma
Liberdade de modificar o software		
(REKHA et al., 2009)	Relato qualitativo: *"Participants mentioned that (. . .) The software is free and they could modify it the way they wanted."*	Confirma
Permite trabalhar com projeto existente		
(ELLIS et al., 2007)	Relato qualitativo: *"Typical programming assignments begin with a blank screen. But in this context, there was a huge repository of pre-existing code, so writing a new page almost inevitably involved modification of an old one."*	Confirma

mais tempo e esforço para a adoção de práticas da indústria; (iv) procrastinação no início das atividades no projeto por parte dos estudantes; (v) necessidade de configuração do ambiente e (vi) tecnologia utilizada no projeto.

Contudo, conforme evidenciado por Carrington (2003), o OSP selecionado influencia diretamente as dificuldades a serem enfrentadas. Possivelmente esta seria uma das principais explicações para a existência de evidências que representam resultados diversos sobre a 'dificuldade de compreensão do projeto'; que confirmam ou confirmam apenas parcialmente as dificuldades na 'execução das atividades' e da 'documentação ser incompleta', ou ainda de estudos distintos apresentarem resultados opostos com relação ao 'suporte insuficiente dado pela comunidade do projeto' e à 'limitação de tempo'.

Tabela 5.13: Evidências sobre as dificuldades encontradas no uso de projetos de código aberto.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Dificuldade de compreender o projeto		
(ELLIS et al., 2007)	Relato qualitativo: *"The top roadblocks to student learning identified by students included the steep learning curve for some aspect of the project. . . "*	Confirma
(REKHA et al., 2009)	Relato qualitativo: *"Difficulty in understanding code written by experts with limited background knowledge acquired in the university."*	Confirma
(CARRINGTON, 2003)	Estatística descritiva (pergunta inversa): *"My Java tool is easy to understand."* 37% concordaram e 31% discordaram.	Neutro
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"How many days passed before you started finding bugs/ writing code/ writing the SRS document?"* Em média, 15 dias	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva (pergunta inversa): *"The code base I contributed to was easy to understand."* 59% discordaram	Confirma
(XING, 2010)	Estatística descritiva (pergunta inversa): *"Easy to understand."* Média igual a 4,4	Não confirma

(MORELLI; LANE-ROLLE, 2009)	Relato quantitativo: *"Challenging?"* Muito (12,5%) e não muito (12,5%)	Neutro
Documentação incompleta		
(ELLIS et al., 2007)	Relato qualitativo: *"The top roadblocks to student learning identified by students included (. . .) incomplete library documentation. . . "*	Confirma
(MARMORSTEIN, 2011)	Relato qualitativo: *"The results of the survey outline several additional limitations of the project. In particular, they highlight three major areas of concern that negatively impacted successful completion of the project: (. . .), and poor documentation of the upstream project."*	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva (pergunta inversa): *"The API documentation for the project I contributed to was helpful."* 76% concordaram	Confirma Parcialmente
Configuração do ambiente		
(REKHA et al., 2009)	Relato qualitativo: *"FOSS tools sometimes do not come as one package and a lot of dependency software had to be installed separately.This required a lot of effort."*	Confirma
Tecnologia utilizada no projeto		
(ELLIS et al., 2007)	Relato qualitativo: *"The top roadblocks to student learning identified by students included (. . .) technology blocks (. . .), lack of JavaScript skills, and difficult-to-trace bugs."*	Confirma
O OSP selecionado influencia as dificuldades a serem enfrentadas		
(CARRINGTON, 2003)	Relato qualitativo: *"Student feedback at the end of the course indicated that they felt that the initial choice of a tool had unduly influenced their results: "The tools should be equally difficult" and "It is obvious that some tools are much harder to understand and do the assignments on. I feel this disadvantages students who have to do a larger, more complicated tool (i.e. NetBeans)."*	Confirma
Suporte insuficiente dado pela comunidade do projeto		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Did you receive feedback through project forums/ emails/ private messages?"* 54% confirmaram	Confirma
(MARMORSTEIN, 2011)	Estatística descritiva (afirmação inversa): *"The developers of the project I contributed to were helpful"* 53% discordaram	Confirma
(LUNDELL; PERSSON; LINGS, 2007)	Relato qualitativo: *"I don't know if I should say that I was surprised by the kind treatment I got, but I was definitely surprised how glad they were to let me join the development team. The kind of welcome I got definitely made me more interested in joining open source projects if I find that something needs fixing."*	Não confirma
(SOWE; STAMELOS, 2007)	Estatística descritiva (afirmação inversa): *"Towards the end of the program, ten out of eleven students reported that their projects' communities are very friendly and responsive."* 91% concordaram.	Não confirma
Existe menos conhecimento sobre o OSP internamente e, consequentemente, menos orientação por parte do professor		
(REKHA et al., 2009)	Relato qualitativo: *"There was very less in-house expertise and guidance for FOSS."*	Confirma
Demanda dos estudantes por respostas rápidas do professor		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Relato qualitativo: *"they asked for faster response times from the instructor and the assistants in the learning environment."*	Confirma
Adotar práticas da indústria exige mais tempo e esforço		
(REKHA et al., 2009)	Relato qualitativo: *"Adopting industry practices for coding and documentation took a lot of time and effort."*	Confirma
Procrastinação no início das atividades no projeto por parte dos estudantes		
(MARMORSTEIN, 2011)	Estatística descritiva (afirmação inversa): *"I started early on this project."* 55% concordaram	Confirma

Limitação de tempo		
(MARMORSTEIN, 2011)	Relato qualitativo: *"The results of the survey outline several additional limitations of the project. In particular, they highlight three major areas of concern that negatively impacted successful completion of the project: project time frame..."*	Confirma
(SOWE; STAMELOS, 2007)	Estatística descritiva (afirmação inversa): *"Eight out of eleven students reported that they had enough time to work on their projects."* 73% afirmaram que o tempo foi suficiente.	Não confirma
Dificuldade na execução da atividade solicitada		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Was it easy for you to find a bug/ develop a functionality/ analyze a requirement?"* 65% confirmaram	Confirma Parcialmente
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Was it easy for you to properly report a bug/ submit your code/ submit the document?"* 76% confirmaram	Confirma Parcialmente
(SOWE; STAMELOS, 2007)	Relato qualitativo: *"Most of the students found it difficult to find bugs in their software."*	Confirma

Alguns estudos também reportaram evidências quanto à dificuldade na seleção dos projetos a serem utilizados, quando os próprios estudantes eram responsáveis por essa seleção. A Tabela 5.14 detalha as dificuldades provenientes desse processo e os estudos que as reportaram. Evidências mostram que houve 'dificuldade em encontrar o OSP que fosse adequado' às atividades que precisavam ser realizadas, ou mesmo, que 'fossem do interesse dos estudantes'. Em um estudo, os estudantes ainda reivindicaram mais suporte na seleção do OSP apropriado (PAPADOPOULOS; STAMELOS; MEISZNER, 2012).

Motivação

Encontramos algumas evidências reportadas nos artigos que o 'uso de projetos reais' e a possibilidade de 'visibilidade externa do que era produzido' estimularam os estudantes na execução de suas atividades. A 'dedicação ao projeto' representa outra evidência que confirma esta motivação. A Tabela 5.15 resume as evidências e os estudos onde foram encontradas.

Satisfação

Satisfação corresponde ao último tema que detectamos a partir das evidências reportadas pelo estudos selecionados. As categorias identificadas demonstram o contentamento dos estudantes ou representam fatores que colaboraram para este contentamento (ver Tabela 5.16).

As evidências confirmaram a satisfação dos estudantes com: (i) o curso de modo geral; (ii) a aprendizagem proporcionada; (iii) a abordagem de aprendizagem da teoria combinada com a prática do assunto; (iv) a utilidade da incorporação de projetos de código aberto nas atividades das disciplinas e (v) o suporte dado pelo professor. Apenas uma neutralidade e uma confirmação parcial foram reportadas para a satisfação em 'utilizar o OSP para aprender sobre o assunto' e com 'o projeto de código aberto utilizado', respectivamente.

Por outro lado, como fatores que colaboraram para a satisfação dos estudantes, as evidências apontam: (i) trabalhar com projetos reais; (ii) trabalhar com projeto que terá

Tabela 5.14 Evidências reportadas sobre as dificuldades de seleção do OSP por parte dos estudantes.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Dificuldade em encontrar o OSP adequado		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva (pergunda inversa): *"Was it easy for you to find an appropriate FLOSS project?"* 35% confirmaram	Confirma
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"How many projects did you try, before selecting the final one?"* Em media, 6,76 projetos.	Confirma
(MARMORSTEIN, 2011)	Relato qualitativo: *"Groups who switched projects in the middle of the semester..."*	Confirma
(SOWE; STAMELOS, 2007)	Relato qualitativo: *"In both surveys the students expressed that it was not easy to find a project to participate in."*	Confirma
Dificuldade em encontrar um OSP de meu interesse		
(SOWE; STAMELOS, 2007)	Estatística descritiva: *"Even a higher number (N=9) in survey 2 reported that it was not easy to get a project which interested them."* 82% confirmaram	Confirma
É preciso mais suporte para achar o OSP apropriado		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Relato qualitativo: *"Students asked for more support during the selection process. Although, support on how to find an appropriate project is already available, students felt that more guidance is necessary in order to identify a good project for the assignment."*	Confirma

Tabela 5.15 Evidências reportadas sobre motivação.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Uso de projetos reais		
(ELLIS et al., 2007)	Relato qualitativo: *"One student indicated an interest in working on a real-world project: 'I was excited to work on something that had the potential for real-world use, rather than just class exercises.' "*	Confirma
Visibilidade externa do que for produzido		
(REKHA et al., 2009)	Relato qualitativo: *"They felt constantly motivated because whatever they contributed could be seen by the whole world..."*	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"Because it was going to be published with their names attached, they took enormous pride in their code and documentation."*	Confirma
Dedicação ao projeto		
(MARMORSTEIN, 2011)	Estatística descritiva: *"I did my best on this project."* 82% concordaram	Confirma

continuidade; (iii) trabalhar com projeto de código aberto e (iv) contribuir com o mundo open source e a sociedade.

Tabela 5.16: Evidências reportadas sobre satisfação.

Categoria Artigo	Formato e evidência encontrada	Interpretação
Trabalhar com projetos reais		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Relato qualitativo: *"Finally, student praised the fact that they were able to work on real projects..."*	Confirma
(LUNDELL; PERSSON; LINGS, 2007)	Relato qualitativo: *"I think all-in-all it was a good experience to make a real contribution to a real open source project. "*	Confirma
Trabalhar com projeto que terá continuidade		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Liked contributing to a project that did not end with the academic term. "* 86% concordaram	Confirma
(REKHA et al., 2009)	Relato qualitativo: *"They felt constantly motivated because whatever they contributed (...) was available for further updation. "*	Confirma
Trabalhar com projeto de código aberto		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Relato qualitativo: *"Finally, student praised the fact that they were able to (...) experience the development of open source software first hand. "*	Confirma
(MORELLI; LANEROLLE, 2009)	Relato qualitativo: *"I really loved delving into a world I'd barely seen before: Open Source. "*	Confirma
(ELLIS et al., 2007)	Relato qualitativo: *"Like most CS students, we had never been contributors to OSS, and working on Sahana was something of a revelation. "*	Confirma
Com o curso de modo geral		
(BUCHTA et al., 2006)	Estatística descritiva: *"Question one asks the students to rate the course as a whole with five being 'Excellent' and one being 'Very Poor'. '* Média igual a 3,7	Confirma
(MORELLI; LANEROLLE, 2009)	Estatística descritiva: *"Class worthwhile? "* 75% concordaram	Confirma
Com a aprendizagem proporcionada		
(ELLIS et al., 2007)	Estatística descritiva: *"I am very satisfied with my learning experience in this course. "* Média igual a4,0	Confirma
(QIAN; FU, 2008)	Relato qualitativo: *"From the anonymous survey we conducted in the second last week of the class, students were generally pleased with the learning outcome. "*	Confirma
Com a abordagem de aprendizagem da teoria combinada com a prática		
(ELLIS et al., 2007)	Estatística descritiva: *"I like the mix of theoretical learning combined with hands-on application of the subject matter. "* Média igual a3,5	Confirma
(XING, 2010)	Estatística descritiva: *"Is effective to learn this subject matter. "* Média igual a 4,8	Confirma
Em utilizar o OSP para aprender sobre o assunto		
(CARRINGTON, 2003)	Estatística descritiva: *"My Java tool is an enjoyable way to learn this subject matter. "* 37% concordaram e 26% discordaram	Neutro
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Did you like working as a tester/ developer/ engineer of a FLOSS project? "* 91% confirmaram	Confirma
(XING, 2010)	Estatística descritiva: *"An enjoyable way to learn this subject matter the project I am involved. "* Média igual a4,8	Confirma
(SOWE; STAMELOS, 2007)	Estatística descritiva: *"All 11 students responded "Yes"in both surveys. This means that throughout the pilot study students enjoyed software testing in their projects. "* 100% confirmaram	Confirma
Utilidade da incorporação de projetos de código aberto nas atividades das disciplinas		

(REKHA et al., 2009)	Estatística descritiva: *"Usefulness of incorporating FOSS into coursework. "* 61% concordaram e 4% discordaram	Confirma
Em contribuir com o mundo open source e a sociedade		
(RAJ; KAZEMIAN, 2006)	Relato qualitativo: *"(...) the students felt empowered as they had contributed to the OSS world. "*	Confirma
(REKHA et al., 2009)	Relato qualitativo: *"This gave them the satisfaction of having contributed to the society. "*	Confirma
(LUNDELL; PERSSON; LINGS, 2007)	Relato qualitativo: *"I think all-in-all it was a good experience to make a real contribution to a real open source project. "*	Confirma
Com o suporte dado pelo professor		
(XING, 2010)	Estatística descritiva: *"The instructor is actively helpful in the course. "* Média igual a 5,0	Confirma
(SOWE; STAMELOS, 2007)	Estatística descritiva: *"The students acknowledged the assistance we gave them in selecting their projects. "* 82% confirmaram	Confirma
Com o projeto de código aberto utilizado		
(MARMORSTEIN, 2011)	Estatística descritiva: *"The project was fun. "* 77% concordaram	Confirma parcialmente

A satisfação dos estudantes com a abordagem também pode ser comprovada através de alguma forma de continuidade das atividades. Como podemos visualizar na Tabela 5.17, a 'disposição em continuar a executar atividades em projetos de código aberto' apresentou apenas uma confirmação parcial contra três evidências positivas. Cabe ressaltar que, o mesmo estudo que confirmou parcialmente esta disposição (PAPADOPOULOS; STAMELOS; MEISZNER, 2012), os estudantes também não confirmaram a 'disposição em servir de suporte para futuros estudantes, no emprego da abordagem', apesar de terem demonstrado satisfação em 'em utilizar o OSP para aprender sobre o assunto' (Tabela 5.16). Portanto a falta de interesse de continuidade, nesse caso, pode ter outros motivos, como falta de tempo ou interesse em trabalhos voluntários, e não a insatisfação com o uso de projetos de código aberto.

Por outro lado, estudos distintos apresentaram evidências que comprovam o contentamento dos estudantes com a abordagem, inclusive com alguma forma de continuidade, uma que vez 'participariam de outra disciplina que trabalhasse com projetos de código aberto', 'recomendariam a utilização de projetos de código aberto em outras disciplinas' ou mesmo que 'prefeririam que as disciplinas empregassem projetos de código aberto'.

5.3 INTEGRAÇÃO DA SÍNTESE AOS OBJETIVOS DA PESQUISA

Realizada a síntese, o próximo passo foi analisar estes resultados a luz dos objetivos traçados para a pesquisa. Sendo assim, a partir das evidências reportadas verificamos quais objetivos de aprendizagem podem ser considerados como alcançados, quais as consequências para aprendizagem de modo geral e quais deficiências de formação listadas pela indústria, a abordagem ajuda a suprir. Posteriormente, no Capítulo 9, triangulamos estes resultados com nossos próprios resultados, obtidos com a realização dos estudos de caso.

Tabela 5.17 Evidências reportadas sobre a possibilidade de continuidade.

Categoria		
Artigo	Formato e evidência encontrada	Interpretação
Disposição em continuar a executar atividades em projetos de código aberto		
(REKHA et al., 2009)	Estatística descritiva: *"Willingness to continue FOSS activities."* 76% concordaram e 5% discordaram	Confirma
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Will you continue to participate in the FLOSS project after the completion of the assignment?"* 68% confirmaram	Confirma parcialmente
(LUNDELL; PERSSON; LINGS, 2007)	Relato qualitativo: *"Wishing to continue involvement in OS projects was a recurring theme."*	Confirma
(SOWE; STAMELOS, 2007)	Estatística descritiva: *"All the students were considering participating in their projects after they graduate."* 100% confirmaram	Confirma
Participaria de outra disciplina que trabalhasse com projetos de código aberto		
(RAJ; KAZEMIAN, 2006)	Estatística descritiva: *"Would take another course that provided opportunity to modify OSS."* 85% concordaram	Confirma
Disposição em servir de suporte para futuros estudantes, no emprego da abordagem		
(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)	Estatística descritiva: *"Would you be interested in helping students next year by assuming the role of forum moderators, consultants, etc.?"* 53% confirmaram	Não confirma
Recomendaria a utilização de projetos de código aberto em outras disciplinas		
(RAJ; KAZEMIAN, 2006)	Relato qualitativo: *"The students suggested several courses. The courses suggested included our Computer Science IV, which is our UML and C++ course that currently uses standalone course projects, and Software Engineering I, which has a scratch-to-finish team-based course project that covers the entire software lifecycle."*	Confirma
Preferiria que as disciplinas empregassem projetos de código aberto		
(SOWE; STAMELOS, 2007)	Estatística descritiva: *"Most of the students expressed that they would prefer to have their other courses taught using F/OSS methodology, especially towards the end of the study, in survey two where all the students answered in the affirmative."* 100% confirmaram	Confirma

Tabela 5.18 Heurística para análise do alcance do objetivo segundo as evidências reportadas.

Heurística	Alcance do Objetivo
Todas as evidências confirmam a ocorrência da categoria	Sim
Todas as evidencias não confirmam a ocorrência da categoria	Não
Todas as evidências confirmam parcialmente a ocorrência da categoria	Parcial
Todas as evidências representam uma situação de neutralidade com relação à categoria	Não
Evidências que confirmam e evidências que confirmam parcialmente a ocorrência da categoria	Sim
Evidências que confirmam e evidências que representam uma situação de neutralidade com relação à categoria	Sim
Evidências que confirmam e evidências que não confirmam a ocorrência da categoria	Parcial
Evidências que confirmam parcialmente e evidências que não confirmam a ocorrência da categoria	Parcial
Evidências que representam uma situação de neutralidade e evidências que não confirmam a ocorrência da categoria	Não

5.3.1 Objetivos de aprendizagem em ES alcançados segundo evidências existentes nos estudos relacionados

Após capturarmos e categorizarmos as evidências reportadas nos diversos estudos, podemos compará-las aos objetivos de aprendizagem estabelecidos no Capítulo 3. O propósito é avaliar quais objetivos de aprendizagem possuem categorias de evidências correlacionadas e, de acordo com essas evidências, quais destes podem ser considerados como alcançados. Visto que algumas categorias agregam mais de uma evidência e estas evidências podem ter interpretações distintas, precisamos especificar mais um conjunto de heurísticas para o julgamento se o objetivo de aprendizagem correspondente foi alcançado, parcialmente alcançado ou não alcançado. A Tabela 5.18 lista as heurísticas estabelecidas e o respectivo julgamento para o alcance do objetivo de aprendizagem.

Como resultado da correlação entre os objetivos de aprendizagem estabelecidos e as categorias encontradas na síntese, apresentamos a Tabela 5.19. A tabela lista os objetivos de aprendizagem, as categorias relacionadas, o conjunto de interpretação das evidências para cada categoria e finalmente, o julgamento se o objetivo foi alcançado, parcialmente alcançado ou não foi alcançado, segundo as heurísticas definidas. Para descrever o conjunto de interpretação das evidências, utilizamos uma codificação, onde as letras representam um ponto da escala de interpretação definida na Tabela 5.3 e os números correspondem à quantidade de evidências com esta interpretação. De modo que, o conjunto '3C-2CP-1N-1NC' representaria, por exemplo, que três evidências confirmam a ocorrência da categoria, duas confirmam parcialmente, uma representa uma situação de neutralidade e, por fim, uma não confirma a ocorrência da categoria.

Tabela 5.19: Alcance de objetivos de aprendizagem segundo as evidências reportadas.

Id.	Objetivo de Aprendizagem	Categoria	Conjunto de interpretação das evidências	Alcance do objetivo
LO157	Descrever os passos necessários para planejar e dar andamento em um projeto de software em um ambiente real.	Conhecer as fases que compreendem o desenvolvimento de um software real.	2C	Sim
LO52	Enfatizar a importância do design do software.	Compreender sobre design de software.	3C	Sim

LO156	Descrever como desenvolver um software é diferente de um esforço individual de um programa, no que diz respeito a compreender uma base de código grande, o processo de build, e o contexto de mudanças constantes.	Descrever o impacto da complexidade do projeto nas abordagens usadas para desenvolver software.	1C	Sim
		Descrever o impacto do tamanho do projeto nas abordagens usadas para desenvolver software.	1C	
LO177	Reconhecer atributos de qualidade de software.	Identificar questões de qualidade do software.	2C	Sim
LO6	Capacidade de modelar e especificar os requisitos de software de médio porte, usando métodos comuns.	Elaborar diagramas.	1C	Sim
LO38	Construir os modelos apropriados para modelar a estrutura e o comportamento que atendam as especificações dos requisitos do software.	Elaborar diagramas.	1C	Sim
LO63	Demonstrar habilidade em programação	Programar.	2C-1N	Sim
LO66a	Aplicar várias estratégias de testes em programas simples.	Testar software.	1C	Sim
LO66b	Aplicar várias estratégias de depuração (debug) de programas simples.	Depurar código.	1CP	Parcial
LO100	Realizar o processo de testes e correção de erros sem que haja a reintrodução de erros antigos.	Corrigir bugs do software.	1CP-2NC	Parcial
LO124	Aplicar engenharia reversa em um software de médio porte.	Executar engenharia reversa	1C	Sim
LO118	Realizar a manutenção de um software real de médio porte.	Modificar software de maior porte	1C-1CP-1N	Sim
LO47	Demonstrar a capacidade de usar ferramentas de modelagem de design em suporte ao desenvolvimento de um projeto de médio porte.	Usar ferramentas de modelagem de design de software.	1C	Sim
LO82	Construir, executar e depurar (debug) programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração (debug) visual.	Usar ferramentas de desenvolvimento de software.	1C	Parcial*
LO199	Distinguir comportamento profissional ético e não ético.	Compreender qual deve ser o comportamento ético do profissional de computação.	1C-1NC	Parcial
LO227	Habilidade de trabalhar efetivamente como membro de equipes.	Trabalhar em equipe.	3C	Parcial
		Trabalhar com profissionais externos.	2C-1N-2NC	
LO234	Coordenar o próprio trabalho com relação ao trabalho dos outros (habilidade de gerenciamento de tempo e auto-organização).	Auto-organizar o próprio trabalho.	1C	Sim
LO266	Ser capaz de solucionar problemas.	Ser capaz de solucionar problemas.	2C	Sim
LO267	Gerar soluções alternativas para os problemas.	Ser capaz de propor outras soluções.	1C	Sim
LO268	Ter experiência com código desenvolvido por terceiros.	Entender código desenvolvido por terceiros.	2C-1CP	Sim
LO270	Compreender como teoria e prática influenciam uma a outra.	O uso de projetos reais permite colocar em prática toda a teoria estudada.	1C	Sim

		Com a prática em projetos reais é possível enxergar as razões de aplicar a teoria.	1C	
LO243	Interagir com outras pessoas (interagir profissionalmente com os stakeholders).	Interagir com a comunidade do projeto.	3C-1CP-1NC	Parcial
LO246	Escrever documentação, clara, concisa e acurada.	Saber elaborar documentação útil para terceiros.	1C	Sim
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).	Entender documentação elaborada por terceiros.	1C-1N	Sim
LO114	Perceber a importância da documentação para a manutenção de software.	Conscientização sobre a importância da documentação.	6C	Sim
LO115	Entender as nuances da programação e os desafios envolvidos em escrever código limpo	Conscientização sobre melhores práticas de programação.	1C	Sim
LO117	Compreender a importância da manutenção e evolução em um projeto de software, especificamente considerando o tamanho do esforço necessário nessas atividades.	Conscientização da importância da manutenção e evolução do software.	2C	Sim
LO269	Ter vivenciado pelo menos um projeto real.	Vivência de um projeto real.	4C	Sim

* A falta de detalhamento da evidência não nos permitiu verificar se o objetivo como um todo foi alcançado.

Fazendo a ressalva da vulnerabilidade da correlação entre os objetivos de aprendizagem e as categorias de evidências, devido à interpretação subjetiva envolvida, podemos constatar, através da Tabela 5.19, que as evidências reportadas demonstram o alcance de diversos objetivos de aprendizagem distribuídos entre as várias áreas de conhecimento da engenharia de software definidas no SWEBOK, como processos de software, qualidade de software, *design* de software, construção de software, manutenção de software, além da prática profissional em engenharia de software. Ao todo foram 22 objetivos de aprendizagem alcançados e seis parcialmente alcançados. Esses resultados demonstram a possível abrangência e potencial da abordagem de utilização de OSP como recurso didático no ensino da engenharia de software.

5.3.2 Outras consequências para aprendizagem identificadas a partir das evidências reportadas

No que diz respeito à aprendizagem, detectamos outras categorias de evidências que não possuem objetivos de aprendizagem com os quais podemos correlacioná-las. Algumas estão diretamente ligadas à aprendizagem conceitual como 'compreender sobre construção de software' e 'descrever benefícios e desvantagens de projetos de código aberto'. Outras são relevantes para a possível mudança de atitudes dos estudantes devido a sua conscientização sobre aspectos importantes da engenharia de software e, por fim, algumas são fundamentais na preparação dos estudantes para o mercado de trabalho, seja pela visão profissional proporcionada, seja pelo sentimento de capacitação ou mesmo pelo desenvolvimento da autoconfiança. No caso particular da categoria de 'usar ferramentas', existem vários objetivos de aprendizagem relacionados ao uso de ferramentas, porém são específicos de certos tipos de ferramentas. O pouco detalhamento das evidências repor-

tadas para esta categoria não nos permitiu categoriza-las e associá-las a nenhum desses objetivos.

Enumeramos as categorias mencionadas na Tabela 5.20, juntamente com o conjunto de interpretação das evidências para cada categoria e o julgamento sobre a aprendizagem proporcionada. Convém ressaltar que neste caso, aplicamos a mesma codificação para o conjunto de evidências e o mesmo método de julgamento empregados para análise do alcance dos objetivos de aprendizagem, descritos em 5.3.1.

Tabela 5.20 Outras aprendizagens proporcionadas.

Categoria	Conjunto de interpretação das evidências	Aprendizagem proporcionada
Compreender sobre construção de software.	3C	Sim
Descrever benefícios e desvantagens de projetos de código aberto.	4C	Sim
Usar ferramentas.	2C	Sim
Reconhecer a importância da modelagem.	1C	Sim
Conscientização sobre a importância do código autodocumentado.	1C	Sim
Conscientização sobre a importância dos projetos de código aberto.	2C	Sim
Conscientização da importância de aprender sobre projetos de código aberto.	1C	Sim
Conscientização de que seu trabalho encaixa-se em um contexto maior.	3C	Sim
Vivência de um projeto real	4C	Sim
Visão profissional.	2C	Sim
Desenvolvimento da autoconfiança.	2C-2N	Sim
Capacitação para o Mercado	6C-1CP	Sim
Provê uma experiência de mundo real	3C-1CP	Sim
Liberdade de modificar o software	1C	Sim
Permite trabalhar com projeto existente	1C	Sim

5.3.3 Relação entre as evidências reportadas e as lacunas de formação apontadas pela indústria

Para analisarmos se a abordagem de uso de projetos de código aberto ajuda ou não a suprir as lacunas de formação apontadas pela indústria, relacionamos as categorias de evidências detectadas ao realizarmos a síntese com as deficiências descritas na Tabela 3.5.

A Tabela 5.21 apresenta as deficiências organizadas por área de conhecimento do SWEBOK, as categorias relacionadas, o conjunto de interpretação das evidências para cada categoria e finalmente, o julgamento se a abordagem contribuiu, contribuiu parcialmente ou não contribuiu para o suprimento da deficiência. Novamente, aplicamos a mesma codificação para o conjunto de evidências e o mesmo método de julgamento empregados para análise do alcance dos objetivos de aprendizagem, descritos em 5.3.1.

De acordo com as evidências reportadas, a Tabela 5.21 mostra que a abordagem de uso de OSP contribui para o suprimento de algumas deficiências de formação nas

Tabela 5.21 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com a síntese das evidências.

Deficiência	LO	Categoria	Conjunto de interpretação das evidências	Contribuição
Design de Software				
Nível de detalhamento dos modelos	LO38	Elaborar diagramas.	1C	Sim
Construção de Software				
Comentar o código adequadamente	LO62	-	*	Parcial
Testes e localização do erro (debugging) no código	LO66a	Testar software.	1C	Sim
	LO66b	Depurar código.	1CP	Parcial
Uso de ambientes integrados de desenvolvimento	LO82	Usar ferramentas de desenvolvimento de software.	1C	Sim
Teste de Software				
Processo de testes e correção de erros sem que haja a reintrodução de erros antigos	LO100	Corrigir bugs do software.	1CP-2NC	Parcial
Manutenção de Software				
Manutenção de software	LO118	Modificar software de maior porte	1C-1CP-1N	Sim
Prática Profissional em Engenharia de Software				
Comportamento ético	LO199	Compreender qual deve ser o comportamento ético do profissional de computação.	1C-1NC	Parcial
Gerenciamento do tempo	LO234	Auto-organizar o próprio trabalho.	1C	Sim
Experiência com trabalho em equipe	LO227	Trabalhar em equipe.	3C	Parcial
		Trabalhar com profissionais externos.	2C-1N-2NC	
Solução de problemas	LO266	Ser capaz de solucionar problemas.	2C	Sim
Propor soluções alternativas	LO267	Ser capaz de propor outras soluções.	1C	Sim
Experiência com projetos reais	LO269	Vivência de um projeto real.	4C	Sim
Habilidade de comunicação escrita para produção de documentação técnica	LO246	Saber elaborar documentação útil para terceiros.	1C	Sim
Hesitação em solicitar ajuda	LO249	-	*	Parcial

(*) Ver justificativas no texto, para considerarmos a contribuição parcial.

seguintes áreas de conhecimento do SWEBOK: design de software, construção de software, manutenção de software e prática profissional em engenharia de software. Também houve a contribuição parcial para a área de testes de software. No total, foram nove contribuições positivas e quatro contribuições parciais. Portanto sob a perspectiva de colaborar para suprir as deficiências de formação indicadas pela indústria, a abordagem também demonstra alguma abrangência e potencialidade.

Observamos ainda que, de alguma forma, a abordagem contribui para suprir as deficiências que dizem respeito à habilidade de 'comentar o código adequadamente' e a 'hesitação em solicitar ajuda'. Nossa suposição fundamenta-se nas evidências reportadas sobre a 'conscientização sobre a importância do código autodocumentado' e sobre o desenvolvimento da habilidade de 'interagir com a comunidade do projeto', respectivamente. Se o estudante está mais consciente da importância do código autodocumentado é mais provável que ele procure comentar o código adequadamente. Do mesmo modo, se o estudante desenvolve a habilidade de interagir com a comunidade do projeto, provavelmente as barreiras porventura existentes são quebradas, de maneira que o estudante possivelmente não hesitará em solicitar ajuda.

No próximo capítulo descrevemos e apresentamos os resultados obtidos com a adoção de um OSP para a aprendizagem de evolução e manutenção de software.

Capítulo

6

ESTUDO DE CASO 1 – EVOLUÇÃO E MANUTENÇÃO DE SOFTWARE

Para incorporar evolução e manutenção de software no currículo de maneira que os estudantes não tenham apenas uma visão teórica e superficial da área, é essencial uma visão autêntica de como ocorre o processo de manutenção na indústria, o que exige a execução de modificações de código na prática. Nesse sentido, o primeiro passo é definir o software com o qual eles irão trabalhar. Este código pré-existente deve conter características similares aos códigos produzidos nas empresas, ao mesmo tempo que seja adequado ao conhecimento e preparação apresentada pelos estudantes (SMITH et al., 2014). A possibilidade de emprego de softwares de código aberto aparece como uma oportunidade. A disponibilidade do código permite que os estudantes lidem com projetos existentes, a partir dos objetivos de aprendizagem definidos pelo professor.

Decidimos, como ponto de partida para a nossa própria experiência com a abordagem, realizar um estudo de caso exploratório para investigar as consequências da utilização de um projeto de código aberto como software a ser trabalhado pelos estudantes na disciplina. Como no momento que este estudo foi realizado ainda estávamos em um estágio inicial da pesquisa, outras questões de pesquisa também foram tratadas, mas que não serão descritas neste documento porque fogem aos objetivos finais traçados para a tese.

O propósito desse capítulo é apresentar como o OSP foi utilizado e as consequências da execução de modificações no seu código no processo de aprendizagem dos estudantes. Sendo assim, inicialmente descreveremos todo o contexto de aplicação da abordagem, incluindo a metodologia usada na coleta e análise dos dados. Posteriormente, reportaremos as consequências capturadas a partir da percepção dos estudantes e professor. Encerraremos o capítulo discorrendo sobre como os resultados deste estudo exploratório foram integrados aos demais resultados da pesquisa.

6.1 CONTEXTO DO ESTUDO DE CASO EM EVOLUÇÃO E MANUTENÇÃO DE SOFTWARE

Para melhor organização do texto, dividimos a descrição do contexto do estudo nos seguintes tópicos: (i) cenário do estudo de caso; (ii) população do estudo e envolvidos; (iii) aplicação da abordagem; (iv) coleta de dados e (v) análise de dados.

6.1.1 Cenário do Estudo de Caso

O cenário desse estudo caso foi a disciplina Evolução e Manutenção de Software ofertada no segundo semestre de 2013 [1], para o curso de Engenharia da Computação da Universidade Estadual de Feira de Santana (UEFS). Uma importante característica deste curso é a utilização da abordagem *Problem Based-Learning* (PBL) nos componentes curriculares chamados de módulos integradores.

A disciplina de Evolução e Manutenção de Software, caso deste estudo, não compreende nenhum dos módulos integradores, mas é uma disciplina optativa que tem como pré-requisito o módulo integrador de Engenharia de Software e os módulos teóricos de Engenharia de Software e Análise de Sistemas.

No decorrer da disciplina, o professor utilizou-se de aulas expositivas, projetos práticos e seminários como métodos didáticos. O professor adotou o livro de Rajlich (RAJLICH, 2012) como referência principal para a disciplina. Portanto, o modelo do processo de mudanças incrementais empregado em (BUCHTA et al., 2006) (PETRENKO et al., 2007) (RAJLICH, 2013) foi proposto para que os estudantes executassem os projetos práticos. As aulas expositivas trataram de todo o conteúdo introdutório e fundamental sobre evolução e manutenção, como caracterização das mudanças, modelos de tempo de vida de software, envelhecimento de software, além de apresentar o modelo de processo de mudança, cujas atividades envolvem: localização de conceitos, análise de impacto, atualização, refatoração, testes, conclusão das mudanças e controle de versões. Os seminários, por sua vez, cobriram tópicos mais avançados, a exemplo de: engenharia reversa, recuperação arquitetural, reengenharia de sistemas legados etc. Por essa razão, os seminários foram deixados para a etapa final da disciplina, enquanto que os projetos práticos foram executados em paralelo com as aulas expositivas, com exceção das duas primeiras semanas do curso, quando ocorreram apenas aulas expositivas. Os estudantes foram avaliados a partir da média de duas notas. A primeira composta pelas notas obtidas com a execução dos projetos práticos de manutenção e a segunda correspondeu a nota do seminário.

Com o objetivo de que o estudante se defrontasse com o contexto de manutenção de software mais próximo possível da realidade, o professor escolheu utilizar o projeto de código aberto JabRef para definir os projetos de manutenção a serem realizados pelos estudantes. Um estudante de pós-graduação auxiliou-o no planejamento das atividades, identificando em um *branch* de uma *release* anterior do projeto, requisições de modificação de complexidade crescente e apropriadas à aprendizagem dos estudantes, dentre as requisições existentes no *bug/issue tracker* do projeto e já solucionadas pela comunidade.

[1] O segundo semestre de 2013 ocorreu no período de setembro a dezembro do mesmo ano.

6.1.2 População do Estudo e Envolvidos

Nossa população de estudo foi composta pelo professor e os estudantes da disciplina, sendo que, dos 30 matriculados, dois estudantes não compareceram a nenhuma aula e cinco desistiram ao longo do curso.

Por se tratar de uma disciplina optativa e seus pré-requisitos serem ofertados logo no quarto período curricular, encontramos estudantes com diferentes níveis de conhecimento e experiência. Assim, além de alguns estudantes terem cursado outras disciplinas de engenharia de software, alguns tiveram diferentes experiências em desenvolvimento de projetos de software, seja na indústria, seja participando de projetos de iniciação científica ou na empresa júnior da Universidade.

Quanto ao professor da disciplina, este tem mais de 10 anos de experiência de ensino. Apesar de já ter ensinado evolução de software em outra oportunidade, esta foi a sua primeira experiência em um curso de graduação. Seu conhecimento na área de evolução de software é principalmente acadêmico, já que não vivenciou os problemas de evolução na indústria de software. Contudo, desde 2007, essa é uma de suas áreas de interesse de pesquisa. Considerando a sua experiência com projetos de código aberto, relatou ser usuário de alguns softwares, ferramentas de apoio ao desenvolvimento, bibliotecas de coleções e funções matemáticas. Participou da construção da *Design Suite*, uma suíte aberta de ferramentas para a recuperação e checagem de design arquitetural, originada na universidade na qual cursou o doutorado. Também durante sua pesquisa no doutorado, fez a mineração de vários repositórios de projetos de código aberto. Todavia, em suas experiências de ensino, esta foi a primeira vez que os estudantes precisariam alterar o código de um OSP.

No que diz respeito à pesquisa, contamos com a participação de um estudante de iniciação científica na coleta de dados, especificamente na observação das aulas, com o intuito de obtermos uma cobertura mais abrangente. As entrevistas e análise dos dados ficaram sob a responsabilidade da autora deste trabalho. Convém salientar que nenhum desses pesquisadores tinha quaisquer outros interesses com a pesquisa, a não ser explorar, entender e avaliar a abordagem sendo empregada.

6.1.3 Aplicação da abordagem

O professor da disciplina foi o responsável por toda a proposta didática empregada. Desse modo, a aplicação da abordagem foi totalmente integrada as demais atividades da disciplina. Como mencionado, na primeira etapa da disciplina, após as duas primeiras semanas, as aulas expositivas foram intercaladas com aulas práticas, onde os estudantes precisavam colocar em prática os conceitos discutidos nas aulas téoricas, em particular o modelo do processo de mudança citado, para modificar o JabRef.

O JabRef, projeto de código aberto selecionado, representa um gerenciador de referencias bibliográficas bastante utilizado pela comunidade acadêmica. Corresponde a um projeto de médio porte, desenvolvido em Java, para ambiente *desktop*.

Os estudantes divididos em equipes tiveram que executar dois projetos práticos de

manutenção no JabRef[2]. Os projetos apresentavam um nível crescente de complexidade e cada equipe, composta em média por cinco membros, deveria atender a determinada solicitação de mudança.

Uma vez que o código fonte do projeto está localizado no sistema de controle de versões Git, o professor, na primeira aula prática, apresentou um tutorial sobre a ferramenta.

6.1.4 Coleta de Dados

Como mencionamos no Capítulo 4, utilizamos entrevistas, observações e questionários como instrumentos de coleta de dados neste estudo de caso.

Realizamos três fases de entrevistas. A primeira fase ocorreu no início da disciplina, a segunda fase ocorreu após a execução dos projetos práticos e a terceira fase ocorreu durante a penúltima semana de aula, um pouco antes do término das apresentações dos seminários. Enquanto que na primeira fase, buscamos identificar o conhecimento e as experiências prévias dos envolvidos na pesquisa, nas demais fases, procuramos capturar suas percepções, considerando as questões de pesquisa traçadas para o estudo. Como os estudantes participaram voluntariamente das entrevistas, não conseguimos que todos participassem de todas as fases. O resumo das pessoas entrevistadas em cada fase pode ser visto na Tabela 6.1, bem como a experiência particular de cada estudante sobre a participação prévia em projetos de software. As entrevistas seguiram o modelo semiestruturado e foram gravadas utilizando-se de dispositivos de áudio. Posteriormente, transcrevemos e revisamos estas transcrições.

Tabela 6.1 Entrevistas realizadas por fase e experiência prévia dos estudantes participando de projetos.

Participantes	**Fase 1**	**Fase 2**	**Fase 3**	**Experiência prévia dos estudantes com projetos**
Estudante SE1	X	X	X	Apenas participou dos projetos das disciplinas durante o curso.
Estudante SE2	X	X	X	Também participou de projetos na iniciação científica e estágio.
Estudante SE3	X			Também participou de projetos na iniciação científica, empresa júnior e estágio.
Estudante SE4	X			Apenas participou dos projetos das disciplinas durante o curso.
Estudante SE5		X		Também participou de projetos no estágio.
Estudante SE6		X		Também demonstrou bastante experiência em projetos, disse já ter trabalhado em três empresas.
Estudante SE7			X	Apesar de afirmar que não tinha muito interesse pela área, também relatou alguma experiência em projetos no estágio.
Estudante SE8			X	Também participou de um projeto na iniciação científica.
Professor P1	X	X	X	-

Como o estudante de iniciação científica, após o devido treinamento, ficou encarregado

[2]Descritos em https://sites.google.com/site/ecompevolucao20132/trabalhos

pelas observações das aulas, conseguimos o registro de quase a totalidade das aulas, ou seja, 25 das 28 aulas onde houve a discussão do conteúdo da disciplina.

Aplicamos ainda um questionário web anônimo para capturarmos de modo quantitativo a percepção dos estudantes sobre o curso e a abordagem de ensino empregada. Todavia, dado que o mesmo foi aplicado após o término da disciplina, obtivemos uma baixa taxa de resposta. Apenas nove estudantes responderam, dentre os 23 que terminaram a disciplina. Entretanto, a maioria das questões tratavam de assuntos que terminaram fugindo do escopo dessa tese.

6.1.5 Análise de dados

No que diz respeito a análise de dados, para os dados qualitativos, aplicamos a análise descrita no Capítulo 4. Quando possível, acrescentamos, às categorias detectadas, dados quantitativos associados, de forma descritiva.

6.2 RESULTADOS

Os resultados do estudo exploratório são fruto da análise qualitativa aplicada, caracterizando assim, o estágio inicial de nossa pesquisa, onde os objetivos finais ainda não haviam sido traçados. Portanto, buscando explorar, entender e avaliar a abordagem sendo empregada, codificamos as informações consideradas relevantes em categorias, agrupamolas em temas e identificamos relacionamentos entre estes temas. A Figura 6.1 ilustra os principais temas encontrados e seus relacionamentos.

No decorrer da seção, discorreremos sobre cada um dos temas apresentados na Figura 6.1, as respectivas categorizações, informações quantitativas associadas (quando adequado) e justificativas para os relacionamentos estabelecidos entre os mesmos.

6.2.1 Projeto de Código Aberto

Como o principal objetivo dessa análise era investigar as consequências da utilização de projetos de código aberto como recurso didático em um curso de evolução e manutenção de software, o tema 'projeto de código aberto' surge naturalmente. Na análise dos dados resultantes das entrevistas, identificamos que o tema agrega um conjunto de situações que este tipo de projeto proporciona ao processo de ensino-aprendizagem (ver Figura 6.2).

Como mostra a Figura 6.2[3], uma das situações proporcionadas é a possibilidade de trabalhar com um 'projeto existente'. Sendo assim, os estudantes distinguiram o cenário de utilizar um projeto de código aberto, onde o software já existe, da que normalmente acontece nas disciplinas de engenharia de software, onde o software é desenvolvido do zero.

> *"Que você tá acostumado em desenvolver do início seu projeto pegar sua classe, modelar, desenvolver, já você pegar um sistema em andamento, aí... o JabRef "* (SE6).

[3]O NVIVO (ferramenta utilizada durante a análise qualitativa) descreve o relacionamento de uma categoria filha para a categoria pai como "Principal".

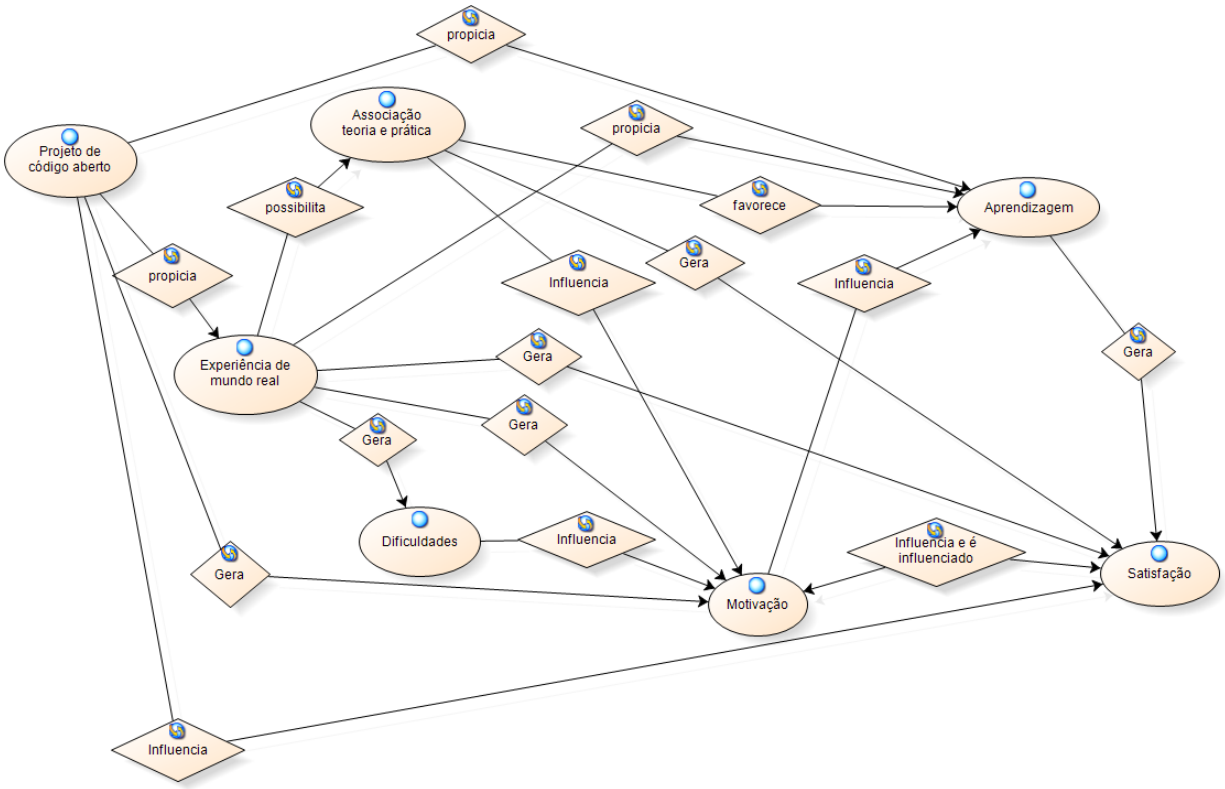

Figura 6.1 Temas provenientes da adoção de um OSP para a aprendizagem de evolução e manutenção de software.

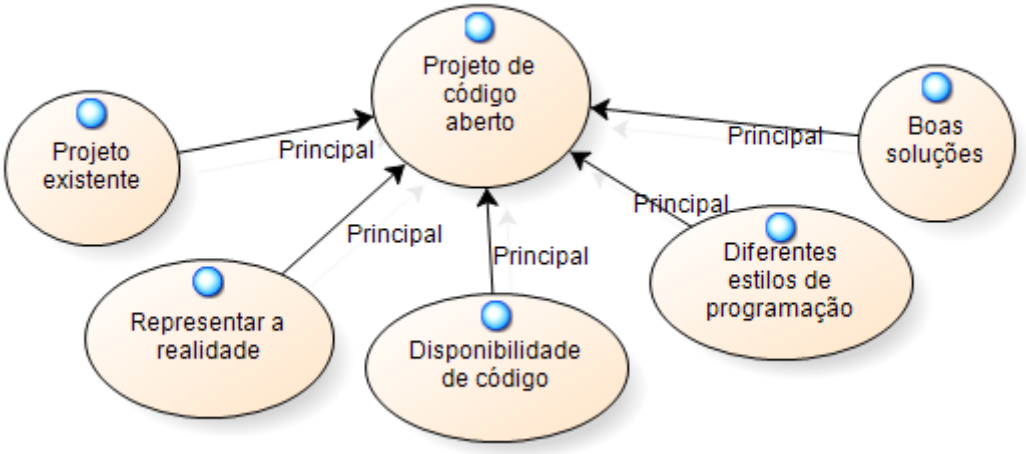

Figura 6.2 Situações proporcionadas por um projeto de código aberto.

> *"(...) saiu um pouco dessa questão do PBL, que a gente desenvolve do zero e pegou uma coisa que já existe, que é usada no mercado (...)''* (SE1, fase 3).

> *"(...) Porque é isso que vai acontecer. Você não vai... geralmente você não vai pegar um problema do início, e mesmo que pegue do início, você... vai ter que... você vai ter que se alertar de todos esses problemas que... esses eventuais problemas que vão ocorrer''* (SE8).

O último estudante ainda ressaltou que, mesmo que o software seja desenvolvido desde o início, o profissional precisa ter conhecimento do que poderá acontecer com o software durante seu ciclo de vida.

Outra situação relevante é o fato desse tipo de projeto 'representar a realidade' (Figura 6.2), ou seja, não constitui um software criado somente para atender as necessidades da disciplina. Estudantes e professor reconhecem esse fato. Inclusive em duas ocasiões, apontam o mesmo como ponto positivo no ensino de evolução.

> *"(...) um exemplo real como o professor pegou o JabRef (...)''* (SE6)

> *"Eu acho legal porque você consegue é... trazer coisas... que são usadas no mundo real e isso dá um, dá um diferencial legal, acho que isso é... traz você pra perto da realidade (...)''* (SE1, fase 2).

> *"Trabalhar num contexto de um sistema real isso é muito bom, extremamente positivo, é um software de verdade, as pessoas estão utilizando aquele software, acho que isso tem um impacto bem interessante pros estudantes, eles estão fazendo mudanças num software que já existe, não é um brinquedo, então isso é muito bom, é... outro ponto positivo (...)''* (P1, fase 2).

As situações de ser um 'software existente' e 'retratar a realidade' também são características dos softwares usados nas empresas. Todavia, na maioria das vezes, o código é proprietário, não disponível para utilização como recurso didático. O seu uso exige a condução de algum processo burocrático. Os estudantes apontam a 'disponibilidade de código' como outra situação proporcionada por projetos de código aberto (Figura 6.2) e que é fundamental para uma disciplina como evolução e manutenção de software.

> *"Eu acho que só tem eles pra gente analisar o código. Porque, assim, se não fosse esses códigos open source que tivessem pela aí disponível, você... tá tudo fechado o restante. Como é que você iria poder analisar um código mesmo, fazer uma disciplina dessa, por exemplo''* (SE8).

> *"Porque trabalhar com ferramentas com códigos fechados requer uma burocracia maior, creio eu nunca mexi com... mas requer uma burocracia a partir de quem fez e tal... a ferramenta open source tá livre pra todo mundo''* (SE5).

Como visto, as três situações 'software existente', 'representar a realidade' e 'disponibilidade de código' são essenciais à disciplina de evolução e manutenção de software. Contudo, os estudantes ainda assinalaram duas outras situações possibilitadas pelo uso de projetos de código aberto: 'diferentes estilos de programação' e 'boas soluções' (ver Figura 6.2).

Segundo um dos estudantes, a dinâmica das equipes que participam do projeto de código aberto permite que os estudantes defrontem-se com 'diferentes estilos de programação', fato que também é importante para a aprendizagem:

> *"(...) quem começa a mexer no código não é quem vai terminar... então assim, a gente aprende a lidar com as diferenças de programação e a entender melhor os códigos'*' (SE2, fase 2).

Em outro momento, esse mesmo estudante destaca que essa situação também é uma oportunidade para identificar casos de boas práticas, ou seja, 'boas soluções'.

> *"(...) às vezes, a gente programa de formas diferentes, né? Algumas pessoas fazem ali o código, tal... outras pessoas têm uma estrutura diferente de organização, então, dentro do código deve existir... pode existir alguma prática interessante de desenvolvimento, que nesse caso se aplicaria (...)"* (SE2, fase 3).

Outro estudante corrobora com o mesmo ponto de vista:

> *"(...) você olha como tá estruturado você... até coisas boas você pode... você acaba adquirindo, algumas práticas como ele estruturou o código, como ele dividiu em classes, eu achei que foi bastante interessante (...)'*' (SE6).

6.2.2 Experiência de Mundo Real

O propósito principal da abordagem de utilização de um projeto de código aberto é propiciar ao estudante uma 'experiência de mundo real' (ver Figura 6.1). Nesse estudo, os dados refletem essa situação pela proximidade do projeto utilizado (JabRef) aos projetos existentes nas empresas e pela proximidade das atividades de manutenção executadas nos projetos práticos, com as atividades que podem ser solicitadas no mercado de trabalho (ver Figura 6.3).

Proximidade dos softwares existentes nas empresas

Em 6.2.1, reportamos a percepção de alguns estudantes de que o projeto de código aberto representa a realidade. Agora discorreremos sobre características que demonstram a sua 'proximidade dos projetos existentes nas empresas'. A Figura 6.3 apresenta as características arroladas pelos estudantes e pelo professor.

Os estudantes consideraram o projeto de código aberto utilizado na disciplina como grande e complexo:

> *"Nesse projeto, a gente teve a noção de um sistema grande (...)"* (SE6).

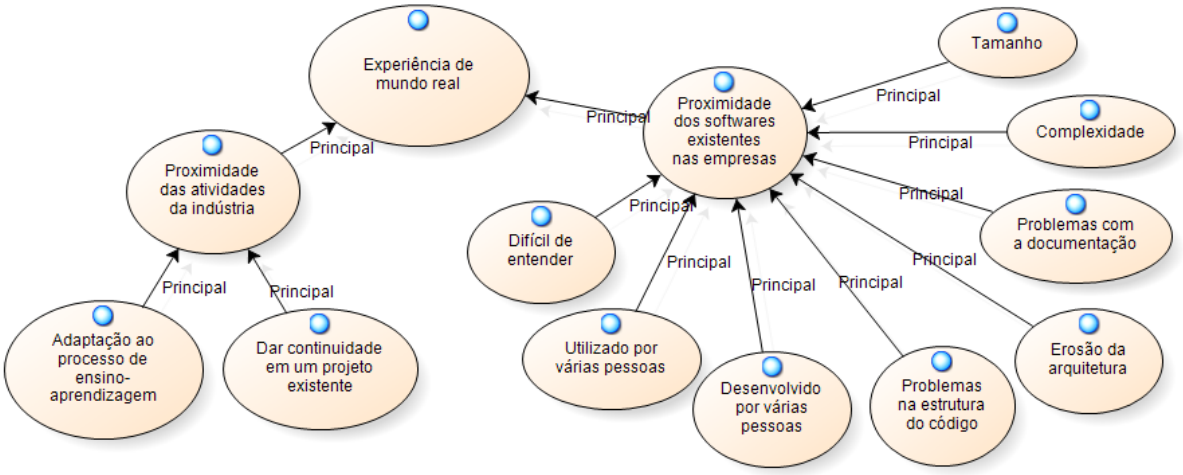

Figura 6.3 Características que aproximam a abordagem de uso do OSP de uma experiência de mundo real.

> *"(...) código fonte complexo, em termos de números de classe é muito grande, muitos conceitos envolvidos (...)"* (SE1, fase 3).

Associando dados quantitativos, entre os respondentes do questionário, 89% consideraram que o JabRef representa a realidade, no sentido em que apresenta complexidade e tamanho similares aos códigos de projetos produzidos em empresas.

Por outro lado, de acordo com o professor, o JabRef não representa um software realmente grande, mas sim de médio porte. Contudo, exibe características próprias de softwares desse porte encontrados nas empresas, como: 'problemas com a documentação', 'erosão da arquitetura' e 'problemas na estrutura do código', consequentemente algumas vezes são 'difíceis de entender' (ver Figura 6.3).

> *"(...) ele traz todos os problemas de um sistema de médio porte, né, que ele é difícil de entender, porque ele não tem uma documentação tão clara, ele não é... ele já tem um... uma certa erosão da arquitetura e da organização do código, então, às vezes é realmente difícil de entender (...)"* (P1, fase 2).

Os estudantes também perceberam a existência de 'problemas na estrutura do código'. Capturamos essa percepção tanto nas observações das aulas, quanto nas entrevistas.

Em uma aula, um estudante comentou:

> *"Se aquele negócio tá organizado, eu acho que ele está tão organizado que eu não entendi essa organização"* (observação da aula ocorrida em 29/10).

Em outra aula, outro estudante comentou:

> *"No JabRef, você entra em um pacote, depois volta para o outro, depois vai para um terceiro e depois vai para o primeiro de volta"* (observação da aula ocorrida em 05/11).

Todavia, dois estudantes ressaltaram que este problema de estrutura do código não é exclusividade do projeto de código aberto empregado. Em suas experiências de trabalho, visualizaram problemas semelhantes ou até mesmo piores.

> *"(...) grande parte do código que eu trabalhei, era feito por outros estagiários... e não tinha um nível assim de... de conhecimento, era realmente uma bagunça... que eu passava noites tentando achar uma função simples, que tava na cara... mas que fazia tanta coisa por cima, que não ficava transparente... era bem complicado" (SE5).*

> *"(...) em algumas empresas, eu acho que tem mais problemas, eu acho que é mais... um pouquinho mais organizado, o JabRef" (SE6).*

Outras duas características, encontradas no projeto utilizado, espelham os projetos presentes nas empresas, são elas: ser 'desenvolvido por várias pessoas' e ser 'utilizado por várias pessoas'.

Pelo fato de ser desenvolvido por várias pessoas, um estudante ressalta que o projeto deve apresentar os mesmos tipos de problemas encontrados nos softwares existentes nas empresas.

> *"Nesse projeto, a gente teve a noção de um sistema grande que foi desenvolvido por várias pessoas e você vê realmente como seria um software quando você chegar num ambiente empresarial, chegar numa empresa, você verifica que você vai dar de cara justamente com um problema parecido com o do JabRef"* (SE6).

Outro estudante acrescenta que de maneira semelhante ao que ocorre nas empresas, a equipe que participa do projeto de código aberto modifica-se ao longo do tempo.

> *"No mercado a gente vai trabalhar com equipe, essa equipe pode ir se modificando e é mais ou menos o que acontece dentro... no código open source, não é sempre as mesmas pessoas que mexem, quem começa a mexer no código não é quem vai terminar (...)"* (SE2, fase 2).

Finalmente, um estudante reconhece que o software é útil, já que é utilizado por várias pessoas:

> *"(...) ele tem uma aceitação muito grande na... pelas pessoas que utilizam Latex, então muito provavelmente, ele tá concorrendo de frente com outras aplicações de grande porte"* (SE1, fase 2).

Proximidade das atividades da indústria

Quando questionados sobre a proximidade das atividades de manutenção executadas nos projetos práticos com as atividades que podem ser solicitadas no mercado de trabalho, cinco estudantes (56%) afirmaram que são próximas, três (33%) disseram que não saberiam dizer e um estudante (11%), que não são próximas. Convém ressaltar que dentre os

estudantes que afirmaram que as atividades são próximas, dois não tinham experiência na indústria de software. Todos que afirmaram que não saberiam dizer, não tinham experiência na indústria. E por fim, o único estudante que afirmou que as atividades não seriam próximas também não tinha experiência no mercado de trabalho. Todos os estudantes que responderam ao questionário e tinham alguma experiência na indústria afirmaram que as atividades executadas eram similares às solicitadas nas empresas.

A análise qualitativa identificou dois aspectos relacionados à proximidade das atividades da indústria: 'dar continuidade em um projeto existente' e 'adaptação ao processo ensino-aprendizagem' (Figura 6.3).

No primeiro caso, um estudante relatou que durante as disciplinas do curso, os estudantes aprendem a fazer o software do início, porém, na sua única experiência participando de um projeto real, este percebeu que, na verdade, precisaria dar continuidade em um projeto existente.

> *"(...) geralmente aqui, a gente pega um projeto no inicio. Tem aqui a disciplina, o componente curricular, ele vai pedir pra você fazer um software, então, você geralmente você pega do inicio. Quando eu fui pra iniciação cientifica, eu tive um projeto já grande pra fazer a modificação (...)''* (SE8).

Outro estudante, com bastante experiência no mercado de trabalho demonstra sua insatisfação com essa situação, inclusive ressaltando que, para a indústria, é importante que o estudante aprenda a aperfeiçoar algo que já existe.

> *"(...) nosso curso ele é muito... você desenvolve, reinventa a roda, você fazer uma coisa que já existe, ao invés de você aprender prática de como melhorar, como otimizar... isso aí pra mim, acho que não é tão viável pra quando você vai pro mercado de trabalho''* (SE6).

Finalmente, um estudante destacou a importância do que foi estudado para o mercado de trabalho, particularmente o de como proceder em determinada situação.

> *"Eu vi muita coisa legal nessa disciplina que facilita a vida quando você for para o mercado de trabalho. Pelo menos, se não ferramenta, a forma de pensar: como é que eu vou agir diante desse problema, como eu encaro essa situação a partir de agora. Então... Eu acho que esse conteúdo, ele foi bem interessante, eu acho que ele é bem a cara do mercado do trabalho (...)"* (SE1, fase 3).

Quanto à adaptação do projeto ao processo de ensino-aprendizagem, os estudantes perceberam que as atividades solicitadas exigiam a aplicação do processo de manutenção. Porém, as modificações em si que precisavam ser executadas não eram tão complexas quanto as que precisam ser executadas em projetos nas empresas.

> *"(...) as alterações que ele pediu, que ele pediu também, são alterações relativamente simples e que... não exige muito do... da pessoa assim de mexer em código... vai ser mais usar o que ele ensinou na... nas aulas expositivas''* (SE5).

> *"Achei a mesma... É a mesma. O que muda aí... só é de projeto, porque a questão de impacto, que vai ser diferente... pelo menos as modificações que nós fizemos, eu acho que o professor não... ele teve o cuidado de não pegar uma identificação tão crítica, que iria envolver uma... que ia envolver diversas classes... já no nosso caso, o impacto não era tão grande... a mudança, mas na empresa a gente pega... geralmente tem um impacto muito maior do que esses que teve no projeto, porque... o difícil foi mais você encontrar o... você encontrar onde você ia alterar no código, não você fazer a mudança'*' (SE6).

Ainda que reconhecendo que as modificações em si foram mais suaves, outro estudante destacou que o raciocínio a ser utilizado será o mesmo.

> *"Eu acho que são bem próximas, imagino que a gente trabalhou com exemplos mais... mais suaves em termos de complexidade, mas... é... eu acho que a carga que você tem que adquirir para tratar esses problemas mais simples, é o mesmo tipo de raciocínio, é o mesmo tipo de coisa, que você vai fazer no trabalho, talvez a implementação da mudança seja um pouco mais complicada, principalmente no mercado de trabalho. Mas acho que o raciocínio, toda a lógica, todo o trabalho feito pra dizer, vai ser feito aqui, vai impactar aqui, acho que é o mesmo (...) "* (SE1, fase 2).

Inclusive um dos estudantes sugeriu, para o aperfeiçoamento do curso, que alguma modificação mais complexa fosse solicitada.

> *"(...) uma coisa que eu não tiraria seria os projetos, talvez acrescentar um ou incluir uma certa dificuldade num terceiro ou aumentar a complexidade do que a gente vai fazer (...) "* (SE2, fase 2).

Convém dizer que a estratégia adotada pelo professor para adaptar a experiência com o projeto real ao processo de ensino-aprendizagem de evolução e manutenção era solicitar modificações cujas complexidades fossem crescentes.

> *"Eu procurei seguir uma... nas aulas e nos trabalhos mais ou menos o que Rajlich fez nas disciplinas dele... Do ponto de vista dos trabalhos, eu acho que a solução dele é interessante: trabalhar mudanças de complexidade crescente, uma mudança trivial que quase não tem impacto, uma mudança que tem um impacto e uma mudança que afeta os outros grupos que estão trabalhando. A gente conseguiu fazer apenas as duas primeiras (...)'*' (P1, fase 2).

Problemas ocorridos no decorrer da disciplina impediram a realização do último projeto, onde modificações mais complexas seriam necessárias. Na Subseção 6.2.7, mencionaremos estes problemas.

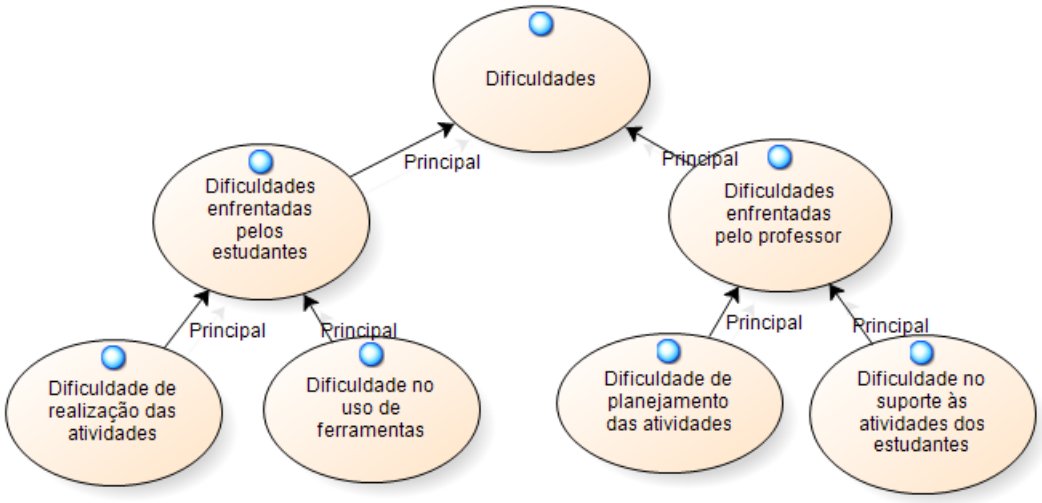

Figura 6.4 Dificuldades a serem enfrentadas.

6.2.3 Dificuldades

Na tentativa de proporcionar uma experiência de mundo real para os estudantes, é natural que o tema 'Dificuldades' (Figura 6.1) aflore em uma análise qualitativa. Nesse estudo, identificamos dificuldades relacionadas às atividades dos estudantes e do professor, a Figura 6.4 resume todas essas dificuldades.

Os estudantes encontraram dificuldades para a 'realização das atividades' e no 'uso de ferramentas' (Figura 6.4. Na realização das atividades, exemplos das dificuldades encontradas foram:

> *"(...) não saber onde é que tá determinada classe, determinado método (...)"* (SE5);
>
> *"(...) o conceito tava espalhado (...)"* (SE1, fase 2) e
>
> *"(...) a gente não tem plena certeza de que seria o lugar mais adequado pra fazer a mudança (...)"* (SE1, fase 2).

Essas dificuldades são provenientes dos problemas na estrutura de código, comuns em 'experiências do mundo real', como mencionado em 6.2.2. Mas também, que são próprias de quando é necessário dar manutenção em software desenvolvido por terceiros, conforme admitido por um estudante.

> *"(...) mas como fazia parte da disciplina saber... buscar... as funções do código e tal, refatorar... acho que pelo fato de não tá tão organizado, ajudou no quesito de você procurar utilizar esses conceitos"* (SE5).

Quanto ao uso de ferramentas, alguns estudantes relataram dificuldades no uso do sistema de controle de versões distribuído e admitiram, do mesmo modo que o professor, que um treinamento mais efetivo para o uso desse tipo de ferramenta seria adequado.

As dificuldades enfrentadas pelo professor são referentes ao 'planejamento das atividades' e ao 'suporte às atividades dos estudantes'. Essas dificuldades surgiram em virtude da falta de domínio pleno do mesmo sobre o JabRef. No que diz respeito ao planejamento, a falta de domínio dificultou a definição das atividades de manutenção a serem realizadas pelos estudantes, uma vez que estas atividades precisavam ter um nível de complexidade equivalente entre os grupos e crescente de uma atividade para outra, à medida que o conteúdo da disciplina era apresentado. O problema foi amplificado pelo pouco tempo disponível para o professor realizar esta tarefa. Para amenizar essa atividade, o professor contou com a ajuda de um estudante de mestrado que tinha experiência tanto profissionalmente quanto no uso de projetos de código aberto. Finalmente, no decorrer da disciplina, por não entender completamente o software, auxiliar os estudantes na solução dos problemas que surgiram, tornou-se uma tarefa complicada.

6.2.4 Associação Teoria e Prática

Conforme ilustrado na Figura 6.1, a 'Associação teoria e prática' foi outro tema que emergiu da análise dos dados. Identificamos códigos que retratam que 'a prática influencia a aprendizagem da teoria'; 'a teoria fundamenta a realização da prática' e que, 'teoria e prática são complementares' (ver Figura 6.5).

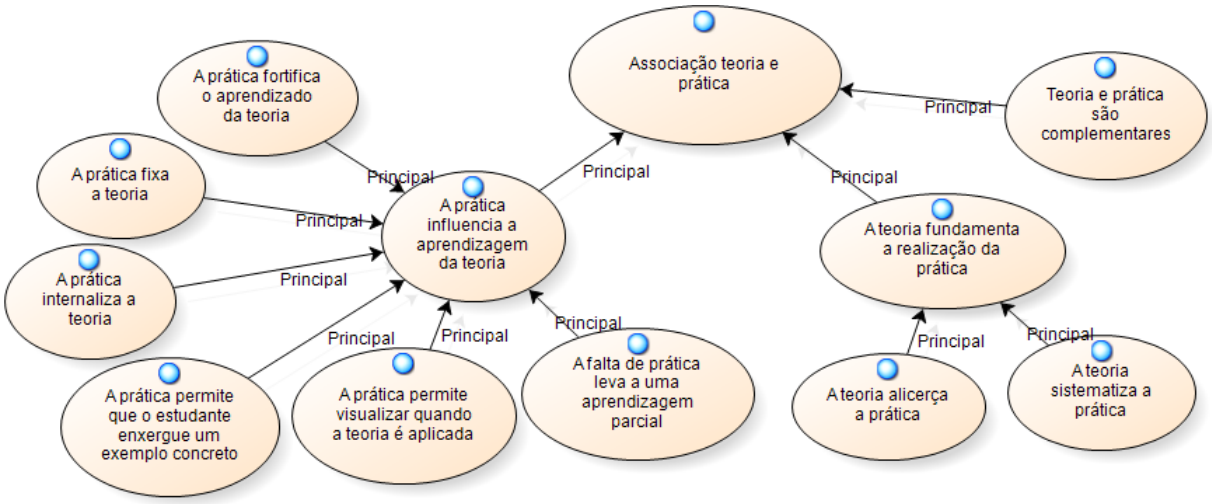

Figura 6.5 A associação entre teoria e prática para a aprendizagem.

A prática influencia a aprendizagem da teoria

Considerando 'a influência da prática para a aprendizagem da teoria', estudantes afirmaram que a realização de um projeto prático após a apresentação da teoria "fortifica" (SE1), "internaliza" (SE6 e SE8) o conteúdo visto. Um estudante acrescentou que exemplos

práticos, principalmente utilizando softwares reais, possibilita que o estudante enxergue realmente algo concreto.

> *"Você tá vendo coisa que existe. Não é: 'existe a ferramenta tal, que faz análise do software tal e ela diz que se acontecer isso, o software tá ruim, se acontecer isso, o software tá bom'. Mas isso é muito teórico, por mais que seja interessante, é teórico. Aí você fala: 'não, vamos ver aqui o software tal, esse software aqui, ele era assim'. O professor mostrou uns casos desse na aula: 'o software é assim. Ele começou assim (...)' "* (SE1, fase 3).

Outro estudante declarou o quanto a prática é importante para a fixação (retenção) do assunto, diferentemente de quando ocorre apenas a exposição de conteúdo e a avaliação da aprendizagem através de uma prova.

> *"(...) aquela questão da prática, pra mim foi essencial, porque se ele ficasse só falando ali, a gente com certeza ia chegar e depois de dois, três dias... ou se fosse fazer uma prova, por exemplo, tá, tá escreva, tá, como é a topologia... Depois da prova ali, vai embora, você aprendeu ali, mas ficou só escrito na prova e quando você torna aquilo prático, ali fixa (...)"* (SE8).

Outro estudante revelou que o projeto prático permitiu visualizar quando os conceitos apresentados e discutidos durante as aulas deveriam ser aplicados.

> *"O projeto prático eu achei ótimo, porque você aplicava tudo o que o professor... tudo o que foi discutido na aula, você na hora que você tava resolvendo, analisando, buscando... aquele conceito que professor falou: 'ah... ele se enquadra nesse, nessa questão aí' (...)"* (SE6).

Em virtude de não terem utilizado muitas das ferramentas apresentadas durante os seminários, um estudante mencionou que quando a prática não está associada, a aprendizagem não é completa.

> *"(...) você vê o contexto, você vê a ferramenta, mas você não utilizou a ferramenta, então você parcialmente sabe que aquilo existe, mas não sabe lidar muito bem com aquilo"* (SE1, fase 3).

A teoria fundamenta a realização da prática

No que diz respeito 'a teoria fundamentar a realização da prática', podemos dizer que 'a teoria alicerça a execução da prática', porque permite o estudante compreender o que está fazendo, porque está fazendo:

> *"(...) a exploração do conteúdo antes de ver esses projetos ajudou na realização dos projetos"* (SE2, fase 2).

> *"(...) eu gosto muito de teoria assim, eu não gosto de fazer nada sem tá embasado em uma teoria, que eu saiba o que eu tô fazendo, então, acho que as aulas, ela trouxe a questão do que é, do que é a etapa (...)"* (SE1, fase 2).

Em outro momento, outro estudante destacou que apesar de parecer redundante, a teoria pode fornecer uma nova forma de tratar um determinado problema.

> *"A maioria das coisas, o professor mesmo sempre falava: 'tem coisas que a gente vai ouvir aqui que vai parecer redundante'. Só que... uma coisa é você ter esse conhecimento... você tá falando isso aí, mas eu já sabia disso... Mas você não sabia, por exemplo, de que formas você poderia lidar com aquilo..."* (SE2, fase 3).

Ou ainda porque 'a teoria sistematiza a realização da prática':

> *"(...)às vezes o professor falava alguma coisa que eu sofri no estágio, mas eu não tinha noção de que daquela forma, eu poderia ter resolvido, poderia ter solucionado de uma maneira melhor. Até mesmo, meu projeto de iniciação científica... uma bagunça total, eu precisava refatorar, e existe todo um conceito por trás disso, que eu acha que era simplesmente, chegar no código, trocar um nome e tava bonito. Não, tem toda uma teoria por trás disso, que pode ajudar inclusive a me dizer quando é necessário... fazer essa refatoração antes que o código... vire uma bola de neve e eu não consiga solucionar"* (SE2, fase 3).

> *"(...) basicamente, o que o professor falou, já utilizei muito daquelas ferramentas, mas eu usava sem ter noção de que já existia toda essa... todo esse padrão"* (SE6).

Teoria e prática são complementares

Concluindo o tema 'Associação teoria e prática', como 'a prática influencia a aprendizagem da teoria' e 'a teoria fundamenta a realização da prática', enxergar 'teoria e prática como complementares' torna-se consequência. Alguns extratos denotam essa visão:

> *"Eu acho que não dá pra separar um do outro não, porque quando a gente vai fazer a parte prática, a gente precisa de algum conhecimento prévio, ou se a gente não tem, a gente vai ter que parar pra ir pesquisar, pra estudar..."* (SE6).

> *"(...) uma disse mais ou menos o que era e a outra contribuiu para o fortalecimento"* (SE1, fase2).

> *"(...) aqueles que foram a fundo nas práticas... é... se sentiram mais... com aquela sensação de que sei fazer... mas esse aprendizado não haveria se não houvesse as aulas de introdução ao processo de mudanças incrementais, então, essas duas coisas foram complementares"* (P1, fase 3).

Observamos ainda que durante as atividades práticas no laboratório, os tópicos teóricos eram lembrados e aplicados, enquanto que nas aulas teóricas, exemplos eram dados a partir do software sendo utilizado nas práticas, isto é, o JabRef.

6.2.5 Motivação

Ao estudarmos as consequências de determinada proposta didática, é importante investigar a motivação dos estudantes considerando essa proposta, isto é, como a proposta pode estimular ou não os estudantes para a aprendizagem. Assim, é legítimo o surgimento do tema 'Motivação' a partir da análise dos dados. A Figura 6.6 exibe as condições encontradas que influenciaram a motivação dos estudantes nesse estudo.

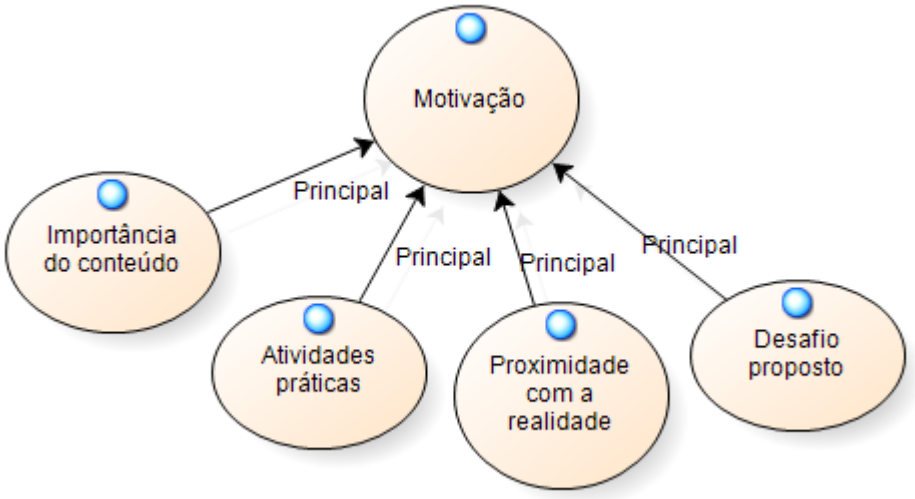

Figura 6.6 Condições para a motivação do estudante.

Identificamos que quando o estudante compreende a 'importância do conteúdo' para o seu futuro profissional, ele sente-se mais motivado para enfrentar os desafios estabelecidos.

> *"Isso é uma questão de desafio né? é... desde o princípio, eu sempre imaginei essa disciplina como algo que fosse remeter ao mercado de trabalho, que é bem a cara do mercado de trabalho, a evolução e manutenção a depender da área que você for '' (...) "minha motivação era legal, pensando nesses... isso, espelhado no futuro"* (SE1, fase 2).

A realização de 'atividades práticas' corresponde a outra situação que motiva os estudantes. O extrato a seguir expressa a motivação do estudante para a aula prática.

> *"As atividades práticas... sempre gostei mais da parte prática. Eu já chegava feliz para tentar desenvolver, tentar mexer no programa (...)"* (SE6).

Esse mesmo estudante, em outro momento, comparou sua experiência em outra universidade com a atual, salientando que falta mais prática no curso atual.

> *"(...) você tinha aquela aula teórica, meio maçante, mas após a aula sempre tinha a parte complementar, que seria a parte prática e falta muito aqui na universidade (...)"* (SE6).

Outro estudante relatou a preferência por aulas práticas (realizadas no laboratório), porém reconheceu a importância do conteúdo teórico ter sido apresentado previamente para a realização das atividades.

> *"Eu gosto de atividades em que a gente participa mais, por exemplo, eu achei mais interessante as aulas de laboratório. Sempre acho as aulas de laboratório mais interessantes e essa disciplina em particular trouxe dois projetos." (...) "eu achei mais interessante isso do que só assistir a aula, porém a exploração do conteúdo antes de ver esses projetos ajudou na realização dos projetos"* (SE2, fase 2).

Por essas e outras situações que serão apresentadas na Subseção (6.2.7) relacionamos 'associação teoria e prática' com 'motivação' (ver Figura 6.1).

Quando a prática envolve algo real, o conteúdo estudado é aplicado à realidade e torna-se, portanto, mais interessante. Um dos estudantes entrevistados critica exatamente que ao longo do curso de computação tudo é muito abstrato, faltando maior 'proximidade com a realidade':

> *"Porque eu acho que o problema da computação, mesmo, é muita coisa abstrata, né? Então, quando você consegue juntar a realidade (...) porque aqui é muito acadêmico, né. (...) É diferente o acadêmico da realidade da indústria... da empresa (...)"* (SE8).

Por essa razão, estabelecemos o relacionamento entre 'Experiência de mundo real' e 'Motivação' (ver Figura 6.1).

O professor indiretamente 'propõe um desafio' ao estudante, ao solicitar que o mesmo realize uma atividade prática. A forma que o estudante encara esse desafio poderá gerar motivação ou, pelo contrário, desmotivá-lo. A aproximação com a realidade torna o impacto desse desafio ainda maior. Neste estudo de caso, detectamos as duas situações. Como exemplos de motivação, apresentamos dois extratos. No primeiro, o estudante ressaltou o desafio representado pela própria realização das atividades no projeto real.

> *"(...) tinha esse quê, essa questão de desafio, quando você tava tentando localizar, tentando modificar, tinha essa questão de você vencer o problema, tinha essa questão, então acho que minha motivação era legal"* (SE1, fase 2).

No segundo exemplo, outro estudante denotou entusiasmo com a experiência proporcionada pelo emprego do projeto de código aberto[4].

> *"Pra gente foi um desafio mexer no código fonte de um projeto open source, coisa que eu nunca tinha feito"* (SE2, fase 2).

Porém em outro momento da entrevista, o mesmo estudante apresentou algum desencorajamento, dada a complexidade das atividades. Mas, ao final, achou positivo.

[4]Por isso acrescentamos o relacionamento entre 'Projeto de código aberto' e 'Motivação' (ver Figura 6.1).

> *"Eu acho que o professor pegou pesado em alguns, em alguns dos... das modificações que a gente teve que fazer... até sofri particularmente com isso. Ele teve que mudar o projeto da gente, mas isso também foi uma coisa positiva (...)"* (SE2, fase 2).

Tal situação representa um exemplo onde as dificuldades a serem enfrentadas podem acarretar em uma situação desmotivante.

Como resultado do questionário aplicado, a Figura 6.7 representa o grau em que o tamanho e a complexidade do código do JabRef influenciaram a motivação dos estudantes para a realização das atividades. A escala variou de zero, indicando a perda de motivação, a quatro, indicando que o tamanho e a complexidade do código foram motivantes e desafiadores. Vemos que para alguns estudantes, as dificuldades proporcionadas pelo software foram desmotivantes, mas nenhum o apontou como totalmente desmotivante. Outros enxergaram as mesmas dificuldades, como um desafio de algum modo motivador.

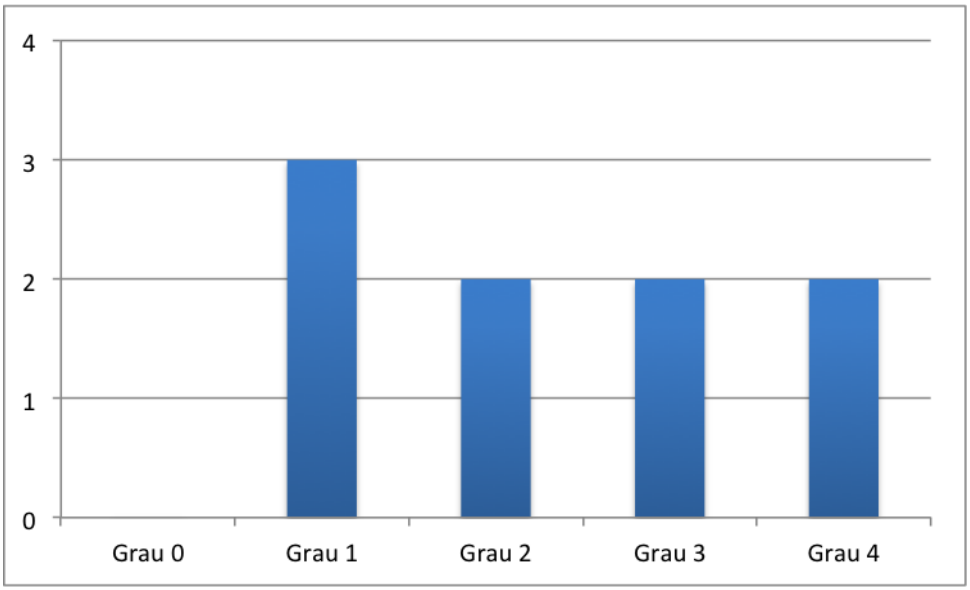

Figura 6.7 Grau de motivação para a realização das atividades considerando o tamanho e a complexidade do JabRef.

Em virtude desses achados, estabelecemos a relação entre 'Dificuldades' e 'Motivação' mostrada na Figura 6.1.

6.2.6 Aprendizagem

Aprendizagem é o principal resultado esperado, quando o foco do estudo é investigar a utilização de uma nova abordagem ou recurso para o ensino (ver Figura 6.1). Usualmente a aprendizagem está associada à aprendizagem de 'conceitos', ao desenvolvimento de 'habilidades' e a 'mudanças de atitudes'. Contudo, em nossa análise detectamos que a experiência profissional proporcionada também representa um tipo de aprendizagem

importante em longo prazo. Ademais, capturamos circunstâncias que colaboraram para que a aprendizagem ocorresse e situações que demonstram que de fato ela ocorreu. A Figura 6.8 detalha estas categorias.

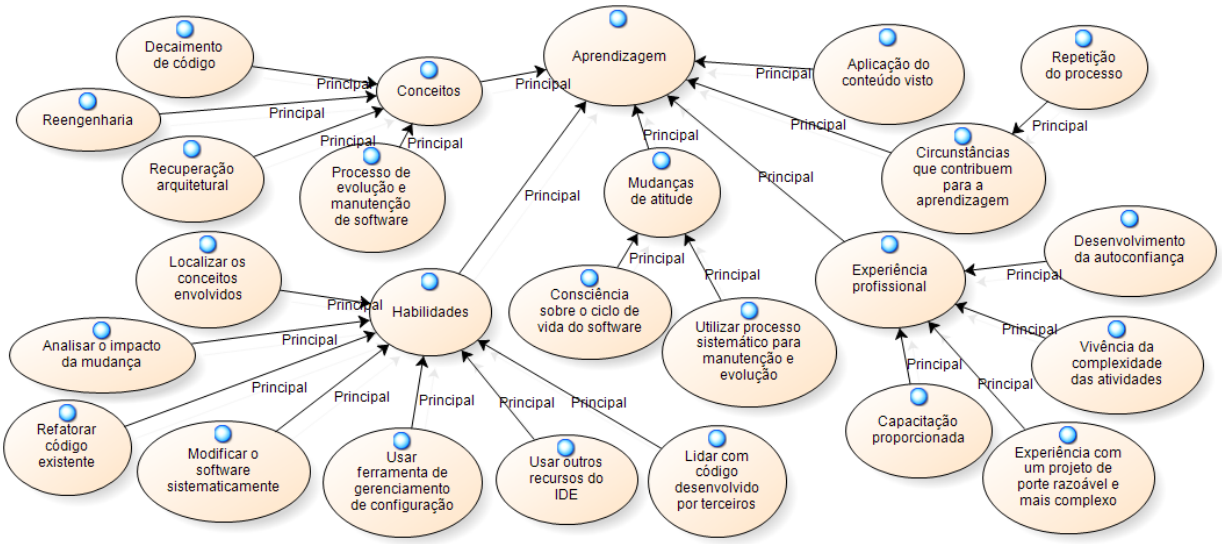

Figura 6.8 Aprendizagem proporcionada pela adoção de OSP no estudo de caso sobre evolução e manutenção de software (EC1).

Desenvolvimento de habilidades e aprendizagem de conceitos

Considerando a aprendizagem em curto prazo, quando questionados sobre qual o conteúdo assimilado ou práticas que eram capazes de aplicar, os estudantes relataram que eram capazes de: (i) "localizar os conceitos envolvidos para a modificação do software" (SE1, SE2, SE6, SE7, SE8); (ii) "analisar o impacto da mudança sistematicamente" (SE1, SE2, SE7, SE8); (iii) "refatorar o código existente" (SE2 e SE6); (iv) "modificar o software sistematicamente" (SE1, SE2, SE8) e (v) usar ferramentas, em especial, a "ferramenta de gerenciamento de configuração distribuída" (SE1 e SE6) e "outros recursos do IDE" (*Interface Development Environment*) (SE2) utilizados pela comunidade do projeto de código aberto.

Segundo o professor, principalmente considerando a prática de localização de conceitos, os estudantes:

> *"(...) melhoraram as práticas que eles tinham anteriormente, estão fazendo de maneira mais sistemática, fazem isso sobre um sistema grande(..)"* (P1, fase 2).

Quanto aos conceitos, os estudantes mencionaram: "decaimento de código" (SE1 e SE6), "recuperação arquitetural" (SE2) e "reengenharia" (SE2). Sendo que para a discussão desses tópicos, outros projetos de código aberto foram utilizados como exemplo, tanto durante as aulas expositivas, quanto ao longo das apresentações dos seminários. Os softwares utilizados foram: Eclipse, Netbeans, Spring e FindBugs.

> *"O professor mostrou um caso desse na aula: 'o software é assim, ele começou assim, a gente fazia um recolhimento semanal do repositório e era assim, começou assim e oh, a partir daqui... oh, percebam, a partir daqui ele começou a erodir e foi evoluindo, depois de um ano, olha como era, olha como ficou'. (...) A gente viu Eclipse, a gente viu NetBeans, a gente viu Spring, a gente viu é... outros que ele mostrou em sala"* (SE1, fase 3).

> *"No seminário de recuperação arquitetural ou foi no de reengenharia (...) a gente mostrou a arquitetura. Eles usaram o Structure 101 e na ferramenta, eles têm vários exemplos. A gente mostrava o FindBugs como exemplo de software é super complexo e já erodiu, né? E a gente mostrava o Spring como o exemplo de software muito bem desenhado... com o acoplamento bem controlado"* (P1, fase 3).

No que concerne à 'capacitação proporcionada' com relação ao processo geral de evolução e manutenção de software, de acordo com a percepção do professor, os estudantes conhecem o processo, sabem o que deve ser feito, porém ainda não se sentem totalmente capacitados para a realização das tarefas.

> *"Porque na verdade, eu acho que eles se sentem, depois de ter passado por essas práticas é: eu entendo o que faz uma pessoa quando faz evolução e manutenção de software... e eu sei fazer um pouquinho... eu não estou perdido nisso... eu sei que quando eu tiver que fazer... eu entendo do que se trata, mas eu não sei fazer ainda com proficiência, eu não me sinto capaz de fazer isso com certeza"* (P1, fase 3).

Dois estudantes confirmaram essa percepção, todavia um deles admite ser capaz de realizar as tarefas de maneira aceitável.

> *"Eu acho que de maneira geral, eu não cheguei a dominar todos os tópicos, mas eu tenho uma visão geral do que seria a análise e manutenção do código"* (SE6).

> *"Olha eu acho que eu consigo... conseguiria tratar... eu acho que conseguiria manipular bem é... o controle de versão distribuído para a aquisição do projeto, conseguiria fazer a análise da mudança que teria que ser feito, a análise de impacto pra mudança... acho que... quase que um passeio pelo processo todo de evolução e manutenção... eu acho que seria mais ou menos isso. Talvez não... não perfeitamente, mas eu consigo passear pelo processo todo de uma forma... aceitável"* (SE1, fase 3).

Finalmente, questionados sobre a aprendizagem geral da disciplina, 78% dos respondentes consideraram a aprendizagem como satisfatória e 22% como boa. Confirmando o pensamento de que a capacitação ocorreu de forma satisfatória.

Além do desenvolvimento das capacidades já mencionadas, observamos o desenvolvimento da habilidade de 'lidar com código desenvolvido por terceiros'. Conforme relatado

por um estudante e reportado em 6.2.1), trabalhar com software desenvolvido por outras pessoas possibilita que o estudante aprenda a lidar com diferentes estilos de programação, por essa razão, acrescentamos a ligação entre 'projeto de código aberto' e 'aprendizagem' na Figura 6.1.

Experiência profissional

Além da 'capacitação proporcionada', como pode ser visto na Figura 6.8, detectamos outras circunstâncias que contribuíram para a formação em longo prazo, dada a experiência profissional possibilitada. São elas: (i) 'experiência com um projeto de porte razoável e mais complexo'; (ii) 'vivência da complexidade das atividades' e (iii) 'desenvolvimento da autoconfiança'.

Quanto à 'experiência com um projeto de porte razoável e mais complexo', um estudante ressaltou a importância de lidar com um projeto com essas características, diferentemente do que ocorre ao longo das disciplinas, onde os estudantes trabalham apenas em projetos pequenos. O estudante acrescenta a falta desse tipo de experiência também entre seus colegas.

> *"(...) você mexer num sistema open source... Porque aí, a gente... a gente não vai sair da universidade apenas com aquela visão de só fazer projetos pequenos, utilizar sistemas pequenos, entendeu? Você vai começar a utilizar esses sistemas maiores, complexidades maiores, problemas maiores, entendeu? Trabalhar com uma equipe maior, a maioria das pessoas, pelo menos, que eu pude perceber, meus colegas que eu conversei, nenhum deles tinha trabalhado com sistemas maiores, entendeu? Só os pequenos sistemas daqui da universidade"* (SE7).

Questionados sobre experiências anteriores com softwares em outras disciplinas do curso, 89% dos respondentes relataram que não tinham trabalhado com software de tamanho igual ou superior ao JabRef, enquanto que 75% responderam que não tinham trabalhado com software de complexidade igual ou superior ao JabRef.

A falta de experiência dos estudantes com projetos maiores foi ainda evidenciada na observação da aula ocorrida no dia 12/12: *"O professor questiona o tamanho do maior projeto que os estudantes já manipularam, a maioria responde que foi o JabRef"*.

Considerando a 'vivência da complexidade das atividades', um estudante comentou sobre a complexidade das atividades do projeto repassado pelo professor para sua equipe, ressaltando o quanto foi positivo vivenciar na prática um problema complexo, uma vez que isso vai ocorrer futuramente.

> *"(...) isso também foi uma coisa positiva, porque se não fosse assim, eu não saberia quão complexo seria fazer uma localização de um código que eu não conheço ou fazer uma análise de impacto de um código desconhecido. Mas no geral, a complexidade sempre vai fazer parte, né, sempre vai ter alguma parte do projeto ou do sistema que não vai ser possível ter conhecimento ao todo, então foi produtivo"* (SE2, fase 2).

O professor reafirmou essa ideia, realçando que é muito bom para os estudantes vivenciarem ainda na universidade, desafios semelhantes aos que precisarão enfrentar na vida profissional.

> *"(...) lidar com a complexidade, trabalhar com o projeto real de... de boa complexidade, então isso é um desafio, e eles tiveram que aprender e às vezes reclamavam: poxa é muito difícil, eu tenho que entender. Mas é isso que você vai ter que fazer na sua vida profissional, então é muito bom você tá passando por essa oportunidade de enfrentar um problema de razoável complexidade (...)"* (P1, fase 2).

Em virtude dessas circunstâncias, incluímos a relação entre 'experiência de mundo real' com 'aprendizagem' na Figura 6.1.

Por outro lado, o tamanho do projeto e a complexidade das atividades a serem executadas podem gerar insegurança nos estudantes. Contudo, um dos estudantes destacou a experiência como semelhante ao sentimento vivido em seu primeiro estágio, servindo para mostrar que o estudante é capaz.

> *"(...) quando você pegar um projeto com a quantidade de linhas de códigos que tem ali você acaba, inicialmente intimidado... como é que eu vou trabalhar em cima de uma coisa que... que o medo que nós temos é esse quando a gente sai da universidade. No primeiro estágio que eu fiz... será que eu vou dar conta?"* (SE6).

Outro estudante complementou de modo mais explícito que tal sentimento de insegurança foi seguido por um sentimento de realização, por ter conseguido executar as atividades.

> *"(...) eu já conversei com outros colegas e... você acha que você não sabe muita coisa, você tem essa impressão de que você viu muita coisa e que você não sabe nada, mas quando você se depara com... por exemplo, um projeto desse, completamente novo, você não conhecia o domínio, que você tem que fazer uma mudança nesse projeto... e você consegue realizar a mudança, você percebe que você... você tem essa capacidade de realizar o trabalho (...)"* (SE1, fase 2).

E acrescentou que o 'desenvolvimento da autoconfiança' foi a principal contribuição do projeto para o seu futuro profissional.

> *"Eu acho que na questão da confiança mesmo, que ...essa questão de reafirmar que você é capaz, que você é... você consegue fazer. Veio contribuir pra essa autoconfiança"* (SE1, fase 2).

Mudanças de atitudes

No que diz respeito às mudanças de atitudes ocasionadas pela participação no projeto, podemos frisar a 'consciência sobre o ciclo de vida do software' e a necessidade de 'utilizar processo sistemático para evolução e manutenção' do software (ver Figura 6.8).

Através do próximo extrato, visualizamos a mudança de comportamento do estudante que já possuía alguma experiência profissional. A partir da disciplina e dos projetos utilizados, ele conseguiu enxergar o ciclo de vida do software, de modo que, ciente que os softwares precisarão ser modificados no futuro e que as decisões feitas durante o seu desenvolvimento irão impactar nessas modificações futuras, é preciso pensar na evolução do software, ainda durante o seu desenvolvimento.

> *"Com essa disciplina eu tive... eu acho que eu tive maior consciência do que é desenvolvimento de software, porque antes eu fazia as coisas, mas não sabia do impacto que pode ter no futuro, não tinha essa questão de que um software poderia morrer, essa questão da degradação de acordo com... às vezes as coisas simples que a gente faz no desenvolvimento pode no futuro... no futuro, ter esse impacto tão grande no sistema, acho que essa consciência... acho que é o legado dessa disciplina, que pra os próximos trabalhos e pro futuro (...)"* (SE6).

Outros dois estudantes também com experiência em estágios demonstraram estar mais conscientes sobre a relevância de executar as atividades de evolução e manutenção de software de modo mais sistemático.

> *"Ter mais disciplina em relação a essas questões de evolução e manutenção... não fazer uma coisa assim mais intuitiva, ter mais, é... organização na hora de realizar uma mudança, de adicionar uma coisa, registrar melhor o que foi feito, o que foi alterado... acho que com a disciplina, assim... abriu um pouco os meus olhos em relação a isso, que antes eu fazia, o que desse na cabeça eu fazia, mas não é bem assim, tem que ter uma organização, uma operação melhor"* (SE5).

> *"Eu já tenho uma visão diferente de como eu vou fazer essa estruturação e descobrir que pontos vão afetar o código, ou o que eu preciso mudar, como eu consigo utilizar das ferramentas que eu tenho disponível pra chegar naquela mudança, naquele problema e resolver. Então, acho que essa parte foi fundamental"* (SE2, fase 3).

Aplicação do conteúdo visto

Como comprovação da importância do conteúdo estudado e da aprendizagem ocorrida, principalmente sob a perspectiva da aprendizagem em longo prazo, os estudantes relataram que já estão aplicando o conhecimento adquirido em outras oportunidades, no decorrer de suas experiências profissionais.

> *"Muitas das coisas eu já apliquei lá na empresa onde a gente trabalha, algumas práticas de você melhorar a organização do seu código, usar sistema de*

> *versionamento, essas práticas que tem... algumas técnicas de você... como foi na aula... nas aulas práticas... as técnicas de você achar onde é que vai fazer a mudança no seu projeto (...)"* (SE6).

> *"(...) eu estou usando essa parte de localização de conceito, análise de impacto, código por código assim (...)"* (SE2, fase 3).

Por fim, ainda pensando em uma aprendizagem a longo prazo, obtivemos como resultado quantitativo que 100% dos respondentes do questionário acreditam que os conhecimentos adquiridos ao longo da disciplina podem melhorar o seu desempenho no futuro.

Circunstâncias que contribuem para a aprendizagem
Para encerrar os achados sobre a aprendizagem, podemos dizer que muitos fatores contribuem para a sua ocorrência. Ao longo dessa seção, destacamos as contribuições do 'uso do projeto de código aberto' e da 'experiência de mundo real'. Na Subseção 6.2.4 apresentamos como a teoria e a prática são importantes para a aprendizagem. Na Subseção 6.2.5, visualizamos condições positivas e negativas de motivação dos estudantes, proporcionadas pela proposta didática empregada, que podem vir a influenciar a aprendizagem dos estudantes. Particularmente, o extrato a seguir demonstra a motivação do estudante para a realização de projetos práticos e consequentemente, gera um 'terreno' propício à aprendizagem.

> *"(...) depois passava aquele projetinho pra gente tentar aplicar todo o conhecimento e internalizar todo o conceito. Eu acho que esse... eu acho que deveria partir mais pra isso... pra ter mais práticas no laboratório, que eu acho que esse conhecimento é o que... que motiva mais você tá, tá indo pra disciplina e aprendendo novas coisas"* (SE6).

Além desses fatores, cujos relacionamentos inclusive estão ilustrados na Figura 6.1, dois estudantes apontaram que a repetição de aplicação do processo na prática (localização de conceitos, análise de impacto e efetuação da mudança) fez com que as atividades fossem bem assimiladas (ver Figura 6.8).

> *"(...) foi muito batido, né? Viu de diversas formas e... e sempre passava por ele, então... ele é o comecinho do processo todo. Pra onde você vai, você faz... Talvez por isso, pela repetição"* (SE1, fase 3).

> *"A parte de... de você procurar... uma coisa que foi muito... muito batido mesmo, foi a parte de você identificar as... as classes que seriam impactadas em uma mudança de software (...)"* (SE7).

Finalmente, sob a perspectiva quantitativa, questionados inicialmente apenas sobre as atividades práticas, 56% dos estudantes afirmaram que o projeto prático teve uma boa contribuição para a aprendizagem, enquanto que o restante, 44%, admitiu que o projeto prático contribuiu bastante para a aprendizagem. Consequentemente todos os estudantes

valorizaram a contribuição do mesmo para a aprendizagem. A Figura 6.9 ilustra a percepção dos estudantes quanto à contribuição dos métodos didáticos empregados ao longo da disciplina, para a sua aprendizagem. Os estudantes escolheram a sequência que representa o método que mais contribuiu para o que menos contribuiu. Desse modo, 56% dos respondentes reconheceram as atividades práticas, aulas expositivas e seminários, como a sequência que mais contribuiu, 22% consideraram a sequência atividades práticas, seminários e aulas expositivas, 11% assumiram a sequência aula expositiva, atividades práticas e seminários e os últimos 11%, admitiram a sequência seminários, aulas expositiva e atividades práticas. Por conseguinte, 78% consideraram as atividades práticas como as mais importantes para a sua aprendizagem e apenas 11% consideraram-nas como as menos importantes.

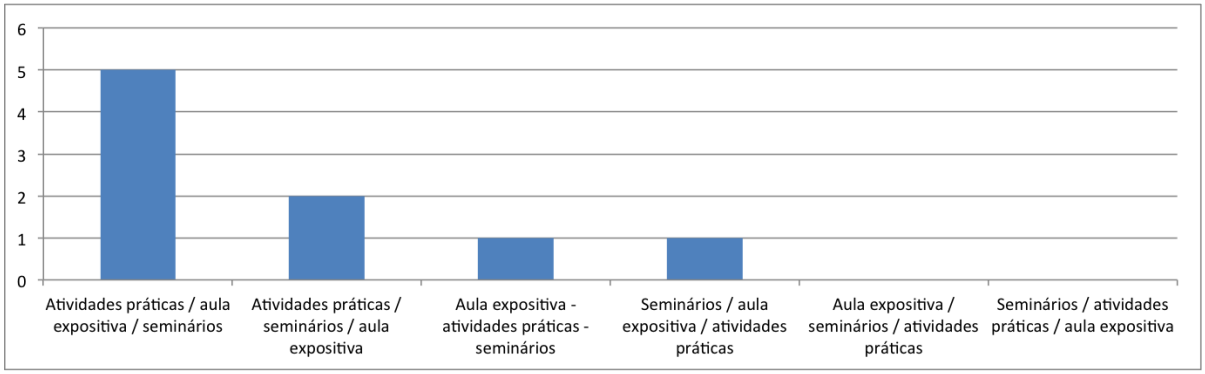

Figura 6.9 Sequência de contribuição do método didático para a aprendizagem.

6.2.7 Satisfação

Ao realizarmos a análise qualitativa, outro tema relevante encontrado foi 'Satisfação'. Motivação e satisfação são sentimentos intimamente ligados e importantes quando uma nova proposta didática está sendo proposta. Constatamos alguns sentimentos e aspectos, positivos e negativos, com relação à disciplina e às atividades práticas realizadas, conforme podemos visualizar na Figura 6.10. Entre os sentimentos e aspectos positivos, percebemos 'contentamento', 'atendimento das expectativas', 'sentimento de realização' e a possibilidade de 'continuidade'. No lado contrário, observamos exemplos de algumas 'insatisfações'. A seguir, mostraremos as expressões dos estudantes e professor que nos levaram a categorizar essas situações.

Contentamento
Extraímos alguns trechos que indicam o 'contentamento' dos estudantes com a disciplina. Cabe ressaltar que nesses trechos, podemos notar que o contentamento está relacionado à 'aprendizagem', à 'associação da teoria com a prática', ou muitas vezes, assinala que existe alguma 'motivação' incorporada. Por essas razões, estabelecemos os relacionamentos desses temas com o tema 'satisfação', presentes na Figura 6.1. Destacamos exemplos dessas situações nas entrevistas realizadas com um dos estudantes. No primeiro extrato,

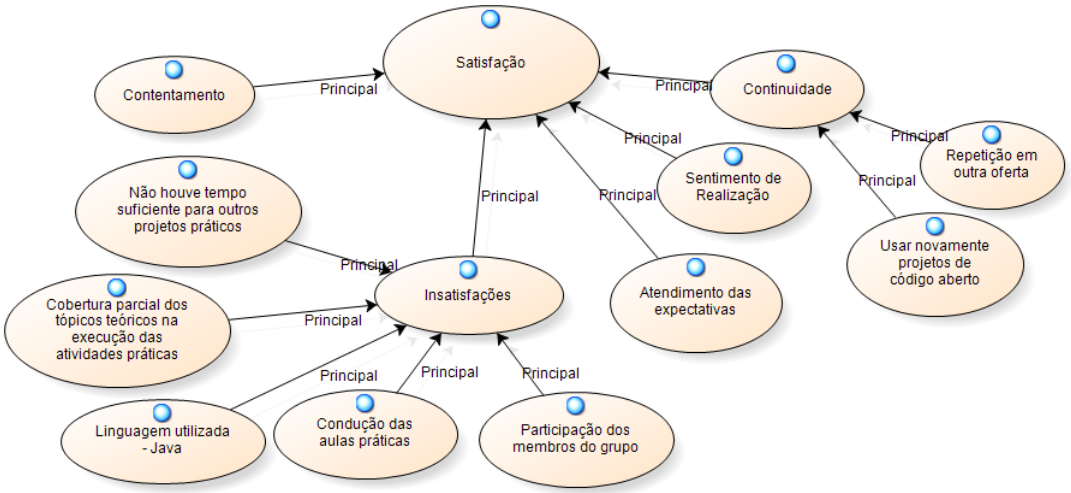

Figura 6.10 Satisfação com a disciplina e projetos práticos.

enxergamos a associação entre teoria e prática, motivação e contentamento, ressaltando ainda a "vivência" de "evoluir um software" que "nunca tinha mexido", o que podemos considerar como uma aprendizagem para longo prazo.

> *"Eu achei muito interessante, muito mesmo porque é... era a oportunidade de confrontar aquilo que a gente tava vendo em teoria, com os exemplos e... com exemplos teóricos, com os exemplos práticos que a gente via do cotidiano e vivenciar essa questão de, de... é...alterar ou dar manutenção, evoluir um software que a gente nunca tinha, nunca tinha mexido, não tinha conhecimento do, do... aquilo como um todo, então isso foi muito legal, muito legal mesmo"* (SE1, fase 2).

Nesse outro trecho, visualizamos o vínculo entre satisfação e motivação, com "visões novas" da própria engenharia de software.

> *"(...) quanto ao conteúdo eu achei que foi bem proveitoso. Pra mim foram... por mais que você dá uma pescada... em coisas que você já tinha conhecimento... de engenharia de software, sempre tinha pitadas de coisas novas ou de visões novas... de tudo o que ele estava trazendo, então pra mim, isso foi bem... interessante"* (SE1, fase 3).

No último extrato considerando a satisfação desse estudante com a disciplina, ressaltamos o contentamento com a aprendizagem proporcionada.

> *"Então eu acho que além de todas as ferramentas, acho que a visão que você tem do processo de evolução... do dia-a-dia, do quê fazer, acho que esse é o grande barato"* (SE1, fase 3).

Outros estudantes também demonstraram o seu contentamento com relação à disciplina. No exemplo a seguir, o estudante está feliz com um aprendizado inesperado.

> *"(...) eu aprendi sobre Eclipse!! Eu achava que Eclipse era só... uma aba de mapa do projeto e a tela de código. Tem muito mais que isso... Tem ferramentas que auxiliam no meu desenvolvimento, dentro dessa ferramenta... Por isso, eu achei que isso foi importante"* (SE2, fase 3) .

Em outro exemplo, um dos estudantes destacou o seu contentamento com a associação das aulas teóricas (expositivas) com a prática.

> *"A aula expositiva também foi legal, eu gostei. Ainda mais essa parte que juntou a aula expositiva com a prática. Se fosse só a aula expositiva, não seria legal... realmente ficaria muito... ficaria muito distante da pessoa aprender"* (SE8).

No que diz respeito ao contentamento apenas com o projeto prático, um dos estudantes em vários momentos demonstrou sua satisfação, inclusive destacando a sua relevância para seu futuro profissional:

> *"Adorei. Foi uma das melhores experiências..."* (SE2, fase 2).

> *"(...) eu achei bastante construtivo, é algo que certamente a gente vai utilizar bastante"* (SE2, fase 2).

> *"(...) toda aquela fase inicial que a gente utilizou no JabRef, que foi muito legal..."* (SE2, fase 3).

Sobre a utilização do JabRef, o estudante acrescentou o quanto foi interessante trabalhar com um projeto de código aberto, experiência não enfrentada anteriormente.

> *"(...) como a gente fez com JabRef. Foi uma experiência interessante. Como eu falei antes, eu nunca tinha tido essa curiosidade de mexer num código open source e eu já utilizei vários softwares open source, mas nunca tinha tido essa curiosidade de buscar e achei legal como as coisas funcionam"* (SE2, fase 3).

Outro estudante destacou a importância da utilização do JabRef para visualizar o impacto das mudanças.

> *"(...) praticando no JabRef ficou bem mais claro... a ideia... a ideia de você... conseguir medir esse impacto"* (SE7).

Sob a perspectiva do professor, ainda sobre o JabRef, ao mesmo tempo em que ficou satisfeito com a sua escolha, por representar domínio fácil de entender e apresentar problemas semelhantes aos existentes em softwares de médio porte (características comentadas

em 6.2.2), o professor relatou algumas dificuldades (ver 6.2.3) e o descontentamento com a ausência de um conjunto razoável de testes já implementados no software.

Após essas colocações específicas sobre o JabRef, verificamos que a utilização de projetos de código aberto podem também influenciar positiva e negativamente a satisfação dos envolvidos, sendo assim, acrescentamos o relacionamento entre 'projeto de código aberto' e 'satisfação' (ver Figura 6.1).

Insatisfações
Os envolvidos também reportaram algumas 'insatisfações' com relação à disciplina e ao projeto prático, conforme ilustrado na Figura 6.10 e descritos a seguir.

Cobertura parcial dos tópicos teóricos na execução das atividades práticas
O professor relatou que as atividades práticas não exploraram os tópicos teóricos em sua totalidade. Tópicos como testes e refatoração não foram abordados, bem como, a análise de impacto foi trabalhada insuficientemente.

> *"(...) o processo de mudança incremental envolve alguns aspectos como refatoração e testes que a gente não explorou e a gente achou de certa forma que faltou explorar esses aspectos, é... então, essencialmente os aspectos que foram mais trabalhados foram, na prática, né, localização de conceitos, análise de impacto, realização de uma mudança, propagação da mudança, né, pra classes vizinhas, e, o... a entrega da mudança e o envio pra um repositório, é... A localização de conceitos foi o que funcionou melhor. A questão da realização da mudança e propagação da mudança também funcionou bem porque eles tinham que fazer isso pra mudança realmente funcionar. A análise de impacto não foi explorada suficientemente, de maneira suficiente, é... não sei se eles tem a percepção da real importância dessa etapa. E a gente não explorou outras questões como testes, testes de regressão e refatoração que a gente poderia ter explorado, se a gente tivesse mais tempo e tivesse mais tempo pra escolher mudanças que exigissem essas... essas etapas"* (P1, fase 2).

O professor justificou a não cobertura do tópico de testes em virtude da ausência de testes implementados para as funcionalidades sendo modificadas. Os estudantes também identificaram que nem todos os passos do processo de mudança sistemática foram cobertos, porém não atribuíram grande gravidade ao mesmo.

> *"A gente não conseguiu ver tudo, né, porque a gente, eu acho... meu ponto de vista, a gente viu muito na prática, localização de conceito, análise de impacto e a tendência da mudança... perto da... perto da teoria que a gente viu em sala tinha, se eu não me engano... eu não sei direito, mais três ou quatro tópicos que a gente passeia mais ou menos por eles mas a gente não acaba... não sei se é vendo direto na prática ou se fica meio oculto mas é... mas acho que a gente tá bem próximo do que a gente viu na teoria, pelo menos se não é tudo, acho que o mais importante, que a gente acabou passeando na prática"* (SE1, fase 2).

> *"Algumas coisas não puderam ser aplicadas, porque é... tinha essa coisa do... do sistema ser desconhecido e também como eu falei antes, o tempo que a gente tinha pra realizar, mas no geral o que era mais importante foi aplicado"* (SE2, fase 2).

Não houve tempo suficiente para outros projetos práticos
Além dos tópicos teóricos que não foram trabalhados pelas atividades práticas, faltou a realização de um projeto onde as modificações necessárias forçassem a interação entre os grupos (problema anteriormente mencionado em 6.2.2). O professor atribuiu a não execução do mesmo à falta de tempo para o seu planejamento e a deficiência apresentada pelos estudantes no uso do sistema de controle de versões distribuído (ver 6.2.3), que atrapalhou o andamento dos projetos anteriores. Por conseguinte, um treinamento a mais sobre sistema de controle de versões distribuído seria necessário.

> *"(...) a gente não conseguiu realmente fazer um projeto onde as equipes se integrassem, isso foi... e isso foi meio que negativo, é...eles não viveram os problemas de desenvolvimento típicos onde você tem que fazer merge de... de mudanças e... então isso faltou, né, e isso demandaria ter mais tempo, né, e mais organização, mas acho que valeria a pena fazer"* (P1, fase 2).

> *"(...) eu percebi o seguinte, que tem algo mais a ser feito, por exemplo, existe um próprio processo de, da pragmática do, de utilizar um sistema de controle de versão distribuído" (...) "então eu acho que talvez devesse ter uma etapa de como lidar com o sistema de controle distribuído pra que isso não atrapalhasse o processo próprio de, de aprendizagem de como fazer a mudança"* (P1, fase 2).

Paralelamente, um estudante também reconheceu a falta de tempo como justificativa para a não realização do terceiro projeto como estava previsto.

> *"Acho que tem sempre a questão do tempo, né? A gente poderia ter explorado mais ou por exemplo, ter feito o terceiro projeto como tava previsto antes. O tempo é sempre uma coisa negativa... nesse ambiente acadêmico"* (SE2, fase 2).

Linguagem utilizada – Java
Apesar de Java ser a linguagem de programação utilizada ao longo de várias disciplinas do curso, um dos estudantes informou que não gostou do JabRef, porque o software foi desenvolvido com essa linguagem. Inclusive, essa insatisfação gerou a desmotivação do estudante para a realização das atividades práticas.

> *"Olhe, eu não gostei muito do software que ele apresentou pra gente, pra que a gente fizesse alteração, o JabRef" (...) "Porque... é .... Java não é uma das linguagens favoritas minha, então, pra mexer assim... não tenho aquela vontade assim de mexer com Java... Assim a proposta do software é*

> *muito boa... e as alterações que ele pediu, que ele pediu também, são alterações relativamente simples e que... não exige muito do... da pessoa assim de mexer em código... vai ser mais usar o que ele ensinou na... nas aulas expositivas"* (SE5).

Condução das aulas práticas
O mesmo estudante que reclamou da linguagem Java, também não gostou das aulas práticas:

> *"(...) acho que a turma ficava um pouco perdida e acho que não andava muito o conteúdo"* (SE5).

Por outro lado, outro estudante declarou que o professor disponibilizou a descrição clara das atividades que precisavam ser realizadas.

> *"(...) o documento era muito bem explicado o que tinha que ser feito"* (SE6).

Como o objetivo da aula prática era que os estudantes discutissem e realizassem as modificações solicitadas no laboratório, observamos que alguns problemas realmente ocorreram. Como exemplo, citamos que inicialmente nem todos os equipamentos estavam com todos os softwares necessários instalados, a maior proximidade entre os estudantes, a falta de condução e cobrança diretas provocaram muitas conversas paralelas. Todavia como estava claro o que os grupos precisariam fazer, constatamos também que houve desinteresse pela execução das atividades por alguns estudantes, outra insatisfação relatada por um dos estudantes e reportada a seguir.

Participação dos membros do grupo
Um estudante demonstrou bastante insatisfação pela falta de participação dos demais membros de seu grupo na realização das atividades práticas. Até mesmo sugeriu que em uma nova oferta da disciplina, o professor procurasse fazer o acompanhamento das atividades de modo mais individualizado.

> *"(...) o grupo às vezes o pessoal não trabalha direito e, felizmente é uma disciplina, se fosse o mercado de trabalho é... cabeças iam rolar porque o pessoal não tava trabalhando, mas aqui tem a mesma pressão por conta de nota, só que é um pouco complicado você sair dedurando todo mundo que não tá trabalhando direito" (...) "Nem todo mundo trabalhou direito, ficou poucas pessoas trabalhando e muitos na ponta. Esse é o lado negativo, infelizmente"* (SE1, fase 2).

> *"Eu acho que tinha que cobrar mais rigor dos grupos, mais participação dos componentes, eu acho que só. Estaria ideal já"* (SE1, fase 2).

Os demais estudantes também reconheceram esse problema, inclusive em seus grupos, porém não demonstraram o mesmo grau de insatisfação com relação ao fato, como o estudante mencionado.

> *"Basicamente, eu trabalhei em DUPLA, com um outro colega"* (SE2, fase 2).

> *"Eu tive contato em particular com duas pessoas de dois grupos diferentes. Um teve um relacionamento muito positivo com os colegas, não sei se pela proximidade que eles já tinham e pela forma como eles lidam com o tipo de trabalho. Já o outro tinha uma mesma visão que eu, tava um pouco desmotivado por conta dos colegas não estarem participando, mas bastante interessado em resolver, em solucionar, em aprender"* (SE2, fase 2).

> *"O meu grupo na verdade foi... minha participação foi metade-metade. Como eu sempre trabalhava junto com eles, a gente sempre dividia o trabalho da melhor forma. Ficaram algumas pessoas que não trabalharam, porque é normal, quando a gente trabalha em grupo. Mas as três pessoas que estavam acostumadas... duas pessoas que eu tava acostumado trabalhar, minha participação foi... nós três fizemos a maior parte do... do desenvolvimento"* (SE6).

O próprio professor reconheceu o problema:

> *"(...) o desenho dos laboratórios, das atividades práticas, não foi suficiente pra gente garantir que todos tivessem uma participação equitativa"* (P1, fase 3).

Atendimento das expectativas

Por outro lado, extratos demonstram que a disciplina atendeu às expectativas dos estudantes, tanto no tocante à aprendizagem, quanto à metodologia aplicada.

> *"Excelente! É... eu tinha uma visão do que eu esperava dessa disciplina e não me decepcionei com isso, foi mais ou menos o que eu esperava, é... aprender novas técnicas de como lidar com evolução e manutenção de softwares... eu acho que eu posso aplicar tudo, eu acho que vai me ajudar bastante, se eu seguir esse caminho"* (SE2, fase 2).

> *"(...) eu acho que atendeu minhas expectativas em relação ao que seria mesmo, ao que eu queria que fosse a disciplina na verdade. Eu queria que fosse desse jeito aí mesmo, com práticas, com aulas expositivas"* (SE8).

Outro estudante ressaltou que a disciplina superou as suas expectativas.

> *"Ela atendeu minhas expectativas. Na verdade, eu pensei... eu me inscrevi na disciplina pensando que fosse... que fosse feito de outra forma... mas foi feito melhor do que eu imaginava que ela fosse feita"* (SE7).

Resultados do questionário também apontam a satisfação com a disciplina, inclusive com a superação das expectativas. Entre as opções de respostas relacionadas ao atendimento das expectativas, 33% dos respondentes afirmaram que a disciplina superou suas expectativas, 67% declararam que a disciplina atendeu as expectativas e nenhum estudante considerou que a disciplina não tivesse atendido suas expectativas.

O professor considerou positivo a experiência proporciona pela disciplina, porém alguns pontos ainda precisam de melhorias.

> *"(...) essa oferta já é positiva por eles terem vivido essas questões, esses problemas, né, de pegar um software desconhecido e tentar fazer mudanças, pegar um software desconhecido e fazer uma mudança de complexidade um pouco maior onde existe impacto, onde a questão não tá localizada em só uma classe, mas a mudança se propaga pra outras, outras classes, isso tudo já é extremamente positivo... eles não tinham... eles não tinham essa vivência, né, e isso é ótimo, né, agora, é... falta algo mais"* (P1, fase 2).

> *"Do ponto de vista da aprendizagem de um processo de mudança implementar, eu acho que sim. Eu acho que faltou algo em relação a eles se aprofundarem na prática"* (P1, fase 3).

Do ponto de vista do professor, é necessário planejar melhor os requerimentos de mudanças para que todas as etapas do processo sejam cobertas e para que haja necessidade de merge entre as modificações realizadas pelos diferentes grupos. Também é preciso estabelecer uma forma de garantir a participação equitativa dos membros de cada grupo e que os estudantes apliquem realmente o processo de mudanças incremental.

Sentimento de realização
Um estudante demonstrou o sentimento de realização por ter conseguido realizar as atividades apesar do desafio proposto:

> *"(...) a gente conseguiu... pegar o código, código novo, código fonte complexo, em termos de números de classe é muito grande, muitos conceitos envolvidos... Pra que eu já vi, pra o que eu já fiz, o software é muito grande, então, por isso, pra mim é muito difícil às vezes de ... de analisar, de chegar algum lugar" (...) "mas nem por isso a gente não conseguiu fazer as mudanças. A gente conseguiu... realizar tudo aquilo que foi pedido. Então, eu acho que é sim e que é um desafio a mais, ainda"* (SE1, fase 3).

A partir desse ponto de vista, enxergamos a possível influência da 'experiência de mundo real' na 'satisfação' do estudante (ver Figura 6.1).

Continuidade
O último aspecto relacionado à satisfação categorizado foi a possibilidade de 'continuidade' (ver Figura 6.10). Existe a perspectiva de continuidade sob dois aspectos: 'repetição em outra oferta' e 'usar novamente projetos de código aberto'.

Repetição em outra oferta
Em vários pontos dessa seção, percebemos sugestões do professor para uma nova oferta da disciplina. Algumas coisas precisariam ser aperfeiçoadas, outras foram boas ou mesmo satisfatórias, de modo que poderiam ser repetidas.

> *"Algumas das coisas que ficaram melhor é... mais sólidos, os conceitos que ficaram mais sólidos para os estudantes e que eles gostaram muito, foi a questão*

> *de localização de conceitos, é... isso sem dúvida eles melhoraram as práticas que eles tinham anteriormente, né, tão fazendo de maneira mais sistemática, né, e fazer isso sobre um sistema grande, então isso são coisas que eu repetiria...”* (P1, fase 2).

A fim de obter a visão dos estudantes sobre a questão da repetibilidade, questionamos se os mesmos recomendariam a disciplina a seus colegas em uma nova oferta da mesma. 89% dos respondentes recomendariam que os colegas cursassem a disciplina e o restante (11%) preferiu não responder à questão.

Usar novamente projetos de código aberto
O professor afirmou que usaria novamente projetos de código aberto, porém dedicaria mais tempo para estudar o software e planejar melhor as mudanças que seriam solicitadas.

> *“Usaria, usaria... Só que eu, provavelmente, eu teria que dedicar mais tempo ao planejamento, né, dessas mudanças, e a entender mais sobre o software, mais do que... do que realmente foi explorado nesse trabalho”* (P1, fase 2).

Utilizamos duas questões do questionário para capturar a visão dos estudantes no que diz respeito ao emprego de projetos de código aberto em disciplinas de engenharia de software. Na primeira, perguntamos sobre o uso dos códigos dos projetos como exemplos nas atividades práticas das disciplinas. 89% dos respondentes acham que esses códigos poderiam ser mais utilizados. Na segunda, perguntamos sobre o uso dos códigos de maneira que os estudantes precisariam manipular diretamente esses códigos, isto é, alterando, testando, corrigindo bugs, etc. Novamente, 89% dos respondentes acham que sim, que esses códigos poderiam ser mais utilizados.

6.2.8 Escolha do Software

O tema ‘escolha do software’ surgiu porque identificamos alguns fatores que influenciaram as dificuldades a serem enfrentadas, motivação, aprendizagem e satisfação dos estudantes com relação ao emprego da abordagem e que dependem desta escolha. Os fatores encontrados foram (ver Figura 6.11): ‘domínio do software’, ‘tecnologia empregada’, ‘situação do código’, ‘situação da documentação’ e ‘atendimento da proposta didática’.

Quanto ao ‘domínio do software’, observamos três aspectos que colaboraram para a motivação e satisfação dos estudantes no uso do OSP escolhido: ‘conhecimento prévio’, ‘domínio fácil de entender’ e ‘utilidade’.

Um estudante destacou que conhecer e ter utilizado a ferramenta previamente tornou o projeto mais interessante, principalmente por estar conhecendo-a de forma mais detalhada e alterando o seu código.

> *“Eu já conhecia por utilizar Latex e achei interessante, pra mim foi mais interessante por causa disso, que era uma ferramenta que eu já utilizava e que eu tive a oportunidade de conhecer mais a fundo e de modificar, então pra mim, teve essa questão do, do conhecimento prévio da ferramenta, tornou mais interessante, eu achei interessante”* (SE1, fase 2).

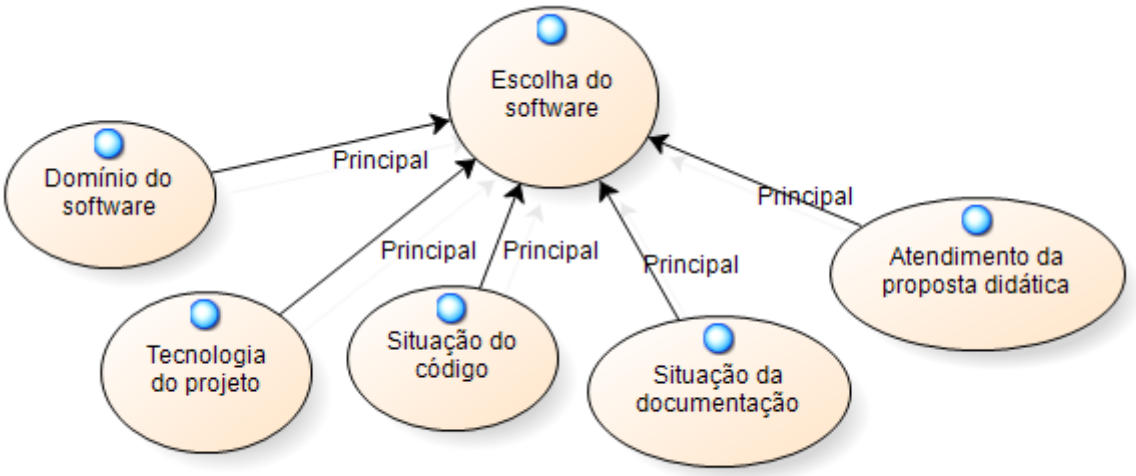

Figura 6.11 Fatores que influenciam a escolha do software.

Posteriormente, novamente o estudante ressaltou que trabalhar com outros softwares conhecidos e utilizados, torna o aprendizado mais interessante.

> *"Então... e esse software tal é sempre coisa que a gente utiliza, na maioria das vezes a gente utilizou. A gente viu Eclipse, a gente viu NetBeans, a gente viu Spring (...). Então, são coisas que você utiliza. Então como você utiliza isso... pra mim foi interessante"* (SE1, fase 3).

O professor lembrou que o domínio da aplicação era fácil de entender.

> *"(...) eu acho que a escolha do JabRef foi muito boa, porque ele não é um software grande demais e aquela história de ter um domínio que é relativamente fácil de entender (...)"* (P1, fase 2).

Outro estudante, de certo modo, confirmou o pensamento do professor, acrescentando que a utilidade da aplicação tornou-a mais interessante.

> *"Eu não conhecia. Eu nunca usei Latex, então nunca tinha utilizado o JabRef antes. Achei interessante é... o professor até fez um comentário sobre essa questão dele poder ser utilizado em diversos ambientes, como por exemplo, a gente poderia utilizar isso no Word e já fazer inclusão de referências e utilização dessas referências... Achei, achei interessante o projeto. Eu não sei qual é a finalidade dele assim a fundo, apenas o que foi explorado dentro é... da disciplina, mas eu achei bastante útil"* (SE2, fase 2).

No que diz respeito à tecnologia utilizada no projeto, mencionamos em 6.2.7 que um dos estudantes relatou que o software ter sido desenvolvido em Java foi o principal fator que o desmotivou na realização das atividades de manutenção no software.

Já em 6.2.2 descrevemos um conjunto de situações do código e da documentação do JabRef que acarretaram na dificuldade de execução das atividades solicitadas (discutidas em 6.2.3).

Finalmente, no cenário de atendimento das necessidades didáticas, vimos em 6.2.7 que, ao mesmo tempo que o professor considerou o software adequado ao apresentar problemas similares aos encontrados em softwares utilizados nas empresas, relatou que a pequena quantidade de testes implementados no JabRef dificultou cumprir o escopo de todo o processo de mudanças incremental que estava sendo ensinado. Sendo esta situação, portanto, uma das causas para a cobertura parcial dos tópicos teóricos, com a execução das atividades práticas.

6.3 ESTUDO DE CASO EXPLORATÓRIO E RESULTADOS PARA A PESQUISA

6.3.1 Integração com a pesquisa final

Como mencionamos previamente, este trabalho representou um estudo exploratório, realizado antes de estabelecermos os objetivos finais para a pesquisa. Dessa forma, os instrumentos utilizados para a coleta de dados não buscaram única e exclusivamente investigar as consequências da aplicação do uso do OSP como recurso didático no ensino de evolução e manutenção de software. Todavia, os temas e categorias capturados permitiram-nos visualizar o projeto de código aberto como uma experiência de mundo real para o estudante e as consequências disso, isto é, a possiblidade de associação entre a teoria e prática, todas as dificuldades provenientes e, finalmente, a influência de todos esses elementos sobre a motivação, aprendizagem e satisfação dos estudantes (ver Figura 6.1).

No contexto da pesquisa, estes resultados, juntamente com os resultados do mapeamento sistemático (Capítulo 5), influenciaram a definição dos objetivos finais. Entretanto, apesar de reconhecermos a importância da motivação e satisfação para qualquer que seja o processo de ensino-aprendizagem, decidimos não tratá-los ao longo dessa pesquisa. A justificativa para essa exclusão seria a limitação de tempo. A inclusão desses fatores psicológicos exigiria um prazo para análise muito maior do que tínhamos disponível.

Finalmente, dada a relevância dos resultados obtidos neste estudo de caso, os mesmos não poderiam ser simplesmente descartados. Sendo assim, na medida do possível, investigamos tais resultados sob a perspectiva dos objetivos finais definidos para a pesquisa. Inicialmente avaliamos quais objetivos de aprendizagem foram alcançados e consequentemente quais deficiências de formação listadas pela indústria, a abordagem ajuda a suprir. Posteriormente no Capítulo 9, compararemos os resultados deste estudo de caso aos demais resultados obtidos com a realização dos outros estudos, ao longo da pesquisa.

6.3.2 Análise dos objetivos de aprendizagem alcançados

Como dito, ao executarmos este estudo, como ainda estávamos em uma fase inicial da pesquisa, nenhum objetivo de aprendizagem (LO) foi explicitamente investigado. Todavia, após a definição dos objetivos de aprendizagem (reportados no Capítulo 3) pudemos confrontá-los com a aprendizagem capturada a partir da análise qualitativa. A Tabela

6.2 apresenta o resultado de tal comparação e o julgamento se o objetivo foi alcançado, foi parcialmente alcançado ou não foi alcançado, de acordo com os relatos dos estudantes reportados em 6.2.6.

Observando a Tabela 6.2, verificamos que a abordagem empregada permitiu o alcance de vários objetivos de aprendizagem relacionados às áreas de conhecimento de Manutenção, Construção e Gerência de Configuração de Software, além da Prática Profissional em Engenharia de Software (Capítulo 3). Entretanto, destacamos algumas situações particulares. A primeira refere-se ao não alcance do objetivo LO125 (aplicar reengenharia para um software existente de médio porte), uma vez que o mesmo trata da aprendizagem ao nível de 'aplicação' na taxonomia de Bloom ou de 'uso' na taxonomia do CS2013 (ver Capítulo 3) e os estudantes reportaram o conhecimento somente em nível de 'recordação' ou 'familiaridade', respectivamente. Quanto ao conteúdo de 'recuperação arquitetural', não encontramos nenhum objetivo de aprendizagem estabelecido no capítulo citado com o qual pudéssemos associá-lo. Para as capacidades de usar sistemas de gerenciamento de configuração (LO133) e fazer merge e ramificação de código (LO134), apesar de alguns estudantes relatarem esse aprendizado, outros reportaram dificuldades (ver 6.2.3), sendo assim, consideramos que os mesmos foram parcialmente alcançados. Já para o objetivo LO251a (desenvolver a habilidade de compreender material técnico), apesar de não ter ocorrido nenhum relato dos estudantes sobre o desenvolvimento dessa habilidade, consideramos o objetivo como alcançado, porque, para que os estudantes realizassem as modificações solicitadas, foi necessário que os mesmos compreendessem o código da aplicação, bem como a pouca documentação disponibilizada. De maneira similar, apesar de não termos associado o objetivo LO270 (compreender como teoria e prática influenciam uma a outra) a nenhuma aprendizagem específica, consideramo-lo como alcançado em virtude do que foi discutido em 6.2.4. Finalmente, mesmo não tendo sido diretamente investigado, consideramos que o objetivo LO227 (habilidade de trabalhar efetivamente como membro de equipes) não foi alcançado, dado que ocorreram várias situações em que não houve comprometimento dos membros das equipes na execução das atividades, conforme reportamos em 6.2.7.

6.3.3 Análise da contribuição da adoção de OSP no suprimento das lacunas de formação apontadas pela indústria

Verificamos as lacunas de formação apontadas pela indústria (descritas no Capítulo 3) que a abordagem pode ajudar a suprir, de acordo com os objetivos de aprendizagem alcançados. Para julgarmos se houve contribuição para o suprimento de determinada lacuna ou deficiência, simplesmente verificamos se o objetivo de aprendizagem associado foi ou não tratado durante a aplicação da abordagem e, consequentemente, se foi alcançado, parcialmente alcançado ou não foi alcançado. Como referência para o alcance ou não dos objetivos utilizamos a Tabela 6.2.

A Tabela 6.3 apresenta as deficiências e os objetivos de aprendizagem associados, organizados por área de conhecimento do SWEBOK. A última coluna indica se a abordagem contribuiu, contribuiu parcialmente ou não contribuiu para o suprimento da deficiência.

Através da Tabela 6.3, notamos a contribuição da abordagem para o suprimento

Tabela 6.2 Objetivos de aprendizagem alcançados no estudo de caso sobre Evolução e Manutenção de Software (EC1).

Aprendizagem capturada	**Objetivo de aprendizagem**	**Id.**	**Alcance do objetivo**
Decaimento de código	Identificar as questões principais associadas com a evolução de software e explicar seu impacto no ciclo de vida do software.	LO113	Sim
Reengenharia	Aplicar reengenharia para um software existente de médio porte.	LO125	Não
Recuperação arquitetural	-	-	-
Processo de evolução e manutenção de software.	Descrever o processo de analisar e implementar mudanças no código de um projeto grande.	LO69	Sim
Localizar os conceitos envolvidos e analisar o impacto da mudança.	Estimar o impacto da mudança solicitada para um produto de médio porte existente.	LO120	Sim
Modificar o software sistematicamente.	Realizar a manutenção de um software real de médio porte.	LO118	Sim
Refatorar código existente	Usar refatoração no processo de modificação de um componente de software.	LO319	Sim
Usar outros recursos do IDE	Construir, executar e depurar programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração visual	LO82	Sim
Usar ferramenta de gerenciamento de configuração	Demonstrar a capacidade de usar as ferramentas de gerenciamento de configuração, controle de versões e gerenciamento de releases em suporte ao desenvolvimento de um software de médio porte.	LO133	Parcial
	Ser capaz de fazer merge e ramificação de código (*branching*)	LO134	Parcial
Lidar com código desenvolvido por terceiros	Ter experiência com código desenvolvido por terceiros.	LO268	Sim
Experiência com um projeto de porte razoável e mais complexo. Vivência da complexidade das atividades.	Ter vivenciado pelo menos um projeto real.	LO269	Sim
-	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).	LO251a	Sim
-	Compreender como teoria e prática influenciam uma a outra	LO270	Sim
-	Habilidade de trabalhar efetivamente como membro de equipes.	LO227	Não

Tabela 6.3 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Evolução de Software (EC1).

Deficiência	**LO**	**Contribuição**
Construção de Software		
Uso de ambientes integrados de desenvolvimento	LO82	Sim
Manutenção de software		
Manutenção de software	LO118	Sim
Gerência de Configuração do Software		
Uso de ferramentas de gerenciamento de configuração de software	LO133	Parcial
Realização de *merge* entre códigos e/ou ramificação do código (*branching*)	LO134	Parcial
Prática Profissional em Engenharia de Software		
Experiência com projetos reais	LO269	Sim
Experiência com trabalho em equipe	LO227	Não

de deficiências relevantes para a formação do profissional da indústria de software, as quais são: saber manter software (LO118), ter experiência com projetos reais (LO269) e saber usar ambientes integrados de desenvolvimento (LO82). Usualmente o profissional terá que lidar com estas circunstâncias em seu dia-a-dia de trabalho. Semelhantemente, ocorre para as deficiências no uso de sistemas de gerenciamento de configuração (LO133 e LO134) e saber trabalhar em equipe (LO227). Contudo nesses casos, a contribuição da abordagem não foi suficiente, uma vez que os objetivos de aprendizagem associados não foram totalmente alcançados. Ao longo do texto, discutimos que o próprio professor reconheceu a necessidade de melhorias nesses pontos.

Convém ressaltar que as áreas de conhecimento de Construção de Software e Prática Profissional em Engenharia de Software possuem ainda outras lacunas de formação identificadas, porém não foram enumeradas na tabela por não terem sido diretamente trabalhadas ou investigadas pela abordagem. Para a lista completa de deficiências consultar Tabela 3.5.

No próximo capítulo descrevemos e apresentamos os resultados obtidos com a adoção de um OSP para a aprendizagem de testes de software.

Capítulo 7

ESTUDO DE CASO 2 - TESTES DE SOFTWARE

Testes de software corresponde a uma das áreas de conhecimento definidas pelo SWEBOK. A execução de testes representa uma relevante forma de garantir a qualidade do software. Atualmente, existem vários *frameworks* e ferramentas que facilitam a construção de testes automatizados. Quando o desenvolvedor incorpora testes automatizados em um software, ele procura garantir que a funcionalidade do software opere corretamente, mesmo que seu código seja alterado. Portanto, o estudante além de saber testar o software, ele precisa aprender a implementar testes automatizados.

O projeto de código aberto aparece como recurso a ser utilizado pelo professor na tentativa de aproximar o estudante de situações que ele poderá se defrontar quando estiver trabalhando na indústria, ou seja, ter que testar um software de tamanho razoável desenvolvido por outras pessoas.

O propósito desse capítulo é apresentar o estudo de caso no qual utilizamos um projeto de código aberto, para que os estudantes praticassem o conteúdo de testes de software implementando testes automatizados. Sendo assim, inicialmente descreveremos todo o contexto de aplicação da abordagem, incluindo a metodologia usada na coleta e análise dos dados. Posteriormente, reportaremos os resultados encontrados, baseando-nos nos objetivos traçados para nossa pesquisa (Capítulo 1).

7.1 CONTEXTO DO ESTUDO DE CASO EM TESTES DE SOFTWARE

Para melhor organização do texto, dividimos a descrição do contexto do estudo nos seguintes tópicos: (i) cenário do estudo de caso; (ii) população do estudo e envolvidos; (iii) aplicação da abordagem; (iv) coleta de dados e (v) análise de dados.

7.1.1 Cenário do Estudo de Caso

O cenário desse estudo caso foi a disciplina Desenvolvimento de Software III ofertada no segundo semestre de 2015 [1], para o curso de Ciência da Computação da Universidade

[1] O segundo semestre de 2015 ocorreu no período de janeiro à maio de 2016, em virtude do atraso no calendário acadêmico da Universidade.

Federal de Sergipe (UFS). A disciplina, representa a terceira e última disciplina obrigatória na área de engenharia de software para este curso. Enquanto que nas disciplinas anteriores, os estudantes aprendem sobre engenharia de requisitos e projeto de software, a disciplina, caso desse estudo, tem um programa muito mais abrangente, englobando desde testes de software, implantação, manutenção, engenharia reversa, reengenharia, refatoração, gerência de configuração, até noções de qualidade de software e gerenciamento de projetos de software. Para termos uma visão mais abrangente da diversidade de áreas estabelecidas pelo SWEBOK que podem ser abarcadas pelo emprego da abordagem de uso de OSP, decidimos nesse estudo focar a sua utilização para o ensino de testes e qualidade de software. Todavia, fugindo da programação inicial, aplicamos a abordagem apenas para o ensino de testes de software. A razão para essa mudança de planejamento foi a sobrecarga de atividades de avaliação imposta aos estudantes ao longo da disciplina.

Para a exposição do conteúdo, o professor aplicou o método tradicional de ensino, com aulas expositivas e exercícios. Acrescentamos a abordagem de utilização de projeto de código aberto para colocar em prática a teoria sobre testes de software de forma opcional. Caso o estudante participasse, as notas de suas provas poderiam ser substituídas pelas notas obtidas com a realização das atividades.

Propusemos o projeto aos estudantes após a realização da primeira prova, onde o conteúdo de testes foi abordado pelo professor da disciplina. Dessa maneira, o conteúdo foi apresentado de maneira independente da abordagem com OSP e a abordagem serviria para complementar a aprendizagem com a implementação de testes em um projeto real de software.

O projeto consistiu de duas etapas. Na primeira etapa, os estudantes, divididos em equipes, seriam responsáveis por implementar individualmente testes de unidade e integração para o módulo de funcionalidades atribuído a sua equipe. Dentro do módulo estabelecido, os estudantes discutiam entre si, o que cada um deveria testar. Na etapa final, deveriam: (i) implementar testes funcionais baseados na interface da aplicação; (ii) analisar a cobertura dos testes implementados; (iii) elaborar e executar ao menos uma vez um plano de testes de regressão, incluindo os testes implementados pela equipe e finalmente, (iv) responder o questionário sobre a aplicação de testes de sistemas e métricas de testes de software no projeto de código aberto utilizado, para a posterior discussão em sala de aula.

7.1.2 População do Estudo e Envolvidos

Nossa população de estudo foi composta pelo professor e os estudantes da disciplina, sendo que dos 34 matriculados, inicialmente 33 estudantes aceitaram participar do projeto, porém com o andamento da disciplina e a participação sendo voluntária, somente 19 participaram da primeira parte do projeto e 15 participaram da etapa final do projeto.

Seguindo a grade curricular do curso de Ciência da Computação, a disciplina encaixa-se no sétimo período, dentro dos oito períodos totais do curso. Portanto, os estudantes ao cursá-la estão prestes a se formar ou já são formandos, isso quer dizer que são razoavelmente maduros quanto à formação acadêmica. Dos 15 estudantes que efetivamente participaram, sete, até o momento não tinham tido nenhuma experiência com projetos

reais, dois participaram de até dois projetos anteriormente e, finalmente, seis estudantes tinham participado de três ou mais projetos reais. Cabe ressaltar que todos os estudantes mais experientes da turma aceitaram o desafio e participaram do projeto.

O professor responsável pela disciplina tem como formação a graduação, duas especializações e mestrado na área de computação. Além de trabalhar na área de desenvolvimento de software há 20 anos, já teve algumas experiências anteriores na área de ensino, lecionando disciplinas de programação e engenharia de software. Ao ser entrevistado, o professor admitiu utilizar o método tradicional de ensino, com aulas expositivas e avaliação por meio de provas, por não ter tido uma formação na área de didática. Consequentemente, reproduz o método pelo qual foi instruído ao longo de sua formação.

> *"A nossa formação é na área da ciência da computação e tudo mais, ela não tem uma coisa muito específica na área didática." (...) "a gente aplica o que a gente... é... aprendeu como aluno (risos)"* (P2).

Por outro lado, o nosso ponto de vista e a motivação para realizarmos essa pesquisa direciona para novas propostas e mudanças no método tradicional de ensino. Depois de mais de dez anos lecionando disciplinas da área de engenharia de software, cremos que mudanças são importantes. Acreditamos que a participação ativa do estudante ao longo do processo é relevante para a sua motivação e aprendizagem, que o professor precisa atuar mais como suporte e orientador, do que como transmissor de conhecimento, enfim, no caso específico dessa pesquisa, que a utilização de projetos de código aberto como exemplo de projetos reais é particularmente benéfica para a formação profissional do estudante.

Para dar suporte aos estudantes tanto no uso das ferramentas quanto na manipulação do código do projeto de código aberto, envolvemos um assistente. Outro estudante que já havia cursado a disciplina e com bastante experiência profissional assumiu esse papel.

7.1.3 Aplicação da abordagem

Para a aplicação da abordagem, o primeiro passo foi ajustar os objetivos da pesquisa ao programa da disciplina. Como mencionamos anteriormente, escolhemos trabalhar testes de software. Sendo assim, selecionamos os objetivos de aprendizagem relacionados a essa área de conhecimento estabelecidos no Capítulo 3. Todavia, analisando como o professor da disciplina apresentou esse conteúdo, para que o escopo da pesquisa não ficasse muito limitado, decidimos aprofundar alguns conceitos em sala de aula, acrescentar alguns assuntos, proporcionar um caráter mais prático para o conteúdo e, por fim, expor algumas ferramentas de testes. Como, no decorrer da disciplina, os estudantes não tiveram contato com nenhum exemplo de implementação de testes automatizados em um projeto de software, resolvemos construir uma aplicação especificamente para esse fim, denominada de TicketSales, e implementar alguns exemplos no próprio projeto de código aberto empregado.

Escolhemos utilizar o JabRef como projeto de código aberto a ser trabalhado. As justificativas para essa escolha foram: (i) o software é implementado em Java, linguagem que faz parte do currículo do curso e é empregada em outras disciplinas do curso;

(ii) representa um sistema de informação, com um domínio fácil de ser compreendido; (iii) pode ser útil aos estudantes que fazem iniciação científica ou para a elaboração do trabalho de conclusão de curso, finalmente, (iv) representa um exemplo de projeto real, com a problemática que lhes é inerente, conforme constatamos no estudo de caso anterior (Capitulo 6).

Além de ajustar os objetivos da pesquisa ao programa da disciplina e escolher o projeto de código aberto a ser utilizado, outras atividades foram necessárias para a aplicação da abordagem. A Tabela 7.1 lista todas as atividades que precisamos realizar juntamente com o esforço despendido em sua realização (em número de horas). Essa tabela serve apenas de orientação do que é necessário fazer e uma noção de quanto de esforço é necessário para as pessoas interessadas em aplicar a abordagem. Entretanto, é óbvio que o tempo necessário pode variar bastante, principalmente considerando o material adicional que será disponibilizado aos estudantes e se a escolha do software seguirá critérios mais detalhados. Neste estudo de caso específico, convém ressaltar a quantidade de horas empregadas para a preparação de material de apoio dada a necessidade apontada no início desta subseção. Ainda destacamos que apesar da atividade de compreensão do código do software ser extremamente importante, não a contabilizamos individualmente. Na verdade, ela ocorreu quando estávamos preparando os exemplos, dando suporte aos estudantes e avaliando os testes implementados pelos mesmos.

Tabela 7.1 Atividades necessárias para emprego da abordagem.

Atividades necessárias	**Esforço (horas)**
Estabelecer os objetivos de aprendizagem pretendidos.	2
Identificar as possíveis atividades que com o uso do projeto de código aberto poderiam ajudar com que os estudantes alcançassem os objetivos de aprendizagem pretendidos.	2
Pesquisar e definir o projeto de código aberto a ser utilizado.	8
Compreender o software (leitura da documentação existente e uso das funcionalidades).	10
Compreender o software (código)	
Dividir as funcionalidades a serem testadas entre os grupos de estudantes, analisando as dificuldades envolvidas	4
Preparar slides de apresentação do projeto prático e introdução ao projeto de código aberto a ser utilizado	8
Preparar material de suporte à configuração do ambiente a ser utilizado	10
Detalhar as funcionalidades que podem ser testadas	5
Detalhar as atividades a serem realizadas pelos estudantes	11
Preparar material de apoio aos estudantes	40
Dar suporte as atividades dos estudantes	32
Avaliar as soluções apresentadas	26
Dar o *feedback* sobre as soluções apresentadas	8
Total	166

Ao longo da aplicação da abordagem, realizamos quatro encontros presenciais com a turma, os objetivos e datas de cada encontro, bem como o material disponibilizado aos estudantes nos encontros, encontram-se detalhados nas tabelas 7.2 e 7.3, respectivamente. Convém ressaltar que além dos encontros, tanto a pesquisadora quanto o

assistente ficaram em contato constante com os estudantes através do correio eletrônico, dando suporte e esclarecendo dúvidas.

Tabela 7.2 Encontros presenciais da pesquisadora com os estudantes no estudo EC2.

Encontro	**Objetivo**	**Data**
1	Auto-apresentação e apresentação do assistente. Apresentar o projeto de pesquisa, identificar os estudantes que participariam da pesquisa, obter a assinatura dos TCLEs, levantamento da experiência dos estudantes, apresentar o JabRef	15/03/2016
2	Esclarecer as diferenças entre testes de unidade e integração; apresentar como os testes de unidade devem ser executados; introduzir a simulação de objetos (*mock objects*) e utilização do Mockito. Apresentar exemplos de testes de unidade e integração implementados no TicketSales e JabRef.	31/03/2016
3	Apresentar tópicos finais que seriam trabalhados: testes funcionais (com exemplos implementados), métricas de testes, cobertura de testes, testes de regressão e integração contínua.	14/04/2016
4	Discussão sobre a segunda parte do trabalho	17/05/2016

Algumas dificuldades surgiram no percurso da aplicação da abordagem. A primeira delas deveu-se ao programa extenso da disciplina, consequentemente, a quantidade de atividades de avaliação estabelecida pelo professor e a necessidade de tempo para a realização das atividades do projeto, cuja participação era voluntária. Essa problemática acarretou na redução do escopo do estudo de caso, uma vez que retiramos a área de qualidade de software, como mencionamos no início da descrição desse estudo. Mesmo assim, vários estudantes desistiram e para conseguirmos a participação dos demais, precisamos adiar diversas vezes as entregas periódicas dos produtos das atividades, anteriormente planejadas. Em consequência desses adiamentos, o tempo para execução das demais atividades tornou-se cada vez mais exíguo, fazendo com que, não exigíssemos nenhuma entrega individualizada na segunda parte do trabalho e que a discussão final sobre os trabalhos fosse prejudicada. Os comentários finais gerais e específicos dos trabalhos de cada estudante tiveram que ser enviados eletronicamente, após o término das aulas. Dada a falta de tempo, também desistimos de trabalhar algumas métricas de testes de software utilizando versões anteriores do software ou exigir que os estudantes reportassem os erros encontrados à comunidade do projeto.

Apesar de terem ocorrido quatro desistências entre as duas etapas do projeto, três estudantes pleitearam a sua integração. Porém, como exigimos que os mesmos também executassem as atividades da primeira etapa, dando-lhes novos prazos, apenas dois estudantes de fato envolveram-se.

Outra dificuldade foi a quantidade de material adicional que precisamos preparar e apresentar aos estudantes, conforme mencionamos anteriormente, a fim de permitir a cobertura de pelo menos um escopo razoável de objetivos de aprendizagem específicos da área de testes de software.

Por último, relatamos a dificuldade específica de utilização de um software sobre o qual o professor não tem o domínio, fato comum quando utilizamos projetos de código aberto. Como a visão que se tem do software é superficial, torna-se incerto que as funcionalidades

Tabela 7.3 Material disponibilizado no estudo EC2.

Encontro	Título	Objetivo
1	Apresentação do Projeto Prático	Descrever os objetivos; como seria o andamento do projeto; testes que deveriam ser executados; dicas de como realizar os testes.
1	Apresentação do JabRef	Apresentar uma introdução ao projeto que seria testado; explicar como é o processo de desenvolvimento de um projeto de código aberto e motivar o estudante a participar do projeto.
1	Configuração ambiente JabRef	Apresentar o passo-a-passo de como os estudantes criariam o ambiente de desenvolvimento e testes do JabRef em seus computadores.
2	Testes de Unidade e Integração na Prática	Esclarecer as diferenças entre testes de unidade e integração; apresentar como os testes de unidade devem ser executados; introduzir a simulação de objetos (mock objects) e utilização do Mockito
2	Construção do TicketSales	Software criado para possibilitar a disponibilização de exemplos de testes mais próximos das necessidades encontradas na indústria e mais fáceis de entender que os exemplos elaborados para o JabRef.
2	Exemplos de testes unitários e integração no TicketSales	Apresentar exemplos de testes de unidade e integração implementados
2	Exemplos de testes unitários no JabRef	Apresentar exemplos de testes de unidade no JabRef
2	Exemplos de planejamento e projeto de testes	Apresentar exemplo do planejamento e projeto dos testes de unidade no JabRef
2	Instruções detalhadas trabalho prático	Instruções passo-a-passo de atividades que os grupos precisariam executar para realizar a parte 1 do projeto
3	Testes Funcionais	Introduzir os objetivos, as diferenças para os demais tipos de testes e como fazer testes funcionais
3	AssertJ	Apresentar o AssertJ como framework para ser utilizado para testes da funcionais, a partir da interface da aplicação. Apresentar exemplos no TicketSales e JabRef.
3	Métricas de Testes de Software	Apresentar as métricas relacionadas aos testes que podem ser aplicadas ao longo do projeto, referentes ao processo de testes e para o produto gerado.
3	EclEmma	Apresentar as características e como utilizar a ferramenta EclEmma para identificar a cobertura dos testes executados
3	Testes de Regressão e Integração Contínua	Apresentar conceitos, dicas e discutir a importância de testes de regressão e integração contínua.
3	Roteiro Jenkins	Apresentar o passo-a-passo de como utilizar o Jenkins para o projeto.
3	Instruções para trabalho 2	Instruções para realização da segunda parte do trabalho e questões que deveriam ser respondidas pelas equipes

distribuídas entre as equipes sejam de complexidade semelhantes, ou seja, o professor não tem a medida correta da complexidade da atividade solicitada.

7.1.4 Coleta de Dados

Como mencionamos no Capítulo 4, utilizamos questionários, entrevistas e análise de documentos como instrumentos de coleta de dados neste estudo de caso.

No decorrer do projeto aplicamos três questionários. O primeiro (LET), logo no primeiro encontro com os estudantes (ver tabela 7.2), a fim de identificar a experiência dos estudantes com projetos reais e especialmente com testes e/ou qualidade de software. O segundo (QT1) foi repassado, após o primeiro encontro, com o objetivo de capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a testes de software, antes da aplicação da abordagem. Por fim, ministramos o terceiro questionário (QT2) depois que encaminhamos os comentários dos trabalhos, com a intenção de obter a percepção dos estudantes sobre o seu conhecimento e habilidades sobre testes, após o emprego da abordagem, bem como, a contribuição do projeto para aquisição desse conhecimento e o desenvolvimento dessas habilidades. No mesmo questionário, ainda buscamos detectar a percepção do estudante: (i) sobre a utilização do projeto de código aberto como uma experiência real de desenvolvimento de software; (ii) se as atividades realizadas permitiram que o mesmo fizesse a conexão entre teoria e prática; (iii) sobre a contribuição do projeto para o desenvolvimento de competências relacionadas à prática profissional em engenharia de software; (iv) se a participação no projeto fez com que o mesmo se sentisse mais preparado para o mercado de trabalho, e, por fim, (v) se a participação no projeto contribuiu para a conscientização do mesmo sobre a importância de alguns aspectos da engenharia de software.

As estruturas dos questionários LET, QT1 e QT2 encontram-se nos apêndices B, C e D, respectivamente. Enquanto o LET foi aplicado presencialmente para os estudantes que aceitaram participar da pesquisa, o QT1 e o QT2 foram enviados e respondidos eletronicamente. Para que fosse possível aplicarmos a análise pretendida, solicitamos a identificação dos respondentes em todos os questionários. Convém informar que, um dos estudantes que participou do projeto não respondeu o QT1, portanto ao compararmos as respostas de QT1 e QT2, para as questões cabíveis, somente 14 questionários foram considerados.

Para as entrevistas, como os módulos do software foram divididos entre as equipes formadas, de acordo com a disponibilidade dos estudantes, procuramos entrevistar ao menos um membro de cada equipe. A Tabela 7.4 caracteriza os estudantes entrevistados por equipe e experiência declarada. Para as equipes com maior número de membros, entrevistamos mais de um estudante. Convém reportar que nenhum membro da equipe 5 participou efetivamente do projeto. Também procuramos cobrir proporcionalmente a diferença de experiência prévia com projetos reais apresentada pelos estudantes, ao convidarmos os mesmos para as entrevistas. Como tratam-se de estudantes, classificamos a experiência declarada em três situações: (i) sem experiência, para os estudantes que até o momento não tinham tido nenhuma experiência com projetos reais; (ii) alguma experiência, para aqueles que haviam participado de até dois projetos anteriormente e

(iii) mais experientes, para os que tinham participado de pelo menos três projetos reais.

Tabela 7.4 Caracterização dos estudantes entrevistados no estudo EC2.

Equipe	Módulo do Software	Estudante	Experiência prévia
ET1	Operações sobre bases de dados	ST7	Sem experiência
ET2	Operações sobre as entradas	ST2	Mais experiente
		ST3	Sem experiência
ET3	Ligação Entrada e arquivos físicos	ST8	Sem experiência
ET4	Geração de BibTex Key e Customizações	ST4	Alguma experiência
		ST5	Mais experiente
ET6	Organização das entradas em grupos	ST1	Sem experiência
ET7	Pesquisas normal e avançada	ST6	Mais experiente

Realizamos as entrevistas após o término das aulas, todavia antes de encaminharmos os comentários sobre os trabalhos, por conseguinte, antes que os estudantes respondessem o questionário QT2. Ademais, ressaltamos que entrevistamos o professor (P2) ao término de todas as atividades.

Ainda como fonte de informação, exploramos os testes implementados e os documentos entregues pelos estudantes. O propósito era examinar a aprendizagem demonstrada pelos mesmos.

7.1.5 Análise de dados

No que diz respeito a análise de dados, para os dados qualitativos, aplicamos a análise descrita no Capítulo 4. Para os dados quantitativos, dado o pequeno número de participantes, utilizamos estatísticas descritivas para a análise. Somente para compararmos a aprendizagem antes e depois da aplicação da abordagem, recorremos ao teste de postos sinalizados de Wilcoxon para amostras pareadas, uma vez que os dados não apresentavam distribuição normal.

7.2 RESULTADOS

Nesta seção, apresentaremos os resultados obtidos com a realização desse estudo de caso. As evidências de acordo com os objetivos traçados para a pesquisa estão assim organizadas: (i) experiência real de desenvolvimento de software (Ob7); (ii) conexão teoria e prática (Ob6); (iii) objetivos de aprendizagem alcançados (Ob4); (iv) sentimento de preparação para o mercado de trabalho (Ob8); (v) consequências para a aprendizagem(Ob5) e, por último, (vi) lacunas na indústria de software (Ob9).

7.2.1 Experiência Real de Desenvolvimento de Software (Ob7)

Um dos objetivos dessa pesquisa foi investigar se o estudante consegue enxergar a utilização de um projeto de código aberto como uma experiência real de desenvolvimento de software. Procuramos capturar objetivamente a percepção do estudante a esse respeito, a partir do nível de concordância com quatro afirmações: (i) você acredita que o projeto empregado (JabRef) permitiu que você tivesse uma experiência de mundo real

(Q1R1); (ii) você acredita que o projeto empregado (JabRef) representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software (Q2R2); (iii) o uso do projeto de código aberto (JabRef) permitiu que você visualizasse a complexidade e as dificuldades existentes em um projeto real de software (Q3R3) e (iv) a participação no projeto permitiu que você compreendesse melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software (Q4R4). A Figura 7.1 sumariza os resultados obtidos.

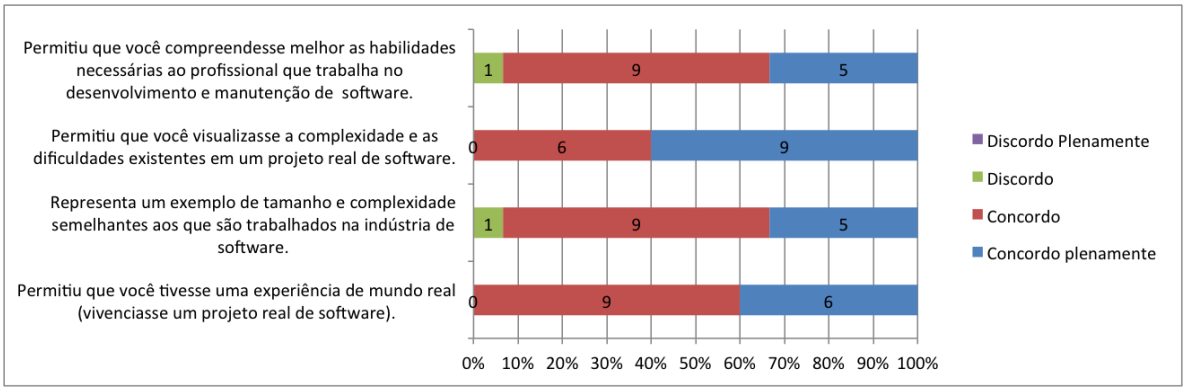

Figura 7.1 Níveis de concordância sobre a abordagem como uma experiência real de desenvolvimento de software no estudo EC2.

Analisando a figura, verificamos que todos os estudantes enxergaram o projeto como uma experiência de mundo real. 93% consideraram que o software utilizado representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software. Novamente, 100% dos estudantes julgaram que a experiência permitiu que os mesmos vivenciassem a complexidade e as dificuldades existentes em um projeto de real de software e 93% acreditaram que foi possível compreender melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software.

A Figura 7.2 ilustra as categorias detectadas subjetivamente sobre a percepção dos estudantes a respeito do projeto como experiência real de desenvolvimento de software, são elas: (i) proximidade dos softwares existentes nas empresas, (ii) proximidade das atividades na indústria e (iii) visão sobre as habilidades necessárias aos profissionais da engenharia de software. A seguir discorreremos sobre cada uma dessas categorias.

Proximidade dos softwares existentes nas empresas

No que diz respeito à proximidade do projeto de código aberto com os softwares existentes nas empresas, encontramos as seguintes características (ver Figura 7.2): (i) tamanho; (ii) complexidade; (iii) problemas na estrutura do código; (iv) escassez de documentação; (v) desenvolvido por várias pessoas; (vi) existe o *feedback* do usuário; (vii) exige configuração de ambiente e (viii) software implementado sem simplificações.

Os estudantes consideraram o JabRef como um software de tamanho médio, reconhecendo que existem softwares "bem maiores" nas empresas. Todavia comparado aos softwares trabalhados ao longo das disciplinas, os mesmos confirmaram que o JabRef foi

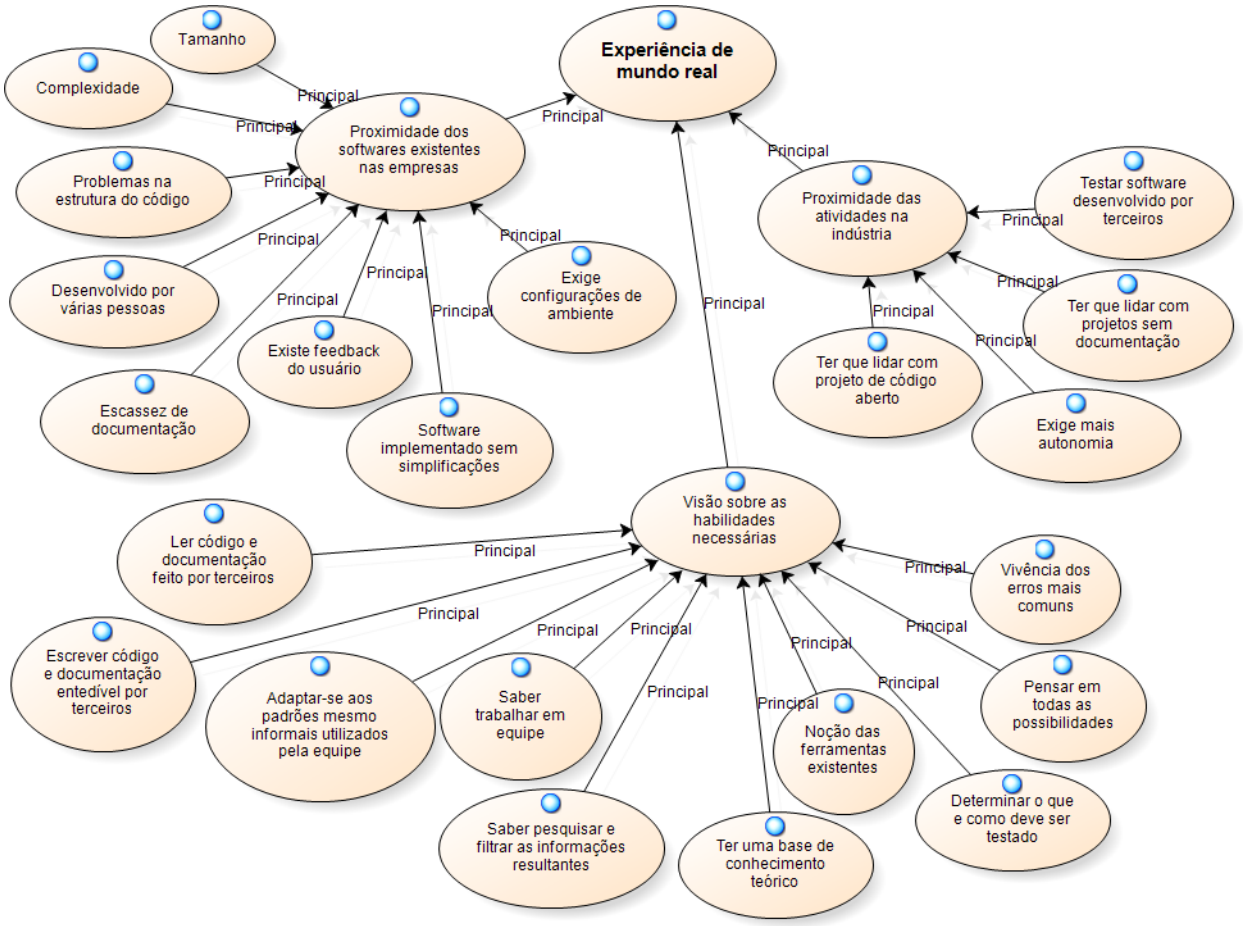

Figura 7.2 Percepção dos estudantes sobre a proximidade do projeto a uma experiência real de desenvolvimento de software no estudo EC2.

o maior projeto com o qual eles tiveram que labutar.

> *"O JabRef é um projeto muito maior comparado aos projetos que a gente desenvolve nas outras disciplinas"* (ST7).

> *"Bem maior do que a maioria dos que eu fiz aqui (risos)"* (ST5).

> *"Eu só poderia dizer que ele seria o projeto maior que eu já mexi"* (ST8).

Quanto à complexidade, em geral os estudantes julgaram o projeto de código aberto utilizado como bastante complexo. Um estudante [2] declarou que os projetos existentes nas empresas atualmente são simples, consequentemente, as atividades desenvolvidas seriam mais simples do que as realizadas no JabRef. Entretanto, ao longo de sua declaração podemos verificar que o mesmo referia-se somente a projetos de gerenciamento de conteúdo Web, ou seja, uma visão limitada de projetos de software.

> *"Os projetos que têm no mercado hoje em dia são coisas bem simples, que até a documentação pode ficar naquele boca a boca, que é só um sisteminha básico para fazer isso (...). Como eles são simples, trabalhar numa empresa é bem mais simples porque... por causa dessas questões. Cuidar de um site online, fazer a publicação de um site online de portifólio é muito mais fácil e você não precisa de documentação nenhuma, porque você só pega o site de portifólio e adapta pra a empresa, você não precisa de uma documentação pra isso"* (ST2).

Por outro lado, confrontando o JabRef com os projetos desenvolvidos na academia, dois estudantes afirmaram que lidaram com projetos também complexos em seus projetos de pesquisa, porque envolviam inteligência artificial, para o primeiro, inteligência artificial e processamento de imagens, para o segundo. Já um terceiro estudante considerou o projeto construído durante a disciplina de projeto e análise de algoritmos também complexo, porém não tão grande quanto o JabRef.

De acordo com os estudantes, a complexidade do JabRef é proveniente de problemas na estrutura do código, resultando em um projeto confuso, difícil de entender.

> *"(...) mas eu achei um pouco... bagunçado, o código em si. É um pouco difícil de ler, de entender o que está acontecendo, porque eles usam algumas coisas meio... meio obscuras, assim... e é um pouco... confuso para se entrar de cara nele (risos)"* (ST5).

> *"(...) eu não sabia onde era interface, não sabia onde era... onde estava instanciando tal coisa... de onde vinha isso... porque eles estavam usando aquilo... então não sei, pareceu complexo (risos)"* (ST6).

[2] O único estudante que discordou, ao responder o questionário, que o JabRef representava um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software.

Um estudante ressaltou que o problema é que não havia a divisão de interesses entre a interface e a lógica da aplicação.

> *"Ele mistura interface com lógica"* (ST1).

Já outro estudante relatou que a geração das telas da interface de forma dinâmica, dependendo da inicialização das preferências, dificultava o entendimento do código.

> *"Como as telas sendo geradas de maneira procedural... dá mais trabalho para poder entender (...) teve esse troço todo intricado do JabRef... das preferências, de inicializá-las..."* (ST4).

Dois estudantes acrescentaram que a complexidade também pode ter sido causada pelo grande número de pessoas envolvidas no desenvolvimento do software e particularmente, pela necessidade deles próprios terem que entender e lidar com código desenvolvido por terceiros.

> *"(...) é uma coisa que se torna complexa, só pelo fato de existir ali... pela quantidade de pessoas envolvidas"' (...) "o JabRef tem essa possibilidade de ser feito por milhares de pessoas"* (ST2).

> *"A complexidade é alta (...) a questão de lidar com o código que outras pessoas viram, que outras pessoas fizeram, diversas pessoas, torna-se também complexo, bem complexo"* (ST3).

> *"Os projetos que eu participei, eles são complexos mas para mim, eles são bem simples, porque eu já conheço de tudo... sei a maioria dos requisitos e entendo tudo por trás deles. Já no JabRef, eu não fazia a mínima ideia de nada, então era extremamente complexo"* (ST2).

Neste último extrato, o estudante comparou a situação de ter que entender o JabRef, que fora desenvolvido por outrem, com a de trabalhar com projetos dos quais ele participou, sendo a primeira situação muito mais complexa. O estudante ainda destacou que a complexidade é agravada pela falta ou escassez de documentação.

> *"Não ter a documentação dele é algo bem complicado, ele se torna extremamente complexo porque ninguém faz ideia do que está rolando, e isso o torna muito mais difícil"* (ST2).

Todas essas situações de 'problemas na estrutura de código', 'ser desenvolvido por várias pessoas' e 'escassez de documentação' também são características que aproximam o projeto de código aberto dos projetos que existem nas empresas (ver Figura 7.2). O relato de um dos estudantes mais experientes da turma descreve bem essa condição.

> *"É bem similar, porque é muito comum a gente ter projetos mal-organizados no mercado hoje em dia... Se tem... Eu trabalhei em alguns lugares que tinham projetos de 99 e projetos que foram crescendo sem documentação, com novas tecnologias surgindo e você vai colocando uma coisa em cima da outra, e tem vários clientes, um cliente quer de um jeito, outro quer de outro, e vai bagunçando e se tornando uma grande... bomba (risos), e quem sofre é o desenvolvedor, principalmente porque aumenta muito o gasto... a rotatividade é sempre alta nessas empresas... Então, eu diria que é bem similar (risos)"* (ST5).

Para um dos estudantes, o projeto de código aberto é semelhante aos existentes nas empresas em virtude da necessidade de configuração do ambiente para que o mesmo seja executado e pela existência de textitfeedback dos usuários do software, solicitando melhorias para os desenvolvedores.

> *"Acredito que seja semelhante, viu... acredito que seja semelhante, até porque, assim... a questão mesmo de configuração.... de... questão de... feedback do usuário, alguma coisa assim, pra melhoria... acho que seria semelhante"* (ST3).

Como última característica que aproxima o projeto de código aberto dos softwares existentes nas empresas, identificamos que o software é implementado sem simplificações (ver Figura 7.2). Na indústria de software, os desenvolvedores têm que produzir software que atenda as necessidades dos usuários. Simplificações ocorrem somente por questões de viabilidade técnica ou financeira. Por outro lado, nas disciplinas dos cursos de computação, os estudantes usualmente lidam com projetos simplificados, isto é, projetos pequenos, pouco complexos, pouco robustos e com poucos módulos:

> *"O máximo que a gente fez... foi um pequeno aplicativo no Android, que mesmo assim, consistia só de... sei lá... quinze classes e acabou"* (ST6).

> *"O que a gente vê geralmente em sala de aula, né, são projetos pequenos, nada nem comercial, nem tão robusto, com vários módulos, nem nada"* (ST4).

Um estudante apontou essa situação como um problema importante a ser enfrentado: nas disciplinas do curso, os estudantes aprendem a lidar com projetos simplificados; no mercado, eles terão que trabalhar com softwares grandes e complexos.

> *"A grande problemática é sempre essa, a academia e meio profissional. Dentro da academia, os projetos tendem não ser tão grandes, exceto os projetos de TCC, mestrado e por aí vai. Fora da academia, os projetos tendem a ser bem maiores, bem mais complexos (...) Na academia você vê projetos pequenos, no mercado são projetos bem maiores"* (ST7).

Utilizar projetos mais próximos da realidade, tal qual é a proposta do emprego de projetos de código aberto, representa então uma quebra nesse paradigma. Os estudantes têm a oportunidade de vivenciar um projeto mais próximo da realidade ainda na Universidade. A opinião de um dos estudantes corrobora com a importância dessa proposta.

> *"É bom pra ter um contato com a realidade também, porque nem tudo são flores, né. Nem todo software está todo bonitinho, fácil de dar manutenção"* (ST1).

Proximidade das atividades na indústria

Os estudantes, para realizarem as atividades solicitadas, tiveram que testar software desenvolvido por terceiros, lidar com projetos sem documentação e lidar com projeto de código aberto. Tais ações exigiram mais autonomia por parte deles. Eles próprios reconheceram que essas atividades são semelhantes às atividades executadas nas empresas (ver Figura 7.2), como podemos ilustrar através dos extratos a seguir.

De acordo com um estudante, as atividades executadas caracterizaram o meio profissional, uma vez que cada estudante teve que entender o que foi feito por outros desenvolvedores para poder planejar como realizaria o teste.

> *"eu acho que assim, ela é válida porque ela caracteriza o meio profissional... O meio profissional é isso, você vai trabalhar com testes, se você não tá participando da equipe que desenvolveu... da parte de desenvolvimento de software, engenharia, vai ter essa duplicação porque você tem que entender o... o que foi feito, você tem que estudar pra poder realizar o teste"* (ST7).

Um segundo estudante destacou que além de ser possível o profissional ter que lidar com um projeto similar ao JabRef no mercado, no sentido da falta de documentação, ele também poderá se deparar com a situação de ter que tratar com software de código aberto.

> *"Eu acho assim, que até no mercado de trabalho, a gente pode se deparar com coisas assim (...) ir trabalhar numa empresa e pegar um software assim, sem documentação... um software assim, de comunidade livre"* (ST1).

Finalmente, um terceiro estudante salientou que alguns estudantes ficaram descontentes porque precisaram ser mais autônomos, enquanto prefeririam algo menos independente. Contudo, admitiu que isso não foi um problema para ele, visto que foi preciso desenvolver esse comportamento ao longo de suas experiências no mercado.

> *"Eu sei que... eu tenho um pouco mais de experiência, eu venho trabalhando desde o segundo período, e aí eu não... eu num... eu me acostumei a pegar coisas que eu não vi, ninguém me explicou e quebrar a cabeça sozinho pra fazer. Mas eu sei que muita gente fica chateada quando isso acontece em sala de aula. Quando eles têm que realmente pegar uma situação que é do mundo real e quebrar a cabeça sozinhos. Sozinhos entre aspas, porque vocês estavam dando suporte, mas é um suporte que pra muita gente é superficial (...) tem gente que realmente quer uma coisa mais mastigada ainda"* (ST5).

Dificuldades
Convém ressaltar que a similaridade do projeto de código aberto dos projetos existentes nas empresas e a proximidade das atividades à realidade trazem também dificuldades. Classificamos as dificuldades encontradas relacionadas a estes pontos em: (i) dificuldades provenientes da utilização de projetos reais; (ii) dificuldades provenientes da utilização de projetos de código aberto e (iii) dificuldades na execução dos testes. Enquanto que os dois primeiros independem do assunto sendo tratado através da abordagem de emprego de projetos de código aberto, o último refere-se às dificuldades de implementar testes em projetos reais.

Dificuldades provenientes da utilização de projetos reais
A utilização de projetos reais representa o emprego de projetos maiores e mais complexos. Somente essas características já implicam em alguma dificuldade para os estudantes. Um dos estudantes reportou justamente que teve muita dificuldade para trabalhar com o código grande, dada a dificuldade em entendê-lo.

> *"(...) realmente eu tive muita dificuldade de trabalhar em um código muito grande." (...) "por ele ser um projeto maior você tem essa questão de dificuldade de entender"* (ST7).

Paralelamente, um estudante justificou a dificuldade em realizar a atividade solicitada, devido à falta de experiência com projetos complexos.

> *"(...) pra desempenhar a tarefa... a complexidade do projeto que pra mim era algo... eu não tinha essa experiência..."* (ST8).

O estudante em questão declarou que se tivesse experiências anteriores com outros projetos, "já saberia por onde começar" (ST8). Este foi exatamente o ponto de dificuldade destacado por outro estudante: como começar a trabalhar em um projeto grande, já que não há tempo de entendê-lo por inteiro.

> *"(...) não dava tempo também de ficar tentando entender a lógica dele toda... isso é até uma coisa que eu sempre quis saber, como é que as pessoas chegam numa base, num código gigante e depois... agora eu já sei meu caminho, sei fazer isso"* (ST6).

Outra dificuldade relatada pelos estudantes é a de entender código desenvolvido por terceiros:

> *"Eu mesmo tenho muita dificuldade de entender um código que não foi feito por mim"* (ST7).

> *"(...) a leitura, né, de códigos que não foram feitos... por mim, também é algo mais desafiador..."* (ST8).

Como mencionado anteriormente, alguns estudantes consideraram essa dificuldade como uma das razões para a complexidade do projeto de código aberto utilizado, o JabRef. O mesmo ocorreu para a falta de documentação, como visto, corresponde a uma situação comum em projetos reais. O extrato a seguir ilustra exatamente a dificuldade proveniente da falta de documentação.

> *"... se tiver uma documentação melhor será menos penoso... se houver uma documentação igual a do JabRef e for uma coisa maior, vai ser bem complicado"* (ST4).

Utilização de projetos de código aberto

Como projetos de código aberto correspondem a projetos reais, todas as dificuldades relacionadas aos projetos reais mencionadas são dificuldades que também ocorrem quando OSP são empregados. Todavia, essencialmente, a dificuldade de não existir um suporte dos desenvolvedores já estabelecido é uma situação comum de acontecer quando projetos de código aberto são utilizados. Um dos estudantes relatou exatamente essa dificuldade. Segundo ele, a complexidade dos projetos de código aberto e dos projetos existentes nas empresas podem até ser similares, porém o suporte nas empresas está disponível. Haveria sempre alguém para esclarecer dúvidas ou indicar algum caminho. Quando questionamos que ele poderia ter contatado os desenvolvedores da comunidade, ele assumiu que não o fez por receio deles não terem paciência para respondê-lo.

> ST6: *"Mas como tinha gente lá, (...) eles podiam me guiar pra o que eu... 'ah, eu quero saber onde é isso' (...) Mas então, a complexidade poderia ser a mesma, mas o nível da informação (...) e a facilidade de como chegar em tal lugar... tinha mais fácil, tinha sempre o pessoal trabalhando lá. E aí, o Wiki lá do JabRef... parece até algumas coisas, que não tem explicando o código como é, como é tal parte do código (...)"*
>
> Pesquisadora: *"Mas você não poderia ter entrado em contato também com a comunidade?"*
>
> ST6: *"Podia, mas isso é uma coisa que eu sempre fico... (risos) não sei se eles vão ... 'não tô entendendo aqui'... tal... e eles vão ter paciência de me explicar, será? Não sei. (risos)." (...) "Porque é meio que falta... alguém pegar a mão (risos) e carregar você um pouco, até você entender como é. (...) Inicialmente eu acho que uma pessoa 'aqui, pelo menos essa parte aqui... você vai cuidar dessa parte... isso aqui faz isso...'. Se eu tivesse alguma dúvida, eu teria alguém pra.. sei lá, mostrar como fazer."*

Dificuldades na execução dos testes

Para executar os testes, objetivo de aprendizagem principal da disciplina, os estudantes reportaram uma série de dificuldades geradas pela utilização de um projeto real, como código a ser testado, foram elas: (i) entender o software que seria testado; (ii) identificar

no software, o que deveria ser testado; (iii) descobrir qual classe e método deveria ser testado; (iv) falta de experiência anterior com testes (v) limitações existentes e ajustes necessários para a elaboração de testes automatizados.

Como os estudantes precisavam implementar testes, era essencial que os mesmos entendessem o código do software estabelecido. Todavia, dadas as diversas características já citadas, os estudantes descreveram como um 'desafio' ou 'problemático' entender o JabRef. Um dos estudantes descreveu que para entender como os módulos funcionavam foi preciso fazer algo muito próximo a uma reengenharia.

> *"(...) pra mim o maior desafio foi entender o JabRef. Pegar ele todo e entender como os módulos funcionavam, o que cada classe queria fazer" (...) "é quase uma reengenharia, você vai voltando... 'ah, é pra isso que isso é feito'. A parte principal, que mais deu trabalho foi isso"* (ST1).

Outro estudante reportou a dificuldade em também entender o que fazer, ou seja, 'identificar o que testar'.

> *"(...) foi muito difícil entender o que fazer, as coisas ficaram um pouco nebulosas..."* (ST7)

Essa foi a primeira dificuldade listada por mais um estudante.

> *"Encontrar realmente (risos) o que testar..."* (ST6).

Ao analisarmos os testes implementados pelas equipes, em alguns casos sugerimos outros testes mais significativos. Esse fato pode servir como outro indício da dificuldade dos estudantes de decidir o que testar, quando um software real está sendo tratado.

Após decidir o que testar, a outra dificuldade foi descobrir qual classe e/ou método deveria ser testado. De acordo com os estudantes, foi preciso vasculhar o código, usando depuração (ST7 e ST1) ou funcionalidades de pesquisa (ST2), porque o projeto é grande (ST7), os métodos são extensos (ST2) e não existe separação de interesses (ST1).

> *"A gente teve dificuldade de achar a classe que a gente pretendia testar... o projeto é grande, então vasculhar os pacotes, deu um trabalho (...) você entender a classe ou método, a funcionalidade que você quer testar... às vezes isso envolve outros métodos que estão integrados com ele... você tem que estudar, sair catando, debugando todo o código pra ver onde vai terminar. (...) Então é um trabalho um pouco complicado"* (ST7).

> *"(...) eu demorei acho que uns 2 dias só pra achar... vasculhando o código, assim... pra achar... exatamente a parte do método que eu ia testar (...) A resposta a um evento às vezes não está em uma classe separada, ela está dentro da interface gráfica, aí fica difícil de você achar (...) Eu tinha que depurar o código, parava em um ponto, via a pilha de execução e ia voltando as classes chamadas (...) eu voltava para uma classe, olhava ela, voltava pra outra classe, olhava ela, até achar o trecho de código que eu queria testar. Isso pra mim foi uma dificuldade, não só pra mim, pros meus colegas também..."* (ST1).

> *"São métodos gigantes que você tem que entender (...) da tela até a base. Por exemplo, pra gente encontrar as coisas, a gente teve que sair dando Ctrl + F no software todo, pra encontrar aonde estava definida a... as constantes de textos, pra daí sair procurando uma a uma (risos) e encontrar onde ela era chamada, daí encontrar onde ela era referenciada, pra encontrar onde ela era utilizada de novo (...) e a gente conseguiu achar o que a gente queria"* (ST2).

As dificuldades encontradas na realização das atividades expressas pelas equipes, nos relatórios entregues, foram: (i) entender como as funcionalidades de determinada classe funcionavam; (ii) compreender os padrões utilizados no projeto; (iii) achar todas as classes e métodos dependentes para instanciá-los através de simuladores de objetos (para o uso de *mocks*) e (iv) encontrar a localização dos componentes.

Ao longo da análise, encontramos dois relatos que demonstram a influência da experiência anterior na realização dos testes. Enquanto o primeiro estudante imputou a facilidade em decidir o que testar à sua experiência anterior, o segundo estudante admitiu que a falta de experiência anterior prejudicou a realização da atividade.

> *"Eu particularmente já tinha... já tinha feito muitos testes de unidade. Eu trabalhei muito tempo com testes funcionais também, na tela (...) usando Selenium (...) facilitou muito pra mim analisar o que eu poderia tá testando"* (ST5).

> *"(...) se não me engano tem um ou dois alunos que trabalham na área de testes de software e eles conseguiram fazer tranquilamente. Lógico eles apontaram a dificuldade de entender o código, mas entender como realizar os testes, pra eles foi tranquilo. Mas no meu caso, foi isso, a falta de experiência influenciou muito"* (ST7).

Ainda com relação às dificuldades enfrentadas pelos estudantes na execução dos testes, quando a proposta é codificar testes para um software já existente, que não foi projetado com a intenção da construção de testes, muitas limitações precisam ser defrontadas e ajustes precisam ser feitos. Como esse é o caso de muitos projetos, incluindo o JabRef, os estudantes deparam-se com tais situações, como por exemplo: componentes não nomeados, que dificultou a utilização do AssertJ; atributos finais, que não poderiam ter seus valores alterados; métodos estáticos, que não poderiam ser simulados ("mockados"); métodos que recebiam expressões regulares como parâmetro, entre outros. Finalmente, muitas vezes, era preciso refatorar o código.

A percepção final dos estudantes sobre a execução dos testes em projetos reais que capturamos foi: não é tão simples implementar testes, porque é complexo achar o que deve ser feito e onde deve ser feito; descobertas as respostas a estas questões, a implementação do teste não é complicada. Os extratos a seguir descrevem essa percepção. O último estudante ainda ressaltou que quando o desenvolvedor conhece o projeto, não existe complicação.

> *"(...) porque fazer teste não é simples... não é só, 'ah, pega essa classe aí, vamos lá' ''* (ST4).

> *"Depois que você entende de onde vem cada objeto, de onde vem cada método, funcionalidade implementada, (...) entendendo de onde você tem que partir, o resto desenrola tranquilamente"* (ST7).

> *"Foi simples fazer, foi complexo de achar o que era para fazer (...) ... o tempo que você perde pra encontrar as coisas, (...) pra entender o que é... como as coisas funcionam, o tempo que você leva pra... definir o que é para ser feito, você leva aí mais da metade do tempo... e quando vai realmente fazer é uma besteirinha que você poderia ter feito... se realmente conhecesse o projeto poderia ter feito em duas horas (...) Os testes não eram complicados, os que a gente fez, realmente quando a gente conseguiu passar por todos os problemas foi realmente fácil, os testes"* (ST2).

Visão sobre as habilidades necessárias

O projeto como experiência de mundo real contribuiu para que os estudantes listassem algumas habilidades necessárias aos profissionais da engenharia de software, a saber (ver Figura 7.2): (i) ler código e documentação feito por terceiros; (ii) escrever código e documentação compreensível por terceiros; (iii) adaptar-se aos padrões mesmo informais utilizados pela equipe; (iv) saber trabalhar em equipe; (v) ter uma base de conhecimento teórico; (vi) ter noção das ferramentas existentes; (vii) saber pesquisar e filtrar as informações resultantes; (viii) determinar o que deve ser testado e como deve ser testado; (ix) ter a vivência dos erros mais comuns e (x) pensar em todas as possibilidades. Enquanto as três últimas estão diretamente ligadas ao profissional que vai trabalhar com testes de software, as demais representam habilidades gerais, importantes para qualquer profissional que venha a trabalhar com desenvolvimento ou manutenção de software.

7.2.2 Conexão teoria e prática (Ob6)

Outro objetivo da pesquisa foi perscrutar se a prática com um projeto de código aberto permite que o estudante compreenda como o conhecimento teórico sobre testes é aplicado na prática em um projeto real e se a prática em projetos reais pode ser importante para a aprendizagem dessa teoria. A Figura 7.3 revela as percepções dos estudantes sobre esses aspectos.

As evidências mostram que a partir das atividades do projeto, 86% dos estudantes compreenderam como o conhecimento teórico sobre testes de software é utilizado na execução de um projeto real. Sendo que 93% consideraram que essas atividades revelaram como a prática pode ser importante para a aprendizagem sobre testes de software.

Por meio das entrevistas, capturamos a percepção dos estudantes sobre a associação entre a teoria e a prática ao realizar as atividades. A Figura 7.4 ilustra as categorias encontradas. As categorias expressam a importância da prática para a teoria, da teoria para a prática e os problemas consequentes de falhas nessa conexão.

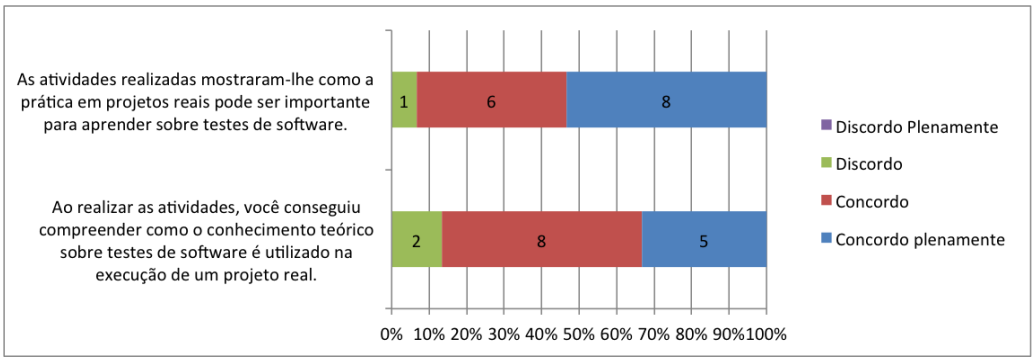

Figura 7.3 Compreensão dos estudantes sobre a conexão entre a teoria e a prática em projetos reais no estudo EC2.

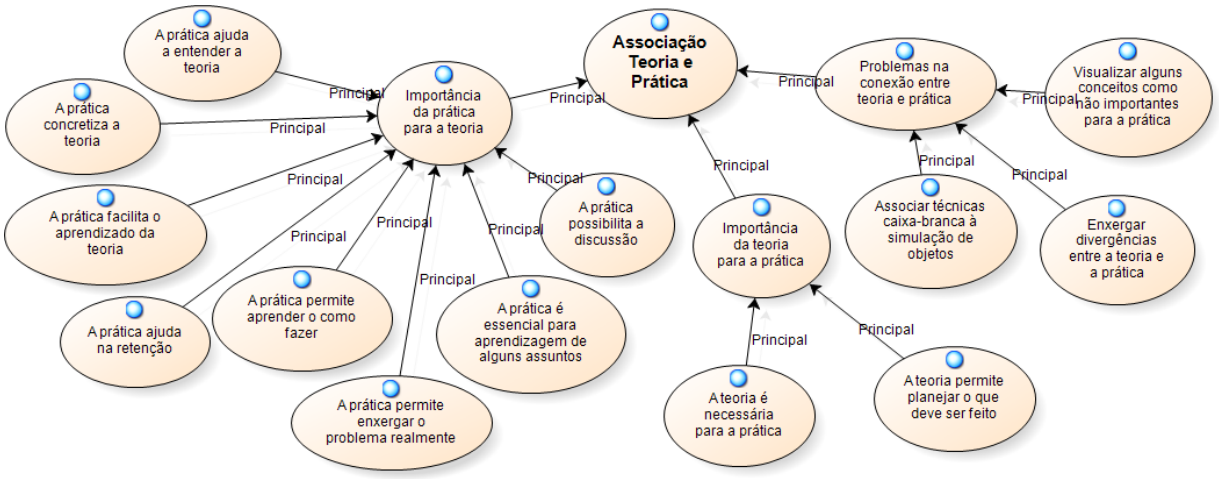

Figura 7.4 Percepção dos estudantes sobre a conexão teoria e prática no estudo EC2.

Segundo os estudantes, por meio da prática eles conseguiram 'entender melhor os conceitos' de testes caixa-branca, caixa-preta, etc. Exatamente por permitir visualizar através dos testes elaborados, as distinções entre os diferentes tipos de testes.

> *"(...) a prática ajudou muito a entender os... os tipos de testes, caixa branca, caixa preta, os testes de interface, assim... você ter uma visão mais dos testes... integração..."* (ST1).

> *" Os testes eles clarificam mais, quando você olha o que você fez e vê como encaixou naquilo (...) quando você tem uma gama de testes para diferenciar um grupo de testes de outro grupo, como por exemplo, os testes de... de integração e os testes unitários. (...) Quando você só tem uma explicação, você não... 'ah, é isso, e é uma coisa muito parecida com isso, que tem uma questão diferente'. (...) quando você começa a escrever um teste, você já... 'ah, tá, estou precisando fazer isso, então isso aqui já é um teste de integração', 'não tô precisando disso então é um teste unitário', 'vou precisar fazer isso, é um teste de interface'. Você já sai diferenciando por grupos... Isso meio que enriquece a... o seu conhecimento teórico"* (ST2).

Para um dos estudantes, o aprendiz, ao ter contato com a teoria, fica apenas imaginando como seria na realidade. Com a prática, o conteúdo se fortalece porque se 'concretiza'. O estudante consegue enxergar como tudo acontece.

> *"Você vê só na matéria, você... vê o assunto, estuda e imagina, né... mas quando você vê na prática... fortalece mais"* (ST1).

Possivelmente, são essas as razões que levaram outro estudante comentar que tem dificuldade de aprender quando a prática não acompanha a apresentação do conteúdo teórico, ou seja, que a prática 'facilita o aprendizado'.

> *"(...) eu tenho dificuldade de aprender só lendo, só teoria, sem fazer uma prática'* (ST3).

Alguns estudantes destacaram o quanto a prática em um projeto é importante para a 'retenção do conteúdo'. Mais significativo que estudar para uma prova, já que nesse último caso, em pouco tempo, o conteúdo é esquecido. Um deles explicou que a retenção acontece, porque é preciso entender o conteúdo para poder fazer a atividade.

> *"Porque só ficar na teoria ia ficar meio, a pessoa daqui a um mês esquece, mas na prática não, quando precisar vai tá ali"* (ST3).

> *"(...) digamos que... fixa melhor o conhecimento. Porque se der algo mais material pra você guardar o que foi visto em sala, não fica apenas pra prova, por exemplo"* (ST5).

> *"Na teoria de DS III, foi bom pra fixar porque... você vê mil conceitos, mil conceitos, mil conceitos e você fica lá voando, às vezes... aí depois você tem que lembrar o que é um teste de unidade, o que é um teste de regressão... vou ter que fazer um (...) como tem que fazer, você tem que entender, então... quer queira ou não, fazer me ajudou a fixar tudo aquilo"* (ST4).

De acordo com um estudante, existem questões que são reveladas apenas quando se tenta fazer, ou seja, somente a prática permite 'aprender realmente como fazer'.

> *"(...) eu fiz minha primeira versão do código, mas, segundo a senhora avaliou, não estava cem por cento... como se implementa bem essa questão dos caminhos, seria um caso pra cada caminho. Então eu acho que foi bom dessa parte... dessa perspectiva, pro aprendizado, pra ver que existem questões práticas, que nem sempre são fáceis de se ver, como vendo só a teoria"* (ST8).

Outro estudante realçou que somente com a prática surgem as dúvidas.

> *"Você entende visualmente, mas quando você põe em prática, o que foi visto em sala, aí começam a surgir as dúvidas"* (ST7).

No extrato a seguir, o estudante assumiu uma postura mais crítica no que diz respeito ao método que usualmente ocorre, onde apenas o conceito teórico é repassado, sem que os estudantes aprendam realmente o como fazer. Falta prática, exemplos e exercícios, para aprender como fazer.

> *"(...) nenhuma matéria até hoje tinha... tocado sobre isso e... quando toca, simplesmente fala 'existe teste unitário que você testa a unidade', uma coisa que não é muito... 'ok, você tá testando uma unidade', mas como é realizado isso? De que forma é realizado?"* (ST6).

Os estudantes ainda revelaram que, na teoria, tudo é mais simples. Portanto, a prática é essencial para 'enxergar o problema realmente', principalmente quando um projeto real é envolvido.

> *"Eu achei que a teoria dá muito a desejar, porque você só vê o problema quando você está com o problema na mão (ter que testar um software). Ter a teoria de como faz os testes, do que é necessário, é bonitinho, mas você só vai saber realmente o que é necessário, quando você tiver lá na situação (...) nada vai ser como você sentar lá e fazer um. Então, acho que a teoria é boa, mas ter a prática vale a pena"* (ST2).

> *"(...) porque na teoria fica sempre tudo mais simples (...) Os exemplos que a gente pega são sempre simples... é um número inteiro, então, um número entre zero e dez. Agora isso daí não são números [comentando sobre o projeto], é uma string, tem uma base de dados que, com uma entrada, com outra coisa como entrada, então fica... foi difícil pra saber... o que da teoria usar"* (ST6).

Se a prática permite aprender o como fazer e se somente ela possibilita enxergar o problema realmente, ela torna-se 'essencial para aprendizagem de alguns assuntos', como o de testes de software, por exemplo. Presumidamente, seriam essas as razões que o estudante, a seguir, assumiu que, sem prática, o aprendizado sobre testes seria insuficiente.

> *"(...) acho que uma matéria de testes sem prática pra mim seria... não seria boa, acho que... aprenderia pela metade..."* (ST1).

A última categoria identificada sobre a influência da prática realça a importância da prática com projetos, para a realização de discussões sobre a teoria, isto é, poderíamos dizer que 'a prática possibilita a discussão'. A partir do caso prático, cada estudante tem a sua experiência e elabora o seu ponto de vista, acrescenta-se que, como todos os estudantes experimentam um pouco do projeto, diferentes perspectivas podem ser expostas, favorecendo a discussão.

> *"(...) você faz, implementa as coisas, vê e tem sua opinião, quando você vê a de outro também, você aumenta... você vê mais coisas também, que você não viu e a outra pessoa viu"* (ST1).

> *"Acho que contribuiu bastante, a questão das perspectivas diferentes a respeito do projeto..."* (ST8).

Importância da teoria para a prática

Por outro lado, como apresentado na Figura 7.4, detectamos categorias que expressam que a teoria também é importante para a prática. Os estudantes reconheceram a necessidade da teoria para realizar a prática, inclusive para planejar o que deve ser feito.

Dois estudantes relataram que para fazer a prática às vezes é necessário revisar ou buscar mais conhecimento teórico.

> *"Você não tem como fazer um trabalho de teste sem saber a teoria... mesmo que você não tenha aprendido direito antes na disciplina, você volta, porque você tem que ver lá todas as estratégias, estudar, implementar (...) prática sem teoria não existe... pra mim é importante"* (ST1).

> *"(...) quando eu realmente fiz a prática, exigiu que eu viesse buscar mais conhecimento em relação àquele assunto, determinado ponto..."* (ST3).

Outros dois estudantes acrescentam que a teoria é importante para identificar o que e como deve ser feito.

> *"Eu tinha que retomar o que eu vi em sala, pra saber o que tinha que ser feito: 'ah, eu tenho que fazer tal coisa, com tal classe JabRef... Como acessa aquela classe, aquele método? Como eu testo ele?' "* (ST4).

> *" Porque quando você faz o teste, você não... às vezes você não sabe exatamente o que você está fazendo ali... você está fazendo um teste, para testar. Quando você tem a base teórica... você precisa fazer o planejamento para fazer os testes e aí você já tem uma noção do que você está fazendo, então ajuda... ajuda bastante, você ter uma ideia de como é que você está testando"* (ST2).

Neste último extrato, o estudante realçou que se o profissional não conhece a teoria, ele testa só por testar. Porém quando conhece a teoria, ele sabe planejar o que e como deve ser testado.

Todos esses achados ilustram como o emprego do projeto de código aberto possibilita, pela prática proporcionada, particularmente a partir de um projeto real, que o estudante faça a conexão do que é visto na teoria com a prática (Ob6).

Problemas na conexão entre teoria e prática

Contudo, apesar do favorecimento à aprendizagem descrito até o momento e os resultados diretamente ligados à aprendizagem que serão apresentados em 7.2.3, as entrevistas também apontaram alguns problemas na conexão entre a teoria e a prática, por conta da especificidade deste estudo de caso. Esses problemas levaram os estudantes a: (i) visualizar alguns conceitos como não importantes para a prática; (ii) enxergar divergências entre a teoria e a prática e (iii) associar técnicas caixa-branca à simulação de objetos (ver Figura 7.4).

No primeiro caso, um estudante não conseguiu visualizar a importância dos conceitos de teste unitário e teste de integração, principalmente para softwares existentes.

> *"É complicado (risos) que às vezes parece que tem muita coisa que a gente aprende que não... que não serve pra muita coisa, na hora que for fazer... A parte da teoria diz o que é um teste unitário, o que é um teste de integração... essa parte pelo menos, de ler assim, simplesmente, não... não sei se adianta muita coisa não, principalmente para um projeto que já esteja andando..."* (ST6).

Como para a implementação dos testes, solicitamos que os estudantes documentassem qual era o teste que haviam implementado, o estudante em questão, dada a falha de conexão mencionada, relatou a dificuldade em especificar qual o teste que estava executando. O estudante admitiu ainda que o problema possivelmente ocorreu em virtude da falta de exemplos prévios.

> *"(...) na hora que você vai botar a mão na massa, não parece ajudar muito, você saber qual a diferença 'ah teste unitário testa a unidade, teste de integração faz isso'... na hora de fazer, acho que não dá... Mas acho que, o que dificultou foi eu nunca ter feito... logicamente eu nunca ter feito... nunca ter tido contato com isso... nem mesmo, assim... algum código, algum exemplo. Vamos dizer assim, foi meio teórico"* (ST6).

Portanto, pelos poucos exemplos ou pela pouca significância dos exemplos apresentados previamente, o estudante em questão não conseguiu compreender que os testes de unidade são importantes para isolar o código sendo testado e assim, facilitar a localização de erros e garantir que aquela parte do código funciona corretamente. Assim como não pôde compreender que os testes de integração vão garantir que esse código agora integrado às outras partes do código continua funcionando corretamente.

O segundo problema na conexão entre teoria e prática ocorreu porque alguns estudantes acreditavam que somente técnicas caixa-branca poderiam ser empregadas para a elaboração dos casos de teste no JabRef, já que seus requisitos não estavam documentados.

> *"(...) a técnica que eu escolhi foi a técnica dos caminhos, né... porque eu acreditei que seria a única válida, porque eu não tinha nenhuma especificação funcional do sistema, de como deveria funcionar cada funcionalidade. Então, eu tinha um código e eu acho que seria a que eu poderia aplicar..."* (ST8).

Um dos estudantes, com experiência na indústria de software, chegou a descrever este problema como uma divergência entre a teoria e a prática.

> *"É... tem algumas divergências, mas que são realmente comuns também no mundo real. Por exemplo, a gente não tem documentação. Toda teoria geralmente, a gente se baseia muito nos requisitos pra saber o que a gente tá fazendo, saber quais são os testes, quais são... qual que é o jeito correto de tá... acontecendo no sistema, e a gente não tinha e... isso difere muito da teoria pra prática"* (ST5).

Para o estudante, como no mundo real, a maioria dos projetos não é documentada e muitas atividades dependem teoricamente dos requisitos, a teoria não pode ser aplicada na prática, a exemplo da aplicação das técnicas caixa-preta para elaboração de casos de testes. A conexão teoria e prática falhou nesse caso, porque os estudantes não entenderam que os requisitos também podem ser recuperados a partir do código, particularmente, que técnicas caixa-preta também podem ser aplicadas a partir da assinatura dos métodos.

No terceiro e último problema detectado, o estudante, desejando realizar um teste de unidade, precisava simular os objetos dos quais o método a ser testado dependia. Como para identificar esses objetos era preciso analisar o código, o estudante pensou que estaria executando um teste tipo caixa-branca.

> *"Eu acho que teve muito pouco, muito pouca interação, no caso, com a relação de testes à prática (...) por exemplo, o que seria um teste de caixa-branca? (...) você realiza um teste caixa-branca pra saber ... quais são os objetos que vão ser mockados (...) Eu achei que ali, eu estaria fazendo um teste caixa-branca (...) Então ficou meio nebuloso da teoria pra prática"* (ST7).

Isso demonstra que alguns estudantes podem não ter entendido que as técnicas caixa-branca ou caixa-preta servem para especificar os casos de testes, ou seja, se o corpo do

método será analisado ou não, respectivamente, para definir os dados que serão utilizados para testá-lo. As configurações necessárias para realizar testes de unidade ou integração independem dessas técnicas.

Convém ressaltar que estes problemas acarretaram em dificuldades na realização dos testes, especificamente, categorizamo-las em 'Dificuldades de conexão da teoria com a prática', divididas em 'Diferenciação entre teste unitário e teste de integração' e 'Diferenciação entre testes caixa-branca e testes caixa-preta'. Essas dificuldades geraram prejuízos no alcance de alguns objetivos de aprendizagem, conforme apresentaremos a seguir.

7.2.3 Objetivos de Aprendizagem Alcançados (Ob4)

Avaliar se a abordagem propicia o alcance de objetivos de aprendizagem é um dos principais objetivos de nossa pesquisa. Afinal, a aprendizagem é o fim desejado, qualquer que seja o recurso utilizado por um professor. A partir dos objetivos de aprendizagem estabelecidos no Capítulo 3, examinamos quais seriam mais adequados a serem tratados pelo estudo de caso, dado que estes precisavam estar também de acordo com o programa e planejamento do professor da disciplina. Sendo assim, selecionamos os objetivos de aprendizagem relacionados a testes de software, bem como, competências associadas à prática profissional que poderiam ser beneficiadas pela aplicação da abordagem. Consequentemente, dividimos a apresentação dos resultados em aprendizagem de aspectos técnicos e desenvolvimento de competências da prática profissional.

Aprendizagem de aspectos técnicos

Os objetivos de aprendizagem associados a testes de software que investigamos encontram-se enumerados na Tabela 7.5. A tabela também apresenta a comparação entre a percepção dos estudantes sobre a sua capacidade considerando estes objetivos após o emprego do método tradicional de ensino, com aulas expositivas, exercícios e prova, com a sua percepção após a execução das atividades práticas com o emprego do OSP. Comparando os resultados, detectamos uma melhora das medianas para 11 objetivos de aprendizagem e uma diferença positiva significativa para nove objetivos [3].

Tabela 7.5: Comparação entre a aprendizagem após o método tradicional de ensino e após o emprego da abordagem com OSP (EC2).

Id.	**Objetivos de aprendizagem específicos sobre testes de software**	**Mediana**		**Soma de postos** positivos	***Valor-p***
		Trad.	OSP		
LO90	Capacidade de descrever os conceitos e práticas relacionadas ao teste de software.	1	1	13,0	0,2975
LO95	Capacidade de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).	1	2	15,0	0,0170*
LO98a	Capacidade de descrever e distinguir entre as diferentes técnicas caixa-preta para elaboração de casos de teste.	1	1	20,0	0,1505
LO98b	Capacidade de descrever e distinguir entre as diferentes técnicas caixa-branca para elaboração de casos de teste.	1	1	2,5	0,0785

[3]Dado que os dados não apresentavam uma distribuição normal, aplicamos o teste de postos sinalizados de Wilcoxon para amostras pareadas.

LO99	Capacidade de criar testes de unidade que não sejam redundantes (aplicar as técnicas de elaboração de casos de teste de modo que os casos gerados não sejam redundantes).	1	1	15,5	0,1440
LO104	Capacidade de planejar e gerar casos de testes para softwares de médio porte.	-1	1	26,0	0,0185*
LO101	Capacidade de criar e documentar um conjunto de testes para um segmento de código de médio porte.	-1	1	41,0	0,0125*
LO107	Capacidade de usar ferramentas de testes.	1	1	15,0	0,0190*
LO91	Entender a diferença entre testar pequenos programas e testar software de grande porte.	1	2	10,0	0,0295*
LO93	Capacidade de avaliar uma suíte de testes para um segmento de código de médio porte.	-1	1	34,0	0,0115*
LO88	Capacidade de avaliar os testes sendo executados por meio de métricas.	-1	1	37,0	0,0375*
LO109	Capacidade de usar ferramentas de teste de cobertura eficientemente.	-1	1	57,0	0,0130*
LO287	Capacidade de descrever o processo de testes de regressão e seu papel no gerenciamento de releases.	1	1	40,0	0,2625
LO103	Capacidade de descrever como selecionar bons testes de regressão e automatiza-los.	-1	1	28,0	0,0715
LO110	Ter algum conhecimento sobre ferramentas de integração continua e testes de regressão.	-1	1	35,0	0,0075*
LO105	Capacidade de descrever como é um processo de gerenciamento de defeitos.	-1	1	29,5	0,4900
LO198	Capacidade de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (usar a ferramenta para reportar erros, monitorar os erros e submeter correções).	-1	1	49,0	0,0735

(*) Representa resultado significativo, para nível de significância de 0,05, unidirecional.

Fazendo uma reflexão sobre esses resultados, podemos dizer que a melhora significativa ocorreu principalmente para os objetivos de aprendizagem relacionados à competência de 'saber fazer'[4], ou seja, para as competências de saber planejar e gerar casos de testes para softwares de médio porte (LO104), saber criar e documentar um conjunto de testes para um segmento de código de médio porte (LO101), saber usar ferramentas de testes (LO107) e saber usar ferramentas de teste de cobertura eficientemente (LO109). Bem como, para situações onde a visão do projeto como um todo é essencial, as quais são: descrever e distinguir entre tipos e níveis diferentes de testes (LO95), entender a diferença entre testar pequenos programas e testar software de grande porte (LO91), avaliar uma suíte de testes para um segmento de código de médio porte (LO93) e avaliar os testes sendo executados por meio de métricas (LO88). Apesar da melhora das medianas no nível de concordância, as evidências não foram tão significativas para as capacidades de descrever como selecionar bons testes de regressão e automatiza-los (LO103), descrever como é um processo de gerenciamento de defeitos (LO105) e de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (LO198), sinalizando alguma deficiência na implementação da abordagem, seja pela pouca profundidade na discussão dos dois primeiros tópicos, seja pela não exigência que os estudantes reportassem os defeitos encontrados, para o terceiro, ou ainda, porque não houve

[4]Correspondente aos níveis de 'uso' proposto no CS2013 e 'aplicação' da Taxonomia de Bloom revisadas no Capítulo 3.

uma cobrança individualizada sobre as atividades que buscavam cobrir esses objetivos de aprendizagem. Particularmente para o objetivo LO105, as equipes apresentaram boas respostas para as questões que tratavam do processo de gerenciamento de defeitos nos trabalhos. Reforçando assim, nossa última suposição.

Com exceção do objetivo correspondente à capacidade de criar testes de unidade que não sejam redundantes (LO99), os demais objetivos, onde não obtivemos evidências de melhoras, estão relacionados às competências de 'familiaridade'(CS2013) ou 'entendendimento' (Taxonomia de Bloom). Os resultados alcançados demonstram que o método tradicional de ensino supre satisfatoriamente o alcance dessas competências.

Numa análise descritiva, percebemos que os estudantes, em sua grande maioria, concordaram com a contribuição da execução do projeto para a sua aprendizagem, para todos os objetivos de aprendizagem tratados pela abordagem. Conforme visualizamos na Figura 7.5, as contribuições mais expressivas foram novamente para os objetivos relacionados à competência de 'saber fazer' (planejar e gerar casos de testes (LO104), criar e documentar um conjunto de testes (LO101), usar ferramentas de testes (LO107) e criar testes de unidade que não sejam redundantes (LO99) – realçado, desta vez). Outro destaque foi a percepção dos estudantes sobre a contribuição do projeto para que entendessem a diferença entre testar pequenos programas e testar software de grande porte (LO91). Nesse caso, o resultado era esperado, uma vez que, usualmente os estudantes lidam apenas com programas pequenos, nos exemplos e exercícios empregados no método tradicional.

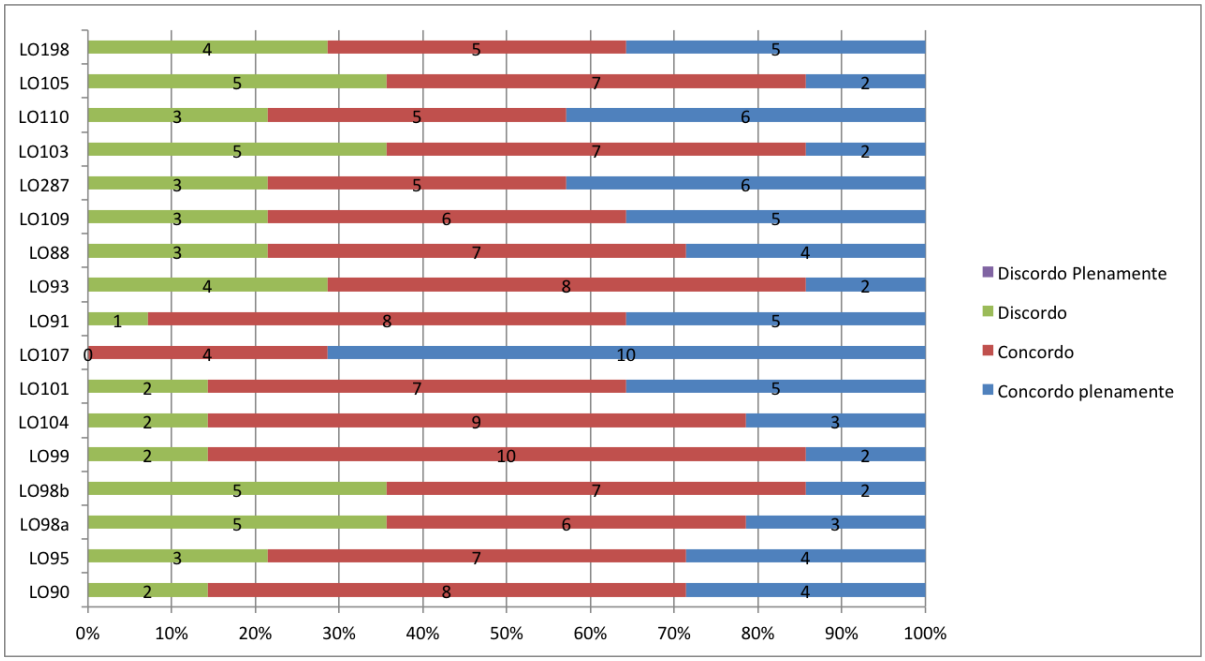

Figura 7.5 Contribuição da abordagem para a aprendizagem sobre testes de software.

Em minoria, os participantes também expressaram discordâncias. Os objetivos de aprendizagem relacionados às capacidades de descrever como selecionar bons testes de

regressão (LO103), descrever como é um processo de gerenciamento de defeitos (LO105) e de usar uma ferramenta de rastreamento de defeitos (LO198), discutidos anteriormente, aparecem com um maior número de discordâncias, bem como os objetivos relacionados às capacidades de avaliar uma suíte de testes para um segmento de código de médio porte (LO93) e de descrever e distinguir tanto entre as diferentes técnicas caixa-preta (LO98a) quanto as técnicas caixa-branca (LO98b) para elaboração de casos de teste.

Para a capacidade de avaliar uma suíte de testes para um segmento de código de médio porte (LO93), apesar da melhora significativa após a aplicação da abordagem, quatro estudantes discordaram que a mesma tenha contribuído para o alcance do objetivo. Comparando o conhecimento antes e depois da experiência com o projeto de código aberto, para cada um destes estudantes, verificamos que seus níveis de discordância mantiveram-se constantes.

Como vimos em 7.2.2, a conexão teoria e prática foi importante para a aprendizagem dos conceitos envolvidos. Apesar de não ter ficado totalmente claro para todos os estudantes, em virtude dos problemas identificados para esta conexão, discutidos na subseção citada, a maioria, 64,3%, concordou que a abordagem contribuiu para que os mesmos fossem capazes de descrever e distinguir tanto entre as diferentes técnicas caixa-preta (LO98a) quanto as técnicas caixa-branca (LO98b) para elaboração de casos de teste. Enquanto que 78,6% concordaram que a abordagem contribuiu para que os mesmos fossem capazes de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação) (LO95).

Os relatos durante as entrevistas também retrataram essa aprendizagem. No extrato a seguir, o estudante acrescentou à aprendizagem dos conceitos, a análise de qual tipo de teste seria mais adequado para o projeto em mãos.

> *"Acho que aprendi o que é teste de unidade, teste de regressão (risos)... aprendi algumas métricas de teste... se o software é assim, então tal tipo de teste, não é tão útil, porque não é o foco dele, sei lá... JabRef, por exemplo, a gente meio que viu que teste de performance não seria o caso..."* (ST4).

Além dos conceitos relacionados diretamente aos testes, os estudantes também relataram a aprendizagem a respeito de projetos de código aberto, propriamente ditos.

> *"(...) sem dúvida, contribuiu muito para o meu aprendizado na questão de poder compreender o que é um projeto open source..."* (ST3).

> *"(...) ter esse pequeno vislumbre de como é trabalhar numa comunidade open source"* (ST4).

> *"(...) eu nunca peguei um código assim tão... que fosse feito por uma comunidade aberta... pra mexer, sempre já vi, já vi algumas partes mas nunca meti a mão na massa..."* (ST6).

Entretanto, mais importante do que a aprendizagem sobre conceitos, os estudantes reportaram a aprendizagem de 'saber fazer', isto é, o desenvolvimento das habilidades de como testar e de usar as ferramentas e *frameworks*.

Dois estudantes resumiram o seu desenvolvimento com relação à habilidade de testar um software, ou seja, de como implementar e executar o teste.

> *"Contribuição do projeto pra minha aprendizagem em teste de software... eu acho que a aplicação direta dos testes." (...) "Aprendi os testes, a testar, a questão de testes automatizados"* (ST1).

> *"Foi realmente saber como é que faz um teste (risos)" (...) "saí do zero, saí da estaca de... 'ah, eu quero ajudar, eu quero... pensar em fazer um teste', pelo menos eu já sei como fazer, como rodar"* (ST6).

Todos os estudantes entrevistados destacaram a aprendizagem sobre ferramentas e de como usar essas ferramentas, ao serem questionados sobre sua experiência de aprendizagem realizando o trabalho. Este resultado corrobora o resultado das respostas do questionário quanto aos objetivos de aprendizagem relacionados a saber usar ferramentas de testes (LO107), ferramentas de teste de cobertura (LO109) e ter algum conhecimento sobre ferramentas de integração continua e testes de regressão (LO110) (Figura 7.5)

Os relatos demonstram a aprendizagem de diversas ferramentas, desde o uso do IDE (*Interface Development Environment*), com funcionalidades de depuração de código, os próprios *frameworks* de testes como o JUnit e AssertJ, às ferramentas de análise de cobertura, EclEmma, e simulação de objetos, Mockito; além do Jenkins, que permite a configuração de um conjunto de testes para serem executados como testes de regressão. Até mesmo o Git foi citado como aperfeiçoamento com relação à aprendizagem, apesar de ser uma ferramenta de gerenciamento de configuração importante para todo o processo de construção do software, não só para testes.

Ilustrando esses achados, no extrato a seguir, o estudante realçou a diversidade de ferramentas utilizadas, inclusive de ferramentas não conhecidas. Destacou ainda que através do projeto, ele conseguiu entender os testes de regressão no Jenkins, situação vista, mas não compreendida, em um projeto anterior.

> *"Na parte de... conhecimento de ferramentas... foi bom também porque a gente usou algumas ferramentas... na verdade várias ferramentas, de IDE até o AssertJ, o JUnit, foi importante, o Jenkins, o Eclemma, coisas que eu não conhecia... foi importante utilizar, ter uma noção. Eu já tinha visto, por exemplo, no caso do Jenkins, eu já tinha visto esse teste de regressão mas não sabia o que era. Eu já vi um software fazer aquilo mas eu não sabia o que ele tava fazendo, entendeu. Aí quando eu vi o Jenkins, que eu vi a imagem daquele software, eu lembrei daquele teste, aí eu entendi o que ele tava fazendo..."* (ST1).

Um estudante revelou que aprofundou seus conhecimentos na ferramenta de depuração.

> *"(...) eu precisei usar bem mais a ferramenta de debug, aprendi muito mais como ela funciona, de forma mais profunda"* (ST8).

Enquanto que um estudante enfatizou o JUnit e AssertJ, outros dois estudantes frisaram as ferramentas para testes de regressão e análise de cobertura. Sendo que um destes, ainda apresentou certa surpresa pela existência de outras ferramentas além do JUnit (o único que conhecia), que facilitavam a execução e análise dos testes.

> *"(...) vi o JUnit, direitinho, como é... também o AssertJ... essas coisas (...) ferramentas boas... pra você poder realmente fazer um teste de software direitinho"* (ST4).

> *"Ferramentas de teste... pra teste de... de regressão, que eu achei muito interessante, também teve a de cobertura"* (ST3).

> *"(...) algumas ferramentas também, do... que eu nunca tinha usado: Jenkins... Eclemma (...) ferramentas de testes em geral (...) eu sabia JUnit [da existência do JUnit] e achava que era só feito com isso, e não tinha outras ferramentas que você pudesse usar pra fazer relatório"* (ST6).

Mesmo estudantes com alguma experiência com testes de software, apontaram a aprendizagem de algumas ferramentas. Um deles comentou sobre capturar as especificidades de como usar o AssertJ e no que diz respeito ao Mockito, afirmou ter sido interessante aprender a utilizá-lo, por ter proporcionado uma experiência de testes onde existe dependência entre funcionalidades.

> *"Eu nunca tinha realmente feito um teste com o AssertJ, foi a primeira vez que eu fiz (...) eu realmente aprendi como utilizar o básico dele. Se eu tiver uma aplicação que não seja muito complexa em termos de telas (...) eu consigo utilizar ele relativamente bem, porque eu já descobri as manhas dele"* (ST2).

> *"Utilizar o Mockito foi algo novo pra mim... nunca precisei utilizar. Normalmente, as coisas... os testes que eu fazia eram testes mais... de uma funcionalidade que não depende de outras, então esses testes de... interdependência nunca foram necessários. Como... Aprender a utilizar o Mockito foi algo interessante"* (ST2).

Outro estudante experiente acrescentou ao AssertJ e Mockito, a ferramenta de análise de cobertura como algo totalmente novo pra ele.

> *"(...) eu aprendi muitas ferramentas específicas, que eu nunca tinha visto. Por exemplo, o AssertJ (risos), um pouco do Mockito também, que... pra decidir não usar ele (risos), eu precisei ler um pouco sobre ele. Então, eu aprendi algumas ferramentas que eu nunca tinha visto, nem ouvido falar. A de... cobertura também, análise de cobertura. Já tinha usado o Jenkins, mas a análise de cobertura foi nova também"* (ST5).

Ainda considerando a aprendizagem sobre o uso de ferramentas, a última relatada foi o Git. Enquanto três estudantes realçaram que o projeto possibilitou o aprofundamento da utilização do software de gerenciamento de configuração pela execução de comandos de *branch*, *checkout* e *pull request*. Um estudante admitiu que finalmente aprendeu a utilizá-lo e outro estudante comentou sobre algum aprendizado sobre a ferramenta com o projeto, admitindo que apesar da importância da ferramenta, seu conhecimento sobre a mesma era deficiente.

> *"A gente tem contato com o Git mas não naquela profundidade, de branch, pull request, checkout.."* (ST1).

> *"(...) eu aprendi mais... a mexer... melhorar, aperfeiçoar na questão de mexer em ferramenta de gerência de configuração"* (ST3).

> *"(...) como fazer um pull request pra galera"* (ST6).

> *"Aprendi usar o Git, finalmente, depois de vários anos (risos)"* (ST4).

> *"(...) utilizar um pouco do Git, que é realmente uma falha minha (...) uma ferramenta de versionamento (...) porque eu sei que é algo muito fundamental"* (ST7).

Através desses extratos, podemos dizer que, além dos objetivos de aprendizagem particularmente relacionados ao uso de ferramentas de testes (LO107, LO109 e LO110) terem sido alcançados, os objetivos relacionados à capacidade de usar um IDE moderno (LO82), ferramentas de depuração de código (LO108) e gerenciamento de configuração (LO133 e LO134) em suporte ao desenvolvimento de um software de médio porte também foram atingidos.

Ainda para validar o desenvolvimento das habilidades de saber fazer', analisamos os códigos de teste implementados pelos estudantes. O resultado da produção por equipe encontra-se resumido na Tabela 7.6. Na tabela, descrevemos o número de membros da equipe que participou do projeto, a quantidade de métodos testados, a quantidade de métodos de testes implementados, o total de linhas de código dos métodos de testes implementados, a quantidade de linhas de código do software que foram acrescentados ou refatorados para a realização dos testes, a cobertura média do código dos métodos testados e finalmente, alguns comentários associados.

Ao todo, 16 estudantes implementaram 84 métodos para testar 27 métodos do JabRef. Com exceção da equipe ET1, as demais equipes conseguiram uma cobertura[5] razoável para os métodos sendo testados. Convém ressaltar que nas equipes ET1 e ET3, apenas um membro terminou participando do projeto, sendo que, enquanto o membro da ET3 dedicou-se desde o início ao projeto, tirando diversas dúvidas, o membro da equipe ET1 só entregou o trabalho após a extensão do prazo, além de que, não participou da segunda parte do projeto.

[5]Contabilizamos a cobertura medida pelo EclEmma (http://www.eclemma.org/) do método sendo testado sem considerar os métodos chamados pelo mesmo.

As equipes ET4 e ET6, apesar do grande número de linhas de código de testes implementados, não obtiveram uma cobertura tão boa. Essa grande quantidade de linhas de código não só implementa vários casos de teste, mas também configura o ambiente para que esses testes sejam executados. Consequentemente, essas informações comprovam o esforço despendido pelas equipes na execução da atividade.

Outra observação relevante refere-se ao comentário recorrente para as equipes de que os métodos de teste implementam mais de um caso de teste. Esse fato justifica o número reduzido de métodos de testes implementados, principalmente para as equipes ET6 e ET7, e indica que vários estudantes não seguiram a recomendação da comunidade do projeto de que cada método implementasse um único caso de teste. Além desse problema, a análise do código apontou algumas falhas e outros problemas encontrados. Estes se encontram contabilizados na Tabela 7.7, de acordo com o número de ocorrências.

Tabela 7.6: Resumo da produção de testes por equipe

Equipe	**Num. membros**	**Quant. métodos testados**	**Quantidade de métodos de testes implementados**	**LOC Testes**	**LOC modif.**	**Cobertura média dos testes**	**Comentário**
ET1	1	2	4	203	0	35,85%	
ET2	5	7	20	455	221	93,24%	Os métodos de testes estão implementando mais de um caso de teste.
ET3	1	2	13	354	0	95,85%	Os métodos de testes estão implementando mais de um caso de teste.
ET4	4	8	29	1342	59	75,33%	Os métodos de testes estão implementando mais de um caso de teste. Configuração do ambiente para o teste.
ET6	3	5	8	809	11	66,1%	Os métodos de testes estão implementando mais de um caso de teste. Configuração do ambiente para o teste.
ET7	2	3	10	212	0	100%	Os métodos de testes estão implementando mais de um caso de teste.
Total	16	27	84	3375	291	-	

Convém ressaltar que os problemas identificados na estruturação dos testes não impediram a execução dos mesmos. Na verdade, representam indicações de melhorias para a prática de testes automatizados. Sendo assim, os estudantes deveriam ter procurado não utilizar instruções consideradas obsoletas, poderiam ter otimizado a configuração inicial necessária para os testes, deveriam implementar apenas um caso de teste em cada método

Tabela 7.7 Falhas e problemas encontrados nos testes implementados

Falhas encontradas	**Quantidade**
Falhas conceituais	
Falha na especificação do nível de teste empregado	3
Faltou especificar a técnica aplicada	18
Falha na especificação da técnica de teste empregada	3
Testes redundantes	2
Outros casos de testes poderiam ser criados	15
Problemas na estruturação dos testes	
Emprego de instrução obsoleta	1
Configuração inicial do ambiente dos testes	6
Um método de teste implementado com vários casos de teste	16
Utilização de nomes não significativos	33

de teste e, por fim, utilizar nomes significativos para os métodos de testes implementados.

Por outro lado, encontramos algumas falhas conceituais que também não impedem a execução dos testes, todavia, podem diminuir a sua efetividade em encontrar os erros ou aumentar os custos para a sua implementação. Quando o estudante não compreende a diferença entre construir testes de unidade e integração (falhas na especificação do nível de teste empregado), ele termina não isolando os testes adequadamente, o que facilitaria a descoberta de erros. Do mesmo modo, se o estudante não entende os objetivos e como aplicar as técnicas de caixa-branca ou caixa-preta (falta da especificação da técnica aplicada e falha na especificação da técnica de teste empregada), maiores são as chances de que nem todos os casos de testes sejam elaborados ou que casos de testes redundantes sejam construídos.

Alguns extratos demonstram que os estudantes tiveram dúvidas de como realmente aplicar as técnicas de caixa-branca e caixa-preta, possivelmente ocasionando as falhas contabilizadas na Tabela 7.7, para a falta de detalhamento da técnica aplicada, erro na especificação da técnica e finalmente, a ausência de outros casos de teste. Ao solicitarmos a um dos estudantes que descrevesse como foi sua experiência de aprendizagem realizando o trabalho, o mesmo relatou que apesar de ter entendido a teoria, na prática, não sabia como fazer.

> *"Foi estranho pra mim a criação dos testes, porque a gente tinha visto na teoria, tudo muito legal, tal, entendi muito bem, o que deveria ser feito, mas eu não sabia como deveria ser feito, tirei até aquelas dúvidas"* (ST8).

De maneira semelhante, outro estudante reportou a sua dificuldade em saber quais os casos de teste deveriam ser codificados.

> *"E... também exatamente, como testar (...) vou testar esse método, mas quais são... os... casos que eu tenho que fazer? Aí tudo bem, tem aquele da... da... teste caixa-branca, caixa-preta" (...) "Mas acho... acho que não é sempre assim, se for fazer todos os caminhos, não dá (...) eu imagino que não saia testando tudo, a torto e a direito"* (ST6).

As dúvidas na conexão entre a teoria e prática para a elaboração dos casos de testes para alguns estudantes, também justificam os níveis de concordância mais baixos para os objetivos de aprendizagem relacionados às capacidades de descrever e distinguir entre as diferentes técnicas caixa-preta (LO98a) e caixa-branca (LO98b), de criar testes de unidade que não sejam redundantes (LO99) e de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação) (LO95), mencionados anteriormente.

Percebemos, no entanto, que em algumas situações, os estudantes até perseguiram implementar a maioria dos casos de teste possível, conforme comentamos ao analisar a cobertura conseguida por cada equipe na Tabela 7.6. Todavia, como ocorre nas empresas, sempre existem limitações de tempo e gastos que impedem que mais casos sejam implementados. Na situação acadêmica, os estudantes tinham que distribuir seu esforço entre outras tarefas da própria disciplina, tarefas de outras disciplinas e outros compromissos como estágios e trabalhos de conclusão de curso. Consequentemente, é compreensível que nem todos os casos de testes tenham sido implementados.

Por fim, concluímos que os estudantes cumpriram o que foi proposto: implementaram testes automatizados para testar um software de médio porte envolvendo complexidade similar a encontrada nos softwares existentes nas empresas. Eles enfrentaram o desafio apresentado, apesar de todos as dificuldades discutidas em 7.2.1, inclusive fazendo ajustes e refatorações no próprio software para que o mesmo fosse testado (coluna 'LOC modif.' na Tabela 7.6). Os envolvidos também reconheceram esse aprendizado. Enquanto um dos estudantes experientes atestou a aprendizagem de seus colegas, o professor confessou que os estudantes que participaram dessa abordagem terminaram a disciplina com conhecimento muito maior que outros, que fizeram a disciplina em uma oferta anterior.

> *"O pessoal realmente aprendeu com o projeto"* (ST5).

> *"(...) eu dei duas turmas de DS III, duas... essa foi a segunda, tô indo pra terceira agora.... essa turma que saiu, saiu com muito mais conhecimento do que a primeira." (...) "eu acho que... é um avanço muito grande... porque realmente os alunos saem com uma quantidade de conhecimento muito... muito maior"* (P2).

Outro ponto positivo bastante relevante para a abordagem foi o interesse da comunidade do projeto JabRef pelos testes implementados. Não sabemos dizer se chegaram a olhar todos os testes, uma vez que a descrição dos *pull requests* e os comentários nos códigos estavam em português. Porém um dos desenvolvedores da comunidade mandou a seguinte mensagem para dois dos estudantes:

> *"This pull request reads interesting. Would you be interested in bringing this test to the JabRef main repository? (With English code comments :))"* (koppor).

Um dos estudantes respondeu a mensagem. Todavia como o mesmo não realizou os procedimentos solicitados, o próprio desenvolvedor da comunidade o fez, incorporando os testes à *branch* principal do JabRef.

"I did it for myself in JabRef#1976" (koppor).

Como o outro estudante não respondeu a mensagem do desenvolvedor, nada foi feito.

Após essa discussão, para o julgamento final se os objetivos de aprendizagem relacionados aos aspectos técnicos foram ou não alcançados, aplicamos as heurísticas e o algoritmo definidos no Capítulo 4, especificamente na Seção 4.5, sobre todos esses resultados. A Tabela 7.8 apresenta o julgamento para cada um dos objetivos. As dimensões empregadas para avaliação da aprendizagem aparecem em cada coluna. Portanto, a coluna 'melhora significativa' retrata o resultado positivo ou não do teste estatístico aplicado (ver Tabela 7.5). A coluna 'análise dos trabalhos' expõe alguns comentários que sumarizam pontos discutidos ao longo do texto sobre os trabalhos apresentados. A coluna 'concordância com a contribuição' denota o percentual do nível de concordância dos estudantes se a abordagem contribuiu para o alcance do objetivo de aprendizagem (representa a soma dos percentuais de 'concordo' e 'concordo plenamente' na Figura 7.5). A coluna 'relato dos estudantes' apenas indica se ocorreram um ou mais relatos dos estudantes sobre o alcance do objetivo em questão. Neste caso, acrescentamos ainda comentários quando algum relato negativo ocorreu. Cabe destacar que utilizamos um hífen ('-') para expressar quando não analisamos qualquer uma dessas dimensões para o objetivo sendo tratado. Finalmente, a coluna 'alcance do objetivo' aponta o julgamento final para cada objetivo de aprendizagem, após a aplicação das heurísticas citadas.

Tabela 7.8: Julgamento do alcance dos objetivos de aprendizagem associados aos aspectos técnicos no estudo EC2.

Id.	**Objetivo de aprendizagem**	**Melhora significativa**	**Análise dos trabalhos**	**Concordância com a contribuição**	**Relato dos estudantes**	**Alcance do objetivo**
LO90	Capacidade de descrever os conceitos e práticas relacionadas ao teste de software.	Não	-	85,7%	Sim	Sim
LO95	Capacidade de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).	Sim	Poucos problemas (3)	78,6%	Sim, mas houve também um problema de conexão	Sim
LO98a	Capacidade de descrever e distinguir entre as diferentes técnicas caixa-preta para elaboração de casos de teste.	Não	Maior número de problemas	64,3%	-	Parcial
LO98b	Capacidade de descrever e distinguir entre as diferentes técnicas caixa-branca para elaboração de casos de teste.	Não	Maior número de problemas	64,3%	-	Parcial

LO99	Capacidade de criar testes de unidade que não sejam redundantes (aplicar as técnicas de elaboração de casos de teste de modo que os casos gerados não sejam redundantes).	Não	Apenas 2 casos	85,7%	-	Sim
LO104	Capacidade de planejar e gerar casos de testes para softwares de médio porte.	Sim	Os estudantes foram capazes de implementar os testes satisfatoriamente	85,7%	Sim	Sim
LO101	Capacidade de criar e documentar um conjunto de testes para um segmento de código de médio porte.	Sim	Algumas falhas na documentação das técnicas e níveis de testes	85,7%	-	Sim
LO107	Capacidade de usar ferramentas de testes.	Sim	Os estudantes foram capazes de usar as ferramentas de testes satisfatoriamente	100,0 %	Sim	Sim
LO91	Entender a diferença entre testar pequenos programas e testar software de grande porte.	Sim	-	92,9%	-	Sim
LO93	Capacidade de avaliar uma suíte de testes para um segmento de código de médio porte.	Sim	-	71,4%	-	Sim
LO88	Capacidade de avaliar os testes sendo executados por meio de métricas.	Sim	-	78,6%	Sim	Sim
LO109	Capacidade de usar ferramentas de teste de cobertura eficientemente.	Sim	Os estudantes foram capazes de usar a ferramenta de teste cobertura eficientemente	78,6%	Sim	Sim
LO287	Capacidade de descrever o processo de testes de regressão e seu papel no gerenciamento de releases.	Não	-	78,6%	-	Parcial

LO103	Capacidade de descrever como selecionar bons testes de regressão e automatiza-los.	Não	-	64,3%	-	Parcial
LO110	Ter algum conhecimento sobre ferramentas de integração continua e testes de regressão.	Sim	O estudantes usaram uma ferramenta de integração contínua para configuraram um conjunto de testes de regressão.	78,6%	Sim	Sim
LO105	Capacidade de descrever como é um processo de gerenciamento de defeitos.	Não	Boas respostas nos trabalhos	64,3%	-	Sim
LO198	Capacidade de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (usar a ferramenta para reportar erros, monitorar os erros e submeter correções).	Não	-	71,4%	-	Parcial
LO123	Refatorar uma implementação de software existente a fim de melhorar algum aspecto do seu *design*	-	Algumas equipes realizaram refatorações no código	-	Sim	Sim
LO82	Construir, executar e depurar (*debug*) programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração (*debug*) visual	-	-	-	Sim	Sim
LO108	Demonstrar a capacidade de usar ferramentas de depuração (debug), em suporte ao desenvolvimento de um software de médio porte.	-	-	-	Sim	Sim

LO133	Demonstrar a capacidade de usar as ferramentas de gerenciamento de configuração, controle de versões e gerenciamento de releases em suporte ao desenvolvimento de um software de médio porte.	-	-	-	Sim	Sim
LO134	Ser capaz de fazer merge e ramificação de código	-	-	-	Sim, apenas um problema na equipe ET2	Sim

Desenvolvimento de Competências da Prática Profissional
Competências diretamente investigadas

Além das habilidades técnicas, investigamos a contribuição da abordagem para o desenvolvimento de algumas habilidades gerais. A Tabela 7.9 enumera os objetivos de aprendizagem associados ao desenvolvimento de habilidades gerais identificados no Capítulo 3 que perscrutamos diretamente através do questionário aplicado. Convém lembrar, que essas habilidades gerais são consideradas no SWEBOK como competências relacionadas à Prática Profissional em Engenharia de Software, classificação também utilizada no capítulo citado.

Tabela 7.9 Objetivos de aprendizagem explicitamente investigados no estudo EC2.

Id.	**Descrição**
LO233	Aceitar e responder críticas de terceiros
LO235	Avaliar o trabalho desenvolvido por terceiros
LO268	Ter experiência com código desenvolvido por terceiros.
LO236	Habilidade de ser proativo
LO237	Habilidade de ser criativo.
LO266	Ser capaz de solucionar problemas.
LO267	Gerar soluções alternativas para os problemas.
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).
LO251b	Desenvolver a habilidade de sumarizar material técnico (código fonte e documentação).
LO246	Escrever documentação, clara, concisa e acurada

A Figura 7.6 expressa a percepção dos estudantes sobre a contribuição da abordagem para o desenvolvimento das habilidades gerais examinadas. Tendo um visão global, houve concordância para todas as habilidades listadas, com destaque para LO268, LO267, LO236, LO235 e LO251a, como discutiremos a seguir.

Como esperado, uma vez que os estudantes tiveram que lidar com um projeto que não foi desenvolvido por eles, todos concordaram que o projeto contribuiu para que os mesmos tivessem uma experiência com código desenvolvido por terceiros (LO268). Todavia, também foi importante para que eles desenvolvessem a habilidade de avaliar o trabalho

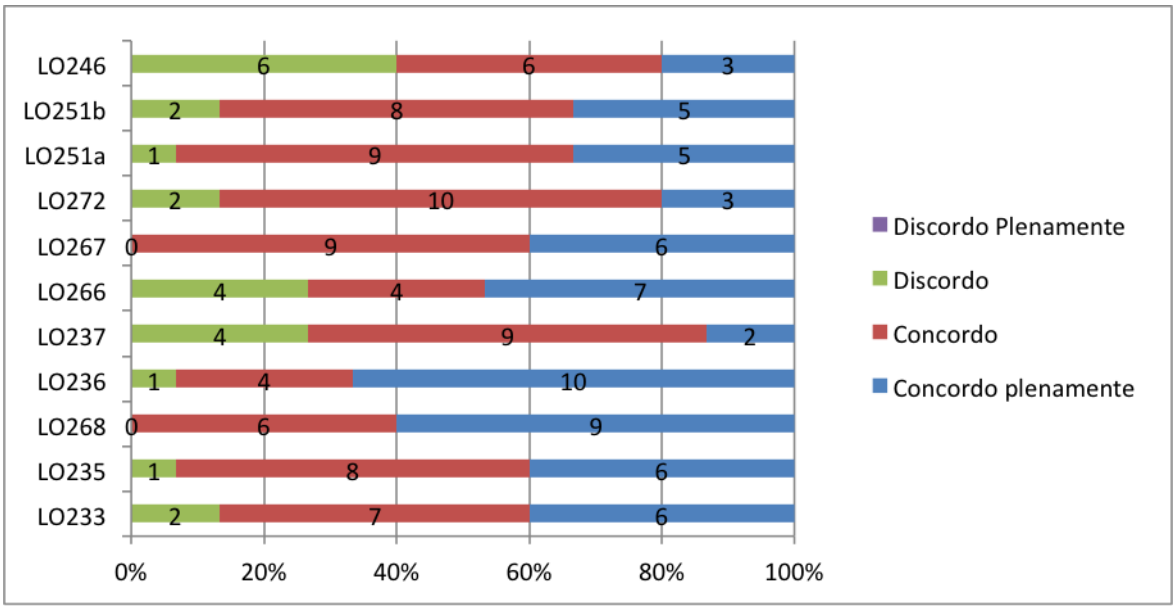

Figura 7.6 Contribuição do projeto para o alcance de objetivos de aprendizagem relacionados à prática profissional de engenharia de software no estudo EC2.

desenvolvido por terceiros (LO235), para 93,3% dos estudantes. Resultado também interessante foi que 86,7% consideraram que o projeto contribuiu para o desenvolvimento da habilidade de aceitar de maneira construtiva as críticas realizadas ao próprio trabalho (LO233).

Sob a perspectiva de solução de problemas, 73,3% consideraram que o projeto contribuiu para desenvolver a habilidade de solucionar problemas (LO266). Por outro lado, 100% concordaram com a contribuição do projeto para o desenvolvimento da habilidade de gerar soluções alternativas para os problemas encontrados (LO267). Geralmente, as habilidades de ser proativo (LO236) e ser criativo (LO237) também são associadas à solução de problemas, nesse caso a percentagem de concordância foi de 93,3% e 73,3%, respectivamente. A fim de tentar entender o maior número de discordâncias para o LO266 e LO237, comparamos as respostas individuais paras as quatro habilidades, dos estudantes que discordaram de algum delas, contudo não encontramos nenhum padrão de comportamento. O máximo que capturamos foram respostas mais uniformes para três estudantes e três estudantes com respostas mais diversas, variando inclusive da discordância à concordância plena. A única interpretação que chegamos foi que os estudantes podem ter associado a questão de ser criativo a gerar algo totalmente novo e a solução de problemas a prover uma solução definitiva para algum problema do software.. Em contrapartida, refletindo sobre os testes implementados pelos estudantes e as dificuldades que precisaram enfrentar para implementa-los no JabRef, a exemplo da mistura de regras de negócio da aplicação com a implementação da interface, dos vários métodos serem privados e da necessidade de refatoração do código, consideramos que os estudantes precisaram ser proativos e criativos para gerar soluções alternativas e assim solucionar os problemas encontrados para testar o software, consequentemente desenvolveram de alguma forma estas habilidades.

Quanto à contribuição para habilidades voltadas para comunicação, particularmente voltadas para a documentação escrita, obtivemos: 93,3% dos estudantes concordaram com a contribuição para o desenvolvimento da habilidade de compreender material técnico (LO251a); 86,7% concordaram com a contribuição para o desenvolvimento da habilidade de sumarizar material técnico (LO251b) e 60% concordaram com a contribuição para o desenvolvimento da habilidade de escrever documentação clara, concisa e acurada (LO246). Cabe ressaltar que essa última habilidade foi a que obteve o maior número de discordâncias, possivelmente porque os estudantes tiveram apenas que documentar os testes implementados, ou seja, nenhuma documentação do software propriamente dito foi solicitada.

Por último, 86,7% dos estudantes concordaram que o projeto contribuiu para eles tivessem uma visão do processo como um todo do desenvolvimento de software (LO272).

Competências detectadas a partir da análise qualitativa

Analisando qualitativamente as respostas das entrevistas, percebemos que outros objetivos de aprendizagem foram também cobertos por meio da abordagem. A Tabela 7.10 lista esses objetivos. A seguir discorreremos sobre o contexto do desenvolvimento dessas competências.

Tabela 7.10 Outras competências também desenvolvidas pela abordagem no estudo EC2.

Id.	Descrição
LO227	Habilidade de trabalhar efetivamente como membro de equipes.
LO242	Habilidade de lidar com incertezas e ambiguidades
LO250	Usar ferramentas de comunicação
LO270	Compreender como teoria e prática influenciam uma a outra
LO232	Expressar opiniões próprias.
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.
LO203	Compreender as expectativas de trabalho (habilidades esperadas)
LO269	Ter vivenciado pelo menos um projeto real.

Habilidade de trabalhar em equipe

Dois estudantes, quando questionados sobre a contribuição do projeto para a sua aprendizagem, assumiram o fato de trabalhar em equipe como importante para a sua aprendizagem. Os dois destacaram que suas equipes trabalharam e enfrentaram as dificuldades em conjunto.

> *"E o principal pra mim é a questão da equipe, a experiência de trabalhar em equipe (...). Tem as dificuldades... discutir as dificuldades... pra mim isso é uma aprendizagem, essa questão (...) trabalho em equipe é... pra mim é essencial... também, que teve muito no projeto"* (ST1)

> *"(...) trabalhar em equipe, assim, ter uma dificuldade e aí toda a equipe... foca naquilo, trabalhar, pesquisar... Acho que a dinâmica em grupo, acho que é o mais válido, assim..."* (ST3).

Esses estudantes pertenciam às equipes ET6 e ET2, respectivamente. Os relatos dos estudantes da equipe ET4 descrevem que o trabalho foi individualizado, porém a equipe trocava dicas e enfrentou, em conjunto, o desafio que surgiu.

> *"(...) cada um foi mais independente, porque a gente pegou funcionalidades separadas, e... a gente trocava dicas um com o outro, 'ah eu fiz assim', 'dá uma olhada lá no meu', e tal..."* (ST5).

> *"(...) para poder fazer funcionar pelo AssertJ foi um desafio que a gente teve que trabalhar todo mundo junto..."* (ST4).

A equipe ET7 ficou reduzida a apenas dois membros. Todavia, apesar de um deles [6] considerar que não houve um trabalho em grupo, em virtude de não ter ocorrido uma discussão maior, exatamente, pelas poucas pessoas envolvidas, verificamos que também existiu uma troca de informações e a ajuda mútua entre esses poucos membros, configurando o trabalho em equipe.

> *"(...) basicamente, em grupo o que a gente fez, foi só auxiliar um ao outro assim – 'não conseguiu montar o ambiente', 'não', 'siga isso e isso, faça isso, instale isso' e depois de... 'ah', também quando tava perdido 'como é que a gente vai testar', 'o que a gente vai testar' e... é não foi... não teve um trabalho de grupo mesmo, de sentar, discutir, porque até no começo tinha, o grupo todo ia participar, mas depois todo mundo falou 'não vou' e só ficou eu e ST72, mesmo"* (ST6).

O trabalho em equipe não funcionou realmente para as equipes ET1 e ET3, já que nesses casos apenas um estudante terminou participando em cada uma dessas equipes. Chegamos a sugerir que após a desistência dos demais membros, esses estudantes procurassem se engajar em outras equipes, contudo eles preferiram continuar trabalhando com os módulos do JabRef que já tinham começado a explorar. Entretanto essa situação acarretou em dificuldades para os dois estudantes, tais como transpor os desafios e não ter com quem discutir os tópicos antes da discussão com toda a turma. No entanto, o maior problema enfrentado foi a não execução dos testes funcionais baseados na interface por parte destes estudantes.

> *"Então, como nem todos os integrantes do grupo optaram por fazer o trabalho... e de início, todos estavam de acordo em fazer... falei com um depois... houve algumas alterações devido à disciplina, ficou meio que opcional. (...) Então, meio que não foi um trabalho em grupo, acabou, no final das contas, sendo um trabalho individual." (...) "Prejudicou muito, inclusive na segunda parte do desenvolvimento (...) você fazer o teste da interface... eu não consegui fazer, teve muita complicação, mas eu vi que outras pessoas conseguiram, mas pelo fato de eu ter feito algo individualmente... não, não tive muito sucesso... na implementação da tarefa"* (ST7).

[6] O único que foi entrevistado.

> *"(...) só tinha eu do meu grupo (...) E nisso acaba criando uma... como é que eu posso dizer... a gente acaba não tendo uma visão geral" [com relação à discussão dos tópicos dentro da equipe] (...) "eu poderia talvez ter resolvido minhas questões bem mais rápido, porque uma só pessoa... ela tem limitações. Se você tiver outras pessoas, elas podem me ajudar com outras pesquisas e com suas dúvidas, né... obviamente"* (ST8).

Concluímos assim, que em quatro equipes, o trabalho em equipe foi configurado, resultando em um desenvolvimento consciente (relatado pelos estudantes) e inconsciente da habilidade de trabalhar efetivamente como membro de equipes (LO227). Para as equipes onde as atividades terminaram sendo individualizadas, a relevância do trabalho em equipe foi ao menos reconhecida pelos estudantes, que foram, de algum modo, prejudicados.

Habilidade de lidar com incertezas e ambiguidades

Ao trabalharmos com projetos reais, situações adversas podem ocorrer a todo momento. Essas situações adversas podem causar incertezas e ambiguidades, que dificilmente podem ser simuladas em ambientes acadêmicos. No caso específico desse estudo de caso, um estudante reportou justamente uma contingência vivenciada em sua equipe com a qual eles tiveram que aprender a lidar, reconhecendo inclusive que tal situação ocorre nas empresas.

> *"(...) uma coisa que nunca aconteceu comigo (...) meu grupo eram 4 pessoas e depois saiu uma... e eu nunca tive isso em um trabalho, de ter um grupo, você dividir as tarefas e depois uma pessoa sair e você ter que replanejar as tarefas de novo, 'não, agora vai ser assim' (...) a gente teve que aprender a... lidar com isso aí.... Poderia ter sido até pior, né... sair no meio do projeto (...) isso é uma coisa ruim que acontece na vida real"* (ST1).

Não podemos dizer que a abordagem sempre fará com que os estudantes vivenciem situações adversas, contudo podemos dizer que essa possibilidade é grande, já que um projeto real está sendo utilizado. Sendo assim, sempre existe a possibilidade de desenvolvimento da habilidade de lidar com incertezas e ambiguidades (LO242), quando um projeto de código aberto for utilizado, a exemplo desse estudo de caso.

Usar ferramentas de comunicação

Apesar de não termos exigido que os estudantes interagissem com a comunidade do projeto JabRef, dois estudantes relataram que suas equipes criaram grupos no Facebook para discutir o projeto e distribuir tarefas, internamente. Este fato representa um exemplo simples de que a abordagem permite o desenvolvimento da habilidade de usar ferramentas de comunicação (LO250).

Compreender como teoria e prática influenciam uma a outra

Quando verificamos se a abordagem de utilização de projetos de código aberto permite que o estudante faça a conexão do que é visto na teoria com a prática em 7.2.2, discutimos um conjunto de características detectadas a partir da percepção dos estudantes, de como a teoria é importante para a prática, bem como, de como a prática é importante para a teoria. Portanto, retornando ao que foi discutido, conferimos que os estudantes puderam compreender como teoria e prática influenciam uma a outra, isto é, foi possível alcançar o objetivo LO270.

'Expressar opiniões próprias' e 'Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias'
Ainda referenciando 7.2.2, vimos que a discussão realizada após a execução da prática possibilitou que diferentes perspectivas pudessem ser expostas e que, assim, os estudantes conhecessem a opinião uns dos outros. Esse fato além de 'ampliar a visão', permitiu o 'desenvolvimento do pensamento crítico'. Ademais os extratos já mencionados na subseção citada, agregamos os relatos de três estudantes, para evidenciar esse aprendizado.

> *"(...) eu pensava uma coisa, vi a opinião das outras pessoas, concordei, discordei..."* (ST1).

> *"(...) foi interessante porque eu pude ver assim (...) quais testes poderiam ser feitos, que eu pensava em alguns e a galera pensava outros (...) eu achava teste de acessibilidade, usabilidade, depois vi que realmente era um projeto livre então não precisa também, os caras não têm como fazer todos os tipos de teste"* (ST6).

> *"A gente discute uma... uma questão, algum assunto relacionado ao trabalho em grupo, mas não tem... uma decisão definitiva, uma coisa... que dá uma garantia, né. E a gente compara o que, a senhora explicou lá... explicou algumas coisas e dá pra fazer uma relação e chegar realmente em um ponto, né, 'é isso, é aquilo' "* (ST3).

O último extrato relata a nossa participação na discussão, mas não com uma decisão final. A conclusão é de cada estudante, a partir da construção de sua opinião própria. As palavras dos estudantes, nos dois primeiros extratos, também deixam claro o desenvolvimento do pensamento crítico e a sintetização de novas ideias (LO241).

Convém ressaltar, que a discussão somente ocorreu porque os estudantes foram incentivados e sentiram-se à vontade para expor suas opiniões.

> *"(...) cada um falava sua opinião"* (ST2).

> *"(...) achei bom porque... como eu estava aprendendo mesmo, podia falar"* (ST6).

Consequentemente, implicitamente, houve o desenvolvimento da competência de expressar opiniões próprias (LO232).

'Vivenciar um projeto real' e 'Compreender as expectativas de trabalho (habilidades esperadas)'

Ao investigarmos a percepção dos estudantes sobre o contato com o projeto de código aberto, como uma experiência real de desenvolvimento de software em 7.2.1, verificamos que eles identificaram uma série de características que aproximam o projeto de código aberto à realidade dos softwares existentes nas empresas. Tais características aliadas às atividades que precisavam ser executadas, também similares às realizadas nas empresas, geraram uma lista de dificuldades para a execução das mesmas. Tudo isso, permitiu que os estudantes visualizassem um conjunto de habilidades necessárias aos profissionais de engenharia de software, tanto de modo geral, quanto efetivamente ligadas aos testes de software. Desse modo podemos assumir que, os estudantes, a partir da vivência de um projeto real (LO269), compreenderam as expectativas de trabalho para a profissão (LO203).

Julgamento sobre o alcance dos objetivos de aprendizagem associados às Competências da Prática Profissional

Para julgar se os objetivos de aprendizagem relacionados às competências da prática profissional foram ou não alcançados, aplicamos as heurísticas e o algoritmo definidos no Capítulo 4 (Seção 4.5), do mesmo modo que fizemos para julgar os objetivos associados aos aspectos técnicos. A única diferença é que neste caso, não aplicamos a dimensão relativa aos testes estatísticos, já que estes não foram realizados. Sendo assim, a Tabela 7.11 apresenta o julgamento para cada um dos objetivos. Com exceção da coluna 'melhora significativa', todas as demais colunas que representam as demais dimensões aparecem. Por conseguinte, a coluna 'análise dos trabalhos' expõe alguns comentários que resumem a nossa percepção a respeito do desenvolvimento de determinadas habilidades por parte dos estudantes com a realização do trabalho, discutidas ao longo do texto. A coluna 'concordância com a contribuição' denota o percentual do nível de concordância dos estudantes se a abordagem contribuiu para o alcance do objetivo de aprendizagem associado à competência da prática profissional (representa a soma dos percentuais de 'concordo' e 'concordo plenamente' na Figura 7.6). A coluna 'relato dos estudantes' apenas indica se ocorreram um ou mais relatos dos estudantes sobre o alcance do objetivo em questão. Cabe destacar que utilizamos um hífen ('-') para expressar quando não analisamos qualquer uma dessas dimensões para o objetivo sendo tratado. Finalmente, a coluna alcance do objetivo' aponta o julgamento final para cada objetivo de aprendizagem, após a aplicação das heurísticas citadas.

Tabela 7.11: Julgamento do alcance dos objetivos de aprendizagem associados às competências da prática profissional para o estudo de caso sobre testes de software (EC2).

Id.	Objetivo de aprendizagem	Análise dos trabalhos	Concordância com a contribuição	Relato dos estudantes	Alcance do objetivo
LO233	Aceitar e responder críticas de terceiros	-	86,7%	-	Sim
LO235	Avaliar o trabalho desenvolvido por terceiros	-	93,3%	-	Sim

LO268	Ter experiência com código desenvolvido por terceiros.	-	100,0%	-	Sim
LO236	Habilidade de ser proativo	Precisaram ser proativos	93,3%	-	Sim
LO237	Habilidade de ser criativo.	Precisaram ser criativos	73,3%	-	Sim
LO266	Ser capaz de solucionar problemas.	Precisaram solucionar problemas	73,3%	-	Sim
LO267	Gerar soluções alternativas para os problemas.	Precisaram gerar soluções alternatvas	100,0%	-	Sim
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.	-	86,7%	-	Sim
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).	-	93,3%	-	Sim
LO251b	Desenvolver a habilidade de sumarizar material técnico (código fonte e documentação).	-	86,7%	-	Sim
LO246	Escrever documentação, clara, concisa e acurada	-	60,0%	-	Parcial
LO227	Habilidade de trabalhar efetivamente como membro de equipes.	-	-	Sim	Sim
LO242	Habilidade de lidar com incertezas e ambiguidades	-	-	Sim	Sim
LO250	Usar ferramentas de comunicação	-	-	Sim	Sim
LO270	Compreender como teoria e prática influenciam uma a outra	-	-	Sim	Sim
LO232	Expressar opiniões próprias.	-	-	Sim	Sim
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.	-	-	Sim	Sim
LO203	Compreender as expectativas de trabalho (habilidades esperadas)	-	-	Sim	Sim
LO269	Ter vivenciado pelo menos um projeto real.	-	-	Sim	Sim

7.2.4 Sentimento de preparação para o mercado de trabalho (Ob8)

Buscamos investigar se o uso do projeto de código aberto permitiu ao estudante sentir-se preparado para o mercado de trabalho, a partir da percepção do estudante sobre

três aspectos: (QSP1) confiança em realizar atividades semelhantes em outro projeto; (QSP2) contribuição para a melhoria do seu futuro desempenho na vida profissional e (QSP3) contribuição para que o mesmo se sentisse mais preparado para a realização das atividades de testes no mercado de trabalho. A Figura 7.7 apresenta os níveis de concordância obtidos.

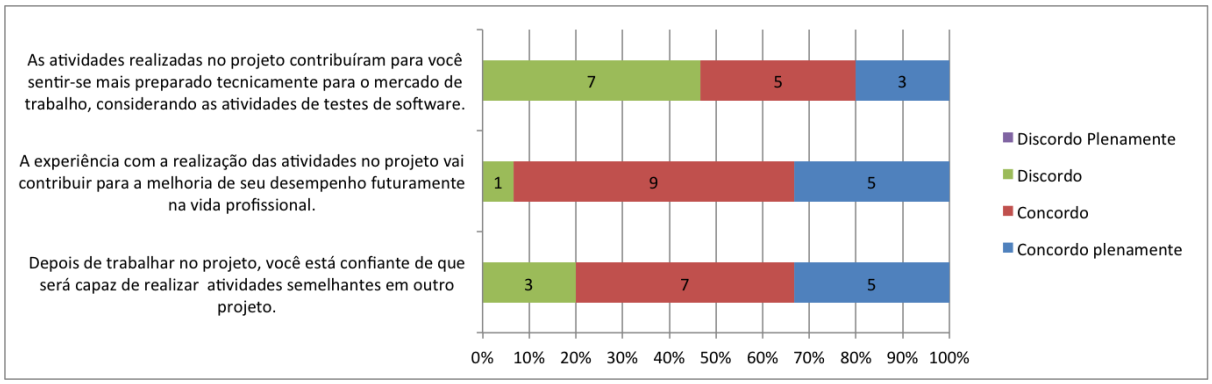

Figura 7.7 Níveis de concordância com relação à preparação para o mercado no estudo EC2.

Observamos que enquanto 80% dos estudantes estão mais confiantes que serão capazes de realizar atividades semelhantes em outro projeto (QSP1) e 93,33% concordam que as atividades no projeto contribuíram para a melhoria do seu desempenho profissional futuramente (QSP2), apenas 53,33% consideraram que estariam mais preparados tecnicamente para o mercado de trabalho, considerando as atividades de testes de software, após as atividades do projeto (QSP3).

Convém ressaltar que o único estudante que discordou que o projeto iria contribuir para o seu desempenho profissional futuramente (QSP2), durante a entrevista revelou que as ferramentas aprendidas não serviriam para suas atividades futuras, porque utilizava linguagens para as quais essas ferramentas não poderiam ser aplicadas. Contudo, mesmo sem reconhecer a aprendizagem proveniente da própria execução dos testes, admitiu ao menos que agora sabe que tipo de ferramenta procurar.

> *"Onde eu trabalho, não utilizamos Java, então... todas as ferramentas que eu aprendi a utilizar no... na prática, com os testes, não serão realmente utilizados, porque nas outras linguagens usa outras ferramentas.(...) Como eu tô trabalhando com programação Web direcionada a linguagens como Javascript e PHP... pra a área de trabalho, pra mim, não serviu muito. Só pra eu ter uma noção mesmo de que se eu fizer um teste PHP, daquela forma, eu vou precisar de outra ferramenta (...) Você sabe que existe uma ferramenta praquilo e se existe para uma linguagem muito provavelmente deve existir pra outra, já que as duas são grandes, e por isso você pode... já tem uma luz para onde ir"* (ST2).

Podemos destacar ainda que esse mesmo estudante discordou também das outras duas questões (QSP1 e QSP3), discordando portanto, de todas as questões. Supomos que a razão para tais discordâncias seja a mesma que citamos.

Por outro lado, encontramos circunstâncias para as quais, o projeto colaborou para que os estudantes sentissem-se preparados. Um dos estudantes enfatizou a situação de aprender a trabalhar em equipe proporcionada pelo projeto, para a preparação para o mercado de trabalho.

> *"(...) quanto mais você trabalha em equipe mais você fica preparado pra o mercado. Você sempre vai estar numa equipe, inserido em uma equipe... testando alguma coisa (...) pode ser pra testar ou não, mas se testar, ter uma experiência prática de trabalhar em grupo sobre testes, vai ajudar lá na frente... acho que também isso ajuda"* (ST1).

Nesse outro extrato, o mesmo estudante realçou a experiência de decidir os módulos a serem testados e realizar os testes.

> *"A gente pode ser contratado e cair lá pra trabalhar no teste de uma parte, com equipe. E tendo esse contato já na disciplina com a equipe, discutindo e tal, fazendo testes, vendo quais os módulos a ser testados também é um ponto positivo"* (ST1).

Já outro estudante, questionado sobre a sua experiência de aprendizagem realizando o projeto, destacou a aprendizagem sobre ferramentas, inclusive para uma futura necessidade no trabalho. Desse modo, ele admitiu sentir-se preparado para o uso dessas ferramentas ou mesmo para a pesquisa de outras ferramentas.

> *"Pra mim foi uma das melhores, porque acredito que as ferramentas que a gente aprendeu vão agregar também, né, a ter um conhecimento maior. (...) Porque ainda... a gente já traz... aprende ferramentas novas e já leva até pra um momento de trabalho ou a gente tem um portifólio" (...) "Eu me sinto preparado. Usando as ferramentas. Eu... não só usando essas ferramentas, mas também na questão... de poder pesquisar novas ferramentas"* (ST3).

Um dos estudantes concordou que após participar do projeto está mais confiante que será capaz de realizar atividades semelhantes em outro projeto (QSP1), que o projeto contribuiu para que ele se sentisse mais preparado para a realização das atividades de testes no mercado de trabalho (QSP3), inclusive, concordou plenamente que as atividades no projeto contribuíram para a melhoria do seu desempenho profissional futuramente (QSP2). Mesmo assim, reconheceu que ainda não está totalmente preparado.

> *"Eu sinto que ainda não tô preparado (risos), que teria que estudar mais (risos), muito mais"* (ST6).

Todavia, admitiu que foi importante vivenciar as dificuldades provenientes do emprego de um projeto grande ainda na universidade.

> *"(...) o aprendizado foi difícil, mas eu achei que finalmente pelo menos tive a oportunidade de testar isso, antes de algum emprego..."* (ST6).

Semelhantemente, outro estudante, referindo-se à aprendizagem da teoria associada à prática, ressaltou o valor da prática especialmente utilizando um projeto real, como relevante como primeira experiência para a preparação para o mercado de trabalho.

> *"Você aprenderia mas sem vê lá... principalmente com um software real, software existente, testar... é meio que aprender, mas na hora que for pro mercado ainda não estar preparado. Acho que ter esse contato já com o software... já... na hora que você cai lá na ficha, no software, estar testando... não é a primeira vez, você já tem um pouco de experiência, isso ajuda, pra mim, isso foi um plus, aí" (...) "também é um ponto positivo, de se fazer... de testar um projeto real antes do mercado"* (ST1)

Outros estudantes também confirmaram a importância da vivência de um projeto de software real, para proporcionar uma experiência mais próxima do que acontecerá no mercado. Como um deles mencionou, o estudante passa a ter uma perspectiva de como as coisas ocorrem fora da academia.

> *"Acho que talvez na academia, no caso, pro JabRef, o uso desse método é válido porque já traz um pouquinho do que seria o você aplicar... o conhecimento da disciplina mesmo na... num projeto acadêmico, visando já tipo, 'oh, isso daqui que você tá trabalhando, vai ser mais ou menos o que vai acontecer fora daqui, da academia' " (...) "você tem essa perspectiva de trabalhar: 'nós trabalharemos com projetos complexos... desse porte em diante' "* (ST7) .

Mesmo o professor reconheceu esse fato.

> *"Os alunos realmente no curso de graduação tiverem essa... essa experiência..."* (P2).

Para um dos estudantes experientes, qualquer que seja a vivência de um projeto real, ela sempre vai agregar para o conhecimento do indivíduo.

> *"(...) eu acho que cada pedaço de experiência, independente de... do quanto você tenha visto aquilo, quando você tá trabalhando com um software novo, com pessoas diferentes, acho que... independente de qual situação aquilo tá sempre adicionando pra o... o seu conhecimento acadêmico, profissional e pessoal também"* (ST5)..

Um exemplo desse fato é que um dos estudantes pôde visualizar outros aspectos do processo final de entrega do software, além de métricas e testes de regressão, dando-lhe uma visão mais profissional, que também é relevante.

> *"(...) a experiência de... de ver, é... por exemplo, ver outros aspectos mais... de entrega do software, mesmo... como... a estruturação do projeto, instalador, multilinguagem... coisas assim, foram interessantes também, que eu não*

> *tinha visto nem tido nenhum tipo de experiência a respeito (...) As ferramentas de métricas também muito interessantes... de análise de código também... da automatização dos testes de regressão... muito, assim, interessante pra saber fazer um produto final... a importância disso pra se fazer um produto final... muito, muito boa. Acho que isso contribuiu bastante assim, pra minha visão profissional. Apesar de que eu tenho uma visão bem mais acadêmica, porque inicialmente é meu objetivo, mas essa visão profissional foi bastante interessante. Que eu sabia que me faz falta mesmo"* (ST8).

Esse mesmo estudante, quando questionado se as atividades realizadas contribuíram para a sua formação profissional, afirmou que a aplicação teórica, em especial, em um projeto grande permitiu a visão do que será realmente encontrado no mercado.

> *"Sim, claramente tudo que eu falei até agora, desde os aspectos teóricos, a aplicação deles, a minha perspectiva a respeito de projetos grandes, que foi ampliada. Todas essas questões, elas ajudam a ter um vislumbre de como fosse... como realmente será no mercado"* (ST8).

Também comentando sobre as atividades que foram realizadas, outro estudante declarou que o projeto proporcionou uma visão profissional diferentemente das outras disciplinas do curso.

> *"O meio profissional é isso" (...) "Então, acho que dentre todas as disciplinas daqui, que eu peguei no curso, talvez uma que me introduziu algo realmente do mercado foi essa... na questão de testes, você trabalhar numa visão mais profissional"* (ST7).

Corroborando com a maioria dos estudantes que concordaram que após a participação no projeto estão mais confiantes para realizar atividades semelhantes em outro projeto (QSP1) (ver Figura 7.7), um estudante afirmou que o fato de ter efetivamente testado um software lhe deu uma maior segurança para o mercado de trabalho, dado o conhecimento suficiente adquirido e a possibilidade de enxergar como tudo acontece.

> *"Ter um conhecimento mínimo para poder pesquisar as ferramentas ou... ou quais as abordagens de teste, as técnicas, os métodos, dá uma segurança maior" (...) "ter uma segurança maior para o mercado de trabalho, você pelo menos realizou uma tarefa de implementação, de testar alguma coisa... e você realmente perceber o que é que tá acontecendo"* (ST3).

Exemplos de desenvolvimento da autoconfiança dos estudantes por meio da participação no projeto, também podem ser visualizados nos extratos a seguir. O primeiro estudante percebeu que é capaz de entender e consegue trabalhar com código razoavelmente grande, desenvolvido por outras pessoas, profissionais experientes, com vários anos produzindo software.

> *"Achei interessante pegar o software de porte médio mesmo e realmente ir a atrás e... realmente conseguir entender um código, ver que você está desse nível de pegar um código grande, feito por gente que trabalha na comunidade mesmo, gente que trabalha sei lá, há quantos anos fazendo software... e vê que você consegue entender aquilo... vê que você tá progredindo no nível de 'ah, isso aqui é o que ele quer fazer' (...)"* (ST4).

Simultaneamente, o segundo estudante afirmou que agora sabe por onde começar a trabalhar com um software grande desenvolvido por terceiros, isto é, o que deve fazer, quais ferramentas aplicar.

> *"(...) como hoje, em certo sentido tenho algumas técnicas pra adentrar, por onde começar, que tipo de ferramenta utilizar pra... tentar descobrir algumas questões. Existe é... algum já... alguns conhecimentos que me ajudam nisso"* (ST8).

Resumimos todas essas circunstâncias capturadas que favoreceram a preparação para o mercado de trabalho, como uma experiência profissional, detalhada em: (i) capacitação proporcionada; (ii) vivência de um projeto de software; (iii) visão profissional e (iv) desenvolvimento da autoconfiança.

Portanto, apesar de alguns estudantes discordarem que o projeto tenha contribuído para a sua preparação para o mercado (QSP3) (ver Figura 7.7), o maior nível de concordância para as demais questões aliado à experiência profissional identificada possibilitam que assumamos que a abordagem contribui para essa preparação. Principalmente, considerando que esse estudo corresponde a uma primeira aplicação da abordagem para a aprendizagem de testes de software e que muitos ajustes ainda podem ser realizados.

7.2.5 Consequências para a aprendizagem (Ob5)

Paralelamente à investigação de objetivos específicos de aprendizagem, procuramos também explorar, de modo mais geral, quais as consequências para a aprendizagem de engenharia de software por parte do estudante, quando projetos de código aberto são empregados. Intuitivamente, através de uma perspectiva qualitativa, surgiram os temas que, de certo modo, já foram discutidos nas seções anteriores, mas que serão complementados por temas não apresentados ou por pontos de vista ainda não discutidos. Sendo assim, dividimos o conteúdo a ser apresentado nesta subseção nos seguintes tópicos: (i) projeto de código aberto como situação autêntica para a aprendizagem; (ii) mudanças de atitude; (iii) dificuldades a serem enfrentadas; (iv) escolha do projeto e (v) visão geral sobre a aprendizagem proporcionada.

Projeto de Código aberto como situação autêntica para a aprendizagem

O projeto de código aberto exibe algumas características que favorecem a aprendizagem sob aspectos distintos dos que são encontrados usualmente em disciplinas de engenharia de software. Ele corresponde a um *"software pronto"* (ST1) (ST3), *"fora do padrão, que a gente tem (...) não é o professor que cria (...) ou mesmo, que a gente cria simplificado,*

pra o ambiente acadêmico" (ST3). Portanto, o software de código aberto propicia que os estudantes trabalhem com um projeto já existente, como muitas vezes ocorre com os profissionais na indústria de software e diferentemente do que os estudantes estão acostumados na academia. Representa um *"projeto real"* (ST1) (ST3), apresentando todas as características e, consequentemente, todas as dificuldades de se lidar com um projeto real, mencionadas em 7.2.1, que possivelmente não são retratadas nas simplificações utilizadas na academia. Do mesmo modo que na indústria, existe *"a questão de lidar com o código (...) que outras pessoas fizeram, diversas pessoas"* (ST3) e *"você tem que entender o... o que foi feito"* (ST7). Novamente, da mesma maneira que na indústria, isso permite que o estudante visualize estilos diferentes de programação: *"tem implementações diferentes que eu não tinha visto na sala de aula"* (ST8). O estudante também percebe que para trabalhar em equipe é necessário algum padrão de organização: *"pegar um projeto com tanta gente e fazer de qualquer jeito, não vai ficar usável"* (ST4) e que *"cada grupo possui seu padrão"* (ST2). Finalmente, que é necessário seguir esse padrão: *"eles têm o padrão de testes deles... então a gente teve que se adequar a esse padrão e seguir à risca o que eles estavam fazendo ali"* (ST2).

Tudo isso faz com que a abordagem represente uma situação autêntica de desenvolvimento de software vivenciada pelo estudante, que promove a aprendizagem de uma maneira diferente do usual, mas importante para a formação profissional do estudante (como vimos em 7.2.4). Embora o OSP represente uma situação autêntica de desenvolvimento de software, segundo um dos estudantes, não é tão difícil contribuir. Até com ações simples, o estudante pode participar.

> *"Então, eu vi que não é esse bicho de sete cabeças todo, né... num... num... felizmente você não tem que... sei lá... entender o código todo, até você poder ajudar em alguma coisa... pode ajudar com pequenas coisas, mesmo tradução... até eu vendo lá, o pessoal... o pull request que o pessoal faz, são coisas assim, bem simples"* (ST6).

Mesmo em se tratando de contribuir com testes, enfrentadas as dificuldades abordadas em 7.2.1, outro estudante comentou que todos os membros de sua equipe entenderam e foi simples seguir as recomendações da comunidade do projeto para a codificação dos mesmos.

> *"Todo mundo na equipe entendeu como é que tava sendo feito, como era que eles pediam pra serem separados ou ... Porque ele tem um arquivo lá que diz todo o passo-a-passo de como fazer o teste, definições de nomes, essas coisas... foi simples seguir"* (ST2).

Mudanças de Atitude

A partir da abordagem também capturamos a conscientização dos estudantes sobre diversos aspectos, entre eles: (i) importância da teoria; (ii) importância da engenharia de software; (iii) importância da organização do projeto; (iv) importância dos testes de software; (v) importância dos testes automatizados e (vi) para testar realmente, é preciso pensar em várias possibilidades.

Para um dos estudantes, a prática tornou a teoria sobre testes de software ainda mais relevante. A teoria seria realmente útil para a execução dos testes.

> *"Eu acho que tornou mais valiosa a teoria pra mim... é no sentido de que... ela se demonstrou útil pra checar algumas questões"* (ST8).

Mais especificamente, outro estudante ressaltou que a experiência com as dificuldades a serem enfrentadas ao trabalhar com um projeto real fez com que ele entendesse o porquê da teoria de engenharia de software.

> *"Acho que é bom ter um contato também com um software mais complexo, até pra você sentir a dificuldade de desenvolver e até pra você ver porque existe essa teoria toda de engenharia de software"* (ST1).

O desafio de lidar com software desenvolvido por terceiros proporcionou que um estudante destacasse que a organização do projeto é essencial para facilitar (tornar mais agradável) o trabalho de futuros desenvolvedores.

> *"(...) aprender que você tem que organizar suas coisas muito bem (risos), porque senão fica muito chato..."* (ST2).

Finalmente, no que diz respeito a testes, o mesmo estudante mencionado que entendeu a importância da teoria da engenharia de software, conscientizou-se sobre a importância dos testes de software.

> *"Eu aprendi a dar mais importância aos testes" (...) "Teste nunca é demais, né... quanto mais teste você colocar num software, melhor"* (ST1).

O estudante ainda acrescentou que a implementação dos testes automatizados na disciplina coincidentemente à leitura de um livro que ressaltava o valor desse tipo de teste fizeram com que ele compreendesse a contribuição dos testes automatizados, para a garantia da corretude do código. Consequentemente, gerou uma mudança de comportamento por parte do estudante.

> *"Eu sempre que programo, assim (...) eu não faço testes automatizados. Geralmente, você faz aquele teste... olhômetro, né, manual. (...) Aí... com a disciplina, eu... eu até lendo um livro, por coincidência tinha um tópico sobre testes, eu vi lá que... o Martin Fowler falando sobre testes (...) apesar de pra programadores iniciantes, o teste parecer ser cansativo, ter que programar mais código, mas ele acelera muito o tempo de desenvolvimento. Por isso, eu fiquei com isso na cabeça... Realmente, fazer um teste bom automatizado, você está fazendo um código, se os testes estão passando, muito dificilmente aquele código tem algum erro. Não quer dizer que não tenha erro, mas é melhor... é mais garantido do que você ir só no olhômetro e testar o código. Isso pra mim foi... foi um plus, a questão de teste automatizado. Agora eu tenho uma visão mais ampla sobre isso e usar muito teste automatizado"* (ST1).

Já outro estudante começou a questionar o pessoal da empresa na qual estava estagiando sobre o porquê de não implementar testes automatizados. Ademais, o estudante reconheceu a relevância dos testes automatizados para garantir que modificações futuras não causem erros no software.

> *"Influenciou na verdade pelo fato de eu começar a questionar o pessoal lá de porque não tá fazendo testes, porque sempre foi... lá no trabalho era 'faça isso, teste', como é o teste, testa o navegador, se tá funcionando, se tá resultando certo, 'ok'. Nada de um teste automatizado que pudesse ... quando mudasse alguma coisa(...) Tanto que a gente tinha muito problema depois que mexia em alguma coisa, aí, só descobria quando o cliente ligava reclamando"* (ST6).

Outro estudante ainda constatou que para testar realmente, não basta executar o código uma única vez, mas sim, que é preciso pensar em várias possibilidades.

> "... ele me deu essa noção de que as telas (...) fazer todos os tipos de testes... de clicar numa tela, voltar pra ela, fazer de novo... não é só fazer uma vez(...)o teste anterior que ia fazer era só clicar em uma, insere, vai na outra, abre e ver se tá lá... na primeira vez funciona... tive que fazer duas vezes... Pensar nesse tipo de coisa não é... não é o padrão, vamos dizer assim (...) Agora eu... 'será se for de novo vai tá lá, de novo?' 'Será que se eu fizer o mesmo processo duas vezes, ele vai se repetir?' Geralmente você faz uma vez só e pronto... o teste da pessoa que não tenha essa visão, que o teste de software faz.. 'fez uma vez, aí tá bom, próximo'" (ST4).

Mais um relato bastante significativo de mudança de comportamento foi-nos enviado por um dos estudantes, seis meses após o término do projeto. Nele, o estudante, agora formado, relata a importância da participação na disciplina para a conscientização sobre os testes automatizados, particularmente após o problema causado, por colocar em produção uma modificação na aplicação, sem a execução de testes. O estudante também assumiu a mudança de atitude em já projetar a aplicação de maneira a facilitar a criação de testes.

> *"Estou fazendo o projeto de forma a facilitar a criação de testes e eu acho legal compartilhar isso com você =) obrigado!"* (ST24).

> *"Outro ponto importante é que após a disciplina que fui me preocupar em fazer testes automatizados (também devido a acontecer um caso de subir uma versão com um problema numa das features do app)"* (ST24).

Através do questionário, procuramos identificar a percepção dos estudantes sobre a contribuição da abordagem para que eles compreendessem a importância dos seguintes aspectos: (i) especificação de requisitos (QA1); (ii) documentação do software (QA2); (iii) construção de testes automatizados dentro do processo de desenvolvimento e evolução do software (QA3); (iv) aplicação de técnicas apropriadas para testar o software (QA4) e (v), criação de casos de testes de maneira sistemática (QA5).

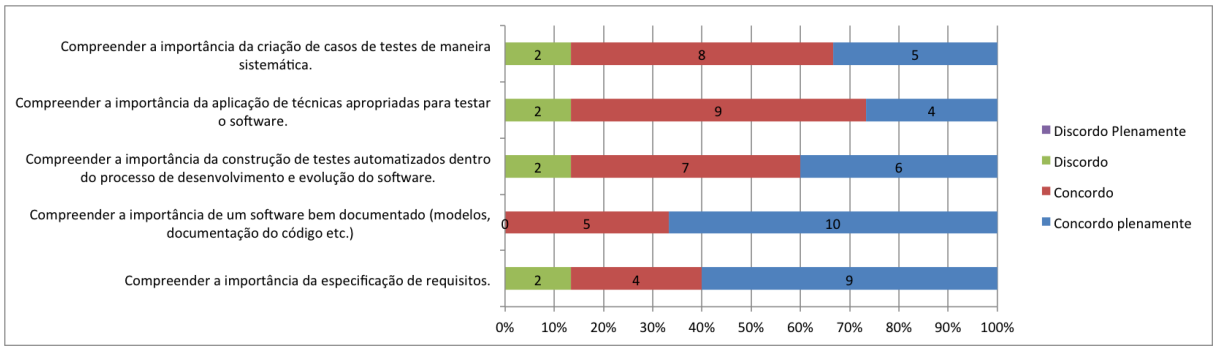

Figura 7.8 Níveis de concordância com relação à contribuição do projeto para conscientização dos estudantes no estudo EC2

A Figura 7.8 apresenta que a grande maioria dos estudantes concordaram que a abordagem contribuiu para que os mesmos compreendessem a importância de todos os aspectos questionados. Os destaques são para o reconhecimento da relevância da documentação do software (QA2), com 100% de concordância, sendo que 66,67% dos estudantes concordaram plenamente com essa contribuição, e para o valor da especificação de requisitos (QA1), com 60% de concordância plena.

Dificuldades a serem enfrentadas

Dentro do contexto do processo de ensino-aprendizagem sempre ocorrem dificuldades que influenciam direta ou indiretamente a aprendizagem dos estudantes. Por isso, vimos por bem descrever as dificuldades que não foram associadas a nenhum dos objetivos da tese, que discutimos anteriormente. Por conseguinte, discorremos sobre as dificuldades relacionadas ao uso das ferramentas e as dificuldades provenientes da própria aplicação da abordagem.

Dado que o objetivo principal das atividades solicitadas aos estudantes foi que eles construíssem testes automatizados, a abordagem exigia a utilização de ferramentas. Como, ao longo da disciplina, os estudantes receberam apenas uma pequena introdução ao JUnit, todas as demais ferramentas precisaram ser apresentadas. Consequentemente, os estudantes precisaram encarar o desafio de aprender novas ferramentas. Os estudantes apontaram uma curva de aprendizagem mais acentuada para o AssertJ e Mockito, sendo que um deles acrescentou a dificuldade de entender como o Gradle (ferramenta de automação de *Build* utilizada pelo JabRef) funcionava e a equipe ET6 adicionou que foi necessário estudar mais sobre JUnit, para entender como o *framework* tratava o lançamento de exceções. Já a equipe ET2, descrevendo as dificuldades que enfrentou, queixou-se que o uso do Mockito dificultou ao invés de ajudar.

Entretanto, apenas um estudante relatou um problema real com a utilização de ferramentas. O problema ocorreu na utilização da ferramenta de controle de versões.

> *"A gente estava fazendo teste com um repositório e quando percebeu o repositório era muito mais recente do que o repositório utilizado na disciplina. E quando a gente foi portar tudo pro... quando a gente foi fazer esses testes*

no... do AssertJ, nada funcionava..." (ST2).

Finalmente, dois estudantes resumiram que a grande dificuldade foi de fato a diversidade de ferramentas e o tempo curto para aprendê-las.

> *"(...) diversidade de ferramentas, que você tem que aprender em pouco tempo"* (ST3).

> *"(...) o que eu quero dizer é que em um tempo curto pra aprender uma ferramenta e colocar ela em prática, tendo a variedade assim de possibilidades... foi uma situação um pouco assim... como eu posso dizer... desafiadora"* (ST8).

No que diz respeito à aplicação da abordagem, a dificuldade mais citada pelos envolvidos, professor e estudantes, foi exatamente esta: a questão do pouco tempo disponível para a aprendizagem das ferramentas (como mencionado), para o entendimento do projeto de código aberto, para a implementação dos testes e por fim, para a discussão do conteúdo associado.

O professor justificou que o espaço de tempo reduzido para a realização de práticas deve-se ao conteúdo programático extenso estabelecido para a disciplina, aliado ao período corrido, onde muitos estudantes eram formandos e paralelamente cursavam muitas disciplinas. Como possíveis soluções, o mesmo sugere o redimensionamento do programa da disciplina ou do escopo das atividades práticas, até mesmo reconsiderando o número de ferramentas a serem aprendidas.

Em contrapartida, os estudantes justificaram o problema do tempo curto ter ocorrido, principalmente devido a carga de trabalho excessiva da própria disciplina, onde os estudantes além do projeto prático, foram avaliados através de provas, seminários e resumos. Segundo um estudante, o projeto poderia ter sido até estendido para refatoração e qualidade de software, conteúdos também presentes no programa da disciplina. Como soluções propostas, eles apontaram além da redução do número de avaliações da disciplina, que o projeto seja iniciado desde o princípio do período letivo, que o processo seja mais gradativo, enfatizando a abordagem do projeto mais simples, inicialmente, enfim, que as atividades sejam quebradas em tarefas com escopos menores e repassadas sequencialmente, estabelecendo-se prazos de entrega também reduzidos. Tais sugestões demonstram, de certo modo, a satisfação dos estudantes com a abordagem prática de utilização do projeto de código aberto e o reconhecimento da aprendizagem proporcionada.

Escolha do projeto

Outra questão que pode ser associada às consequências para a aprendizagem dos estudantes representa a escolha do projeto de código aberto a ser utilizado.

No caso específico do JabRef, fatores como domínio do projeto, tecnologia empregada, situação do código e situação da documentação influenciaram as dificuldades a serem enfrentadas pelos estudantes ao realizarem as atividades, consequentemente a motivação, satisfação e aprendizagem dos mesmos.

Particularmente sobre o seu domínio, os estudantes consideraram uma aplicação importante, já que várias pessoas utilizam-na; que ela está dentro do seu contexto

(acadêmico) e é bastante útil para quem faz pesquisa. Desse modo, os estudantes consideraram interessante trabalhar com a aplicação.

Sobre a tecnologia empregada, apesar de um estudante valorizar a linguagem de construção da aplicação por ser utilizada ao longo do curso, dois estudantes não se sentiram muito motivados. Um deles justificou que não era *"muito fã"* de utilizar Java com Eclipse, enquanto que o outro alegou que não gostava da linguagem e que ela não era utilizada em seu ambiente de trabalho.

Por último, ao longo da Subseção 7.2.1 discutimos uma série de dificuldades causadas pela situação do código e a falta de documentação do projeto. Mesmo que essas situações aproximem o projeto de código aberto das condições existentes na indústria de software, elas impõem desafios. Contudo, tais desafios não podem impor uma sobrecarga muito grande aos estudantes, acarretando desistências dos mesmos. Para um dos estudantes, o projeto grande, com o código *"bagunçado"*, e não conhecer as ferramentas foram as principais razões para que os demais estudantes não participassem do projeto, uma vez que essa participação era opcional.

> *"Eu acho que... a maioria da turma tava um pouco assustada com o fato de... de tá trabalhando com um projeto um pouco bagunçado... e grande, sem saber as ferramentas. Acho que isso assustou muita gente e acho que é por isso que nós não tivemos uma participação muito grande da turma... Porque realmente é complicado"* (ST5).

O professor da disciplina confirmou essa mesma percepção, de modo que, os estudantes preferiram dedicarem-se às atividades que eram obrigatórias, para assim conseguirem boas notas.

> *"(...) pelo que eu percebi... das observações dos alunos (...) uma certa resistência dos alunos que queriam fazer, mas quando sentiram o mínimo de dificuldade, recuaram, acharam que fazer a prova era mais fácil do que encarar aquelas ferramentas que eles não entenderam bem, vasculhar aquele código que eles acharam complicado... né"* (P2).

Um estudante acrescentou que um dos membros do seu grupo declinou de participar por não ter conseguido definir o que ele iria testar.

> *"(...) teve o caso de [ST11] que ele queria ter feito o projeto, só que como ele não conseguiu entender a parte... a busca do código, qual funcionalidade ele tinha que testar, ele acabou desistindo"* (ST7).

Entretanto, o professor ainda salientou que os bons alunos justificaram a não participação em razão da falta de tempo para fazer as atividades. Convém ressaltar que a questão do tempo disponível reduzido surge novamente como principal problema para a abordagem, conforme discutimos anteriormente.

> *"(...) um ou dois bons alunos que passaram com facilidade, não participaram do projeto, dizendo que não fariam porque viram que realmente não tinham tempo, pra fazer (...)"* (P2).

Os estudantes que participaram também admitiram a sobrecarga inicial. A surpresa ao perceberem que era algo realmente grande, levando-os a pensar em desistir. Porém, com persistência, eles conseguiram implementar os testes satisfatoriamente (rever 7.2.3).

> *"(...) pegar um software que eu nunca tinha visto em minha vida, que foi um pouco aterrador, inicialmente"* (ST4).

> *"(...) fiquei perdido muitas vezes pra achar onde é que tá tal funcionalidade... e como testar até... e fiquei... tinha horas que eu falava, (risos) 'vou desistir, porque eu não tô nem sabendo onde eu vou botar esse' (risos) 'preciso de uma coisa aqui' (...) então a gente achava 'é, parece simples' e quando foi lá, (faz um barulho com as mãos) leva um baque!! 'Isso aqui é grande... isso aqui é grande... código é grande...' "* (ST6).

Visão geral sobre a aprendizagem proporcionada
Finalmente, resumimos a aprendizagem que capturamos com a aplicação da abordagem na Figura 7.9. Considerando a aprendizagem em curto prazo, podemos listar o entendimento e a compreensão de conceitos de testes de software e projetos de código aberto, bem como, o desenvolvimento das habilidades de testar um software e usar as ferramentas e *frameworks* associados. Refletindo sobre uma aprendizagem mais em longo prazo, destacamos: (i) o desenvolvimento de habilidades gerais ligadas à melhoria de competências da prática profissional; (ii) mudanças de atitudes, com a conscientização da importância desde a teoria envolvida até a necessidade de pensar em várias possibilidades ao testar o software; (iii) a experiência profissional proporcionada e, por fim, (iv) a aplicação de fato do conteúdo estudado no exercício profissional, de acordo com o relato de um ex-aluno, onde este fez questão de compartilhar conosco a necessidade de construir testes automatizados no projeto no qual está trabalhando atualmente. Cabe lembrar que discorremos sobre todas as demais categorias nas Seções 7.2.3, 7.2.4 e 7.2.5.

7.2.6 Lacunas na Indústria de Software (Ob9)

O último objetivo da pesquisa foi investigar se o emprego da abordagem de uso de projetos de código aberto ajuda a suprir algumas das lacunas de formação identificadas pela indústria de software. Portanto, retornando ao Capítulo 3, analisamos quais deficiências de formação estão relacionadas às áreas de conhecimento do SWEBOK direta ou indiretamente tratadas pela abordagem, em virtude dos conteúdos e competências abordados pelas atividades do projeto.

Para julgarmos se a abordagem ajuda a suprir às deficiências, simplesmente verificamos se o objetivo de aprendizagem associado à deficiência foi ou não tratado durante a aplicação da abordagem e consequentemente, se foi alcançado, parcialmente alcançado ou não foi alcançado. Como referência para o alcance dos objetivos utilizamos as Tabelas 7.8 e 7.11.

A Tabela 7.12 apresenta as deficiências selecionadas e os objetivos de aprendizagem associados, organizados por área de conhecimento do SWEBOK. A última coluna indica se a abordagem contribuiu, contribuiu parcialmente ou não contribuiu para o suprimento

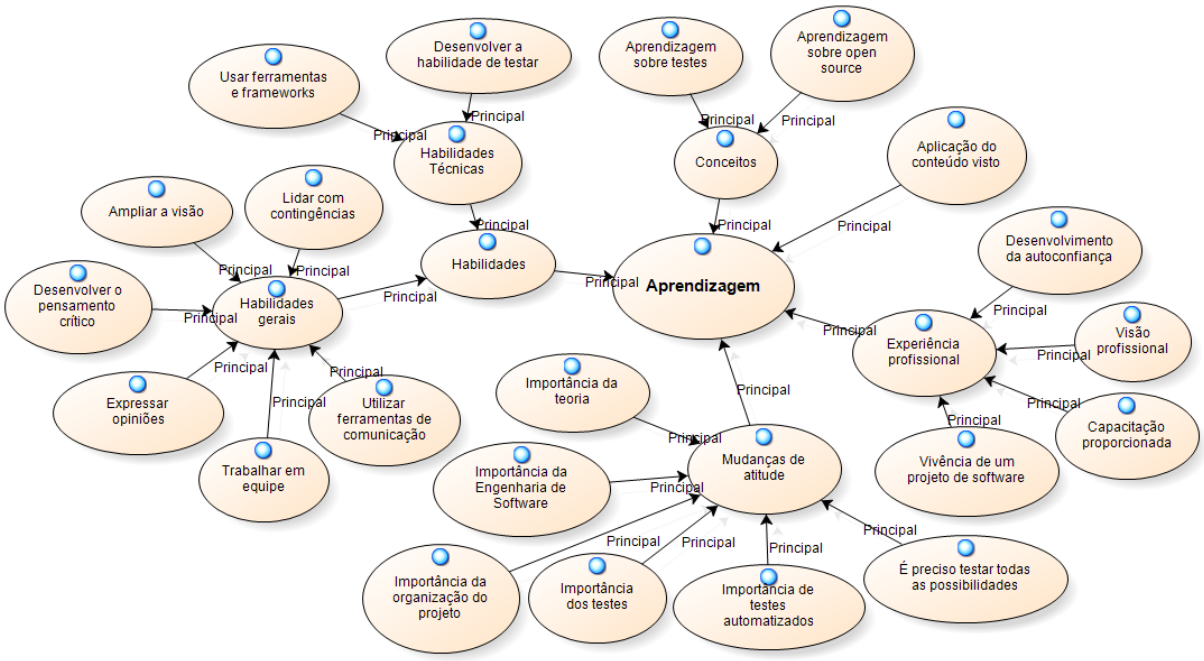

Figura 7.9 Contribuição da abordagem para a aprendizagem no estudo de caso sobre testes de software (EC2).

da deficiência. Assinalamos as deficiências que não foram diretamente tratadas pela abordagem com um hífen (‘-‘).

Tabela 7.12: Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Testes de Software (EC2).

Deficiência	**LO**	**Contribuição**
Teste de software		
Construção de testes de unidades não redundantes	LO99	Sim
Processo de testes e correção de erros sem que haja a reintrodução de erros antigos	LO100	-
Uso de ferramentas de testes	LO107	Sim
Uso de ferramentas de localização de erros (*debugging*)	LO108	Sim
Uso de ferramentas de cobertura de testes	LO109	Sim
Exposição a ferramentas de integração contínua e testes de regressão	LO110	Sim
Construção de Software		
Uso de ambientes integrados de desenvolvimento	LO82	Sim
Gerência de Configuração do Software		
Uso de ferramentas de gerenciamento de configuração de software	LO133	Sim
Realização de *merge* entre códigos e/ou ramificação do código (*branching*)	LO134	Sim
Prática Profissional em Engenharia de Software		
Comportamento ético	LO199	-
Comprometimento com a aprendizagem contínua	LO326	-
Compreensão das expectativas que recaem sobre o profissional da área	LO203	Sim
Motivação e disposição	LO238	-
Reconhecimento e aprendizagem com os próprios erros, aceitar respostas de terceiros	LO233	Sim
Gerenciamento do tempo	LO234	-

Habilidades de liderança	LO240	-
Experiência com trabalho em equipe	LO227	Sim
Participação em equipes multidisciplinares	LO228	-
Pensamento crítico, geração e síntese de novas ideias	LO241	Sim
Solução de problemas	LO266	Sim
Propor soluções alternativas	LO267	Sim
Pró-atividade	LO236	Sim
Paixão por tecnologia ou pelo domínio com o qual está trabalhando	LO239	-
Experiência com projetos reais	LO269	Sim
Habilidade de visualizar o projeto como um todo (visão do todo)	LO272	Sim
Habilidade de comunicação escrita para produção de documentação técnica	LO246	Parcial
Habilidade de comunicação oral (uso de jargões e habilidades de escuta)	LO327	-
Habilidade de apresentação	LO311	-
Hesitação em solicitar ajuda	LO249	Parcial
Articulação de argumentos e fornecimento de explicações claras	LO244	-

De acordo com a tabela, para as deficiências especificamente ligadas aos testes de software, não houve contribuição para somente uma das deficiências. Entretanto, esclarecemos que dentre as atividades que os estudantes precisavam realizar, nenhuma delas solicitava que os erros encontrados fossem corrigidos, desse modo, a deficiência e o objetivo de aprendizagem relacionado não foram trabalhados pela abordagem.

Para automatizar os casos de testes, os estudantes precisaram construir e executar métodos de testes utilizando um IDE que provesse recursos para tal. Alguns estudantes relataram inclusive que usaram o depurador da ferramenta para identificar que classes/métodos deveriam ser testados. Portanto, apesar do objetivo de aprendizagem LO82[7], associado à deficiência no uso de ambientes integrados de desenvolvimento, pertencer a área de conhecimento de Construção de Software, o mesmo também foi alcançado pelo emprego da abordagem para a aprendizagem de testes de software. Consequentemente, consideramos que houve contribuição para o suprimento da deficiência.

O mesmo aconteceu para as deficiência associadas aos objetivos LO133[8] e LO134[9]. Como para implementar os métodos de testes, os estudantes precisaram lidar com o sistema de gerenciamento de configuração utilizado pela comunidade do projeto de código aberto, indiretamente, as deficiências relacionadas a essa área de conhecimento também foram trabalhadas pela abordagem. Portanto, de acordo com o próprio relato dos estudantes, consideramos que houve contribuição no atendimento das deficiências de uso de ferramentas de gerenciamento de configuração de software (LO133) e realização de *merge* entre códigos e/ou ramificação do código (LO134).

No que diz respeito as deficiências de formação relacionadas à prática profissional, vimos que houve contribuição para o suprimento de muitas delas, em virtude do alcance dos objetivos de aprendizagem associados (ver Tabela 7.11. Lembrando que para todas as competências diretamente investigadas, a maioria dos estudantes concordou com

[7]Construir, executar e depurar (*debug*) programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração (*debug*) visual.

[8]Demonstrar a capacidade de usar as ferramentas de gerenciamento de configuração, controle de versões e gerenciamento de *releases* em suporte ao desenvolvimento de um software de médio porte.

[9]Ser capaz de fazer *merge* e ramificação de código.

a contribuição da abordagem. Outras competências afloraram com base nas entrevistas. Talvez se tivéssemos perscrutado as demais deficiências de maneira mais direta, a cobertura poderia ser mais ampla, a exemplo da deficiência em 'articular argumentos e fornecer explicações claras' (LO244). Poderíamos ter utilizado algum instrumento para o registro do debate ocorrido na aula que separamos para a discussão do projeto e assim, poderíamos analisar a ocorrência de contribuição para o suprimento desta deficiência.

Quanto à deficiência de 'hesitar em solicitar ajuda' (LO249[10]), observamos que os estudantes ao executar as atividades do projeto por diversas vezes solicitaram a ajuda do suporte disponibilizado. Porém, em se tratando de interagir com a comunidade, nenhum estudante chegou a fazê-lo. Um dos estudantes, quando questionado porque não interagiu com a comunidade, demonstrou receio em perguntar algo muito simples e eles não terem paciência para respondê-lo, conforme exposto no tópico 'utilização de projetos de código aberto' em 7.2.1. Por essa razão, consideramos que a contribuição para o suprimento desta deficiência foi apenas parcial.

No próximo capítulo descrevemos e apresentamos os resultados obtidos com a adoção de um OSP para a aprendizagem de requisitos de software a partir da perspectiva de engenharia reversa.

[10]Desenvolver a habilidade de solicitar ajuda.

Capítulo

8

ESTUDO DE CASO 3 - ENGENHARIA REVERSA DE REQUISITOS

Requisitos de Software representa outra área de conhecimento definida pelo SWEBOK. A identificação de requisitos é o ponto de partida para a construção do software e é a partir dos requisitos estabelecidos, que validamos se o software atende ou não as necessidades de seus usuários. Portanto, além de guiar o desenvolvimento do software, o atendimento das necessidades dos usuários corresponde a um importante atributo de qualidade para o mesmo.

A tarefa de identificar os requisitos pode representar uma tarefa extremamente complicada, principalmente considerando os diferentes tipos de usuários que o software pode ter e características subjetivas desses mesmos usuários, que facilitam ou não a identificação. Ademais, o próprio domínio do software pode configurar um ambiente onde os requisitos mudam constantemente, seja durante o seu desenvolvimento, seja após a sua entrega. Essas e outras questões geram a necessidade do estabelecimento de processos tanto para a definição, quanto para o gerenciamento desses requisitos ao longo de todo o ciclo de vida do software.

Paralelamente, existem os sistemas legados que precisam ser mantidos, mas que dificilmente encontram-se adequadamente documentados. Muitas vezes, seus requisitos estão implementados, sem que haja qualquer tipo de especificação fora o próprio código.

Desse modo, partindo para uma abordagem complementar à tradicionalmente empregada no processo de ensino-aprendizagem de requisitos de software, onde os requisitos são identificados e o software é construído do zero, propomos a utilização de projetos de código aberto para que os estudantes recuperassem os requisitos e documentassem um software existente. Simultaneamente, como todo o processo de definição e gerenciamento dos requisitos do projeto de código aberto fica registrado nas ferramentas de comunicação *online* empregadas pela comunidade, solicitamos que os estudantes identificassem como ocorre esse processo para esse tipo de projeto e comparassem com os métodos usualmente empregados. Por último, pedimos que rastreassem as dependências existentes entre os requisitos identificados e como estes requisitos estavam implementados no código.

O propósito desse capítulo é apresentar o estudo de caso no qual utilizamos essa abordagem. Sendo assim, inicialmente descreveremos todo o contexto de aplicação da abordagem, incluindo a metodologia usada na coleta e análise dos dados. Posteriormente, reportaremos os resultados encontrados, baseado nos objetivos traçados para nossa pesquisa (Capítulo 1).

8.1 CONTEXTO DO ESTUDO DE CASO EM REQUISITOS DE SOFTWARE

Para melhor organização do texto, dividimos a descrição do contexto do estudo nos seguintes tópicos: (i) cenário do estudo de caso; (ii) população do estudo e envolvidos; (iii) aplicação da abordagem; (iv) coleta de dados e (v) análise de dados.

8.1.1 Cenário do Estudo de Caso

O cenário desse estudo caso foi a disciplina Desenvolvimento de Software I ofertada no segundo semestre de 2015[1], para o curso de Ciência da Computação da Universidade Federal de Sergipe (UFS). A disciplina, representa a primeira disciplina obrigatória na área de engenharia de software para este curso. O objetivo da disciplina é introduzir o estudante na área de engenharia de software. O ciclo de vida, características do produto de software e os principais processos de desenvolvimento são apresentados, no entanto, o principal foco é a engenharia de requisitos. Ao longo da disciplina, o estudante deverá compreender os conceitos envolvidos e capacitar-se para realizar as atividades de identificação, análise, documentação, validação e gerenciamento de requisitos.

Para a exposição do conteúdo, o professor aplicou o método tradicional de ensino, com aulas expositivas e exercícios. Para exercitar a identificação e descrição de casos de uso, bem como, os diagramas UML estudados, o professor propôs exercícios usando exemplos simplificados de possíveis softwares, por exemplo, para atender uma clínica veterinária, um controle acadêmico, etc. Dentre outras atividades de avaliação, os estudantes tiveram que fazer dois projetos práticos. O primeiro projeto prático, seguindo a perspectiva de construção de software a partir do zero, usualmente ensinada, correspondeu ao levantamento de requisitos, elaboração do diagrama de atividades e definição, diagramação e especificação de casos de uso para um software idealizado pelos próprios estudantes. Incorporamos a abordagem de utilização de um projeto de código aberto usando a estratégia de trabalhar os requisitos do ponto de vista de um projeto já existente, como atividade obrigatória, correspondente ao segundo projeto prático. O conteúdo teórico necessário para a realização das atividades já havia sido de algum modo tratado pelo professor, antes de propormos o trabalho. Dessa maneira, o conteúdo foi apresentado de maneira independente da abordagem e a abordagem serviria para complementar a aprendizagem sobre requisitos, com base nessa nova perspectiva.

[1]O segundo semestre de 2015 ocorreu no período de janeiro à maio de 2016, em virtude do atraso no calendário acadêmico da Universidade.

8.1.2 População do Estudo e Envolvidos

Nossa população de estudo foi composta pelo professor e os estudantes da disciplina. Sendo que dos 26 matriculados, 19 estudantes aceitaram participar da pesquisa inicialmente. Porém, apenas 15 responderam ao questionário final.

Seguindo a grade curricular do curso de Ciência da Computação, a disciplina encaixa-se no quinto período, dentro dos oito períodos totais do curso. Portanto, os estudantes ao cursá-la estão finalizando as disciplinas básicas da computação e iniciando as profissionalizantes. Do ponto de vista de experiência profissional, a grande maioria dos estudantes não tinha tido nenhuma experiência com projetos reais, apenas três haviam participado de um ou dois projetos até o momento. Essas condições demonstram que os estudantes encontram-se ainda razoavelmente imaturos quanto à formação acadêmica e profissional, situação esperada para estudantes na metade do curso.

No caso do professor, descrevemos suas características e concepções no Capítulo 7 em 7.1.2, uma vez que as disciplinas nas quais realizamos os estudos foram ofertadas pelo mesmo professor. O mesmo acontece para as nossas concepções. A única diferença é que nesse estudo não houve a necessidade de um assistente.

8.1.3 Aplicação da abordagem

Para a aplicação da abordagem, o primeiro passo foi ajustar os objetivos da pesquisa ao programa da disciplina. Como o foco da disciplina era requisitos de software, selecionamos os objetivos de aprendizagem relacionados a essa área de conhecimento estabelecidos no Capítulo 3 que adequavam-se ao programa da mesma e ao projeto de código aberto utilizado, no caso, o JabRef. Decidimos continuar usando o JabRef, software também empregado nos estudos de caso anteriores, por duas razões. A primeira delas foi a execução em paralelo dos dois estudos de caso, este e o descrito no Capítulo 7. Escolher outro OSP representaria um aumento de sobrecarga de trabalho, ao mesmo tempo em que, trabalhando com o mesmo projeto, aumentaria nosso conhecimento sobre o mesmo, possibilitando-nos a dar melhor suporte aos estudantes. A outra razão relaciona-se às variáveis da pesquisa, trabalhando com o mesmo software diminuiríamos o número de variáveis entre os estudos.

Do mesmo modo que no estudo de caso sobre testes de software (Capítulo 7) outras atividades foram necessárias para o emprego da abordagem, além do alinhamento do objetivos e escolha do software. A Tabela 8.1 elenca todas as atividades que precisamos realizar juntamente com o esforço despendido em sua realização (em número de horas). Nesse caso, o número total de horas foi bastante reduzido, uma vez que, como dito, utilizamos o mesmo software dos estudos anteriores. Portanto nenhum esforço foi contabilizado para as atividades relacionadas à definição e compreensão do mesmo. Considerando o material disponibilizado, como o professor de alguma maneira havia tratado do conteúdo necessário às atividades, com exceção da introdução sobre projetos de código aberto e ao próprio JabRef e mais as explicações sobre a realização das atividades, não disponibilizamos nenhum outro material.

Ao compararmos com o segundo estudo de caso descrito (Capítulo 7), constatamos que gastamos mais tempo no estabelecimento e alinhamento dos objetivos de aprendiza-

Tabela 8.1 Atividades necessárias para emprego da abordagem

Atividades necessárias	**Esforço necessário (horas)**
Estabelecer os objetivos de aprendizagem pretendidos.	5
Identificar as possíveis atividades que com o uso do projeto de código aberto poderiam ajudar com que os estudantes alcançassem os objetivos de aprendizagem pretendidos.	5
Pesquisar e definir o projeto de código aberto a ser utilizado.	-
Compreender o software (leitura da documentação existente e uso das funcionalidades).	-
Compreender o software (código)	-
Preparar slides de apresentação do projeto prático e introdução ao projeto de código aberto a ser utilizado	2
Preparar material de suporte à configuração do ambiente a ser utilizado	-
Detalhar as atividades a serem realizadas pelos estudantes	16
Preparar material de apoio aos estudantes	-
Dar suporte as atividades dos estudantes	6
Avaliar as soluções apresentadas	14
Dar o feedback sobre as soluções apresentadas	8
Total	56

gem pretendidos com as atividades que seriam realizadas pelos estudantes, para o alcance desses objetivos. Essa situação pode ser explicada pela perspectiva totalmente nova que estamos sugerindo de trabalhar requisitos tendo como base um projeto existente. Por conseguinte, precisamos analisar com mais cuidado que atividades estaríamos propondo. Por outro lado, consumimos menos tempo tanto para dar suporte aos alunos, quanto para avaliar as soluções apresentadas pelos mesmos, uma vez que as atividades a serem realizadas eram mais simples. Apenas a execução do rastreamento entre requisitos e código exigia que os estudantes lidassem com o código do software. Esses fatos demonstram que o esforço despendido de aplicação da abordagem pode variar a depender também do foco de aplicação da mesma, porém as atividades necessárias são muito similares.

Dividimos o projeto em duas etapas, sendo que as respectivas atividades deveriam ser realizadas pelos estudantes organizados em equipes. O foco da primeira etapa era desenvolver uma visão geral sobre o software e sobre o processo de definição e gerenciamento dos requisitos em projetos de código aberto, enquanto que na segunda, cada equipe trabalharia outras questões sobre requisitos, para determinado módulo do software.

Na primeira etapa, as atividades solicitadas foram:

- Avaliar a documentação existente;
- Elaborar o modelo conceitual e o diagrama de casos de uso a partir da utilização do software;
- Responder as questões discursivas:
  - Descrever como é o processo de engenharia de requisitos em projetos *open source* (como ocorre a elicitação, análise, especificação e validação de requisitos).

- Comparar os processos de requisitos empregados nos modelos Cascata, Ágil e *Open Source* (identificando riscos e benefícios).
- Por que é importante gerenciar os requisitos?
- Identificar e descrever como os requisitos são gerenciados em projetos *open source*.
- Comparar a abordagem existente em projetos *open source* com formas de gerenciamento de requisitos que devem ser seguidas em empresas que desenvolvem software.

Com cada equipe trabalhando determinado módulo do software, na segunda etapa, os estudantes tiveram que executar as seguintes atividades:

- Identificar os requisitos funcionais e não funcionais envolvidos no módulo;
- Montar a matriz de rastreabilidade entre os requisitos;
- Elaborar o diagrama de sequência do sistema para as interações existentes no caso de uso estabelecido, identificando classes e métodos envolvidos;
- Propor melhorias para o *Help* da funcionalidade correspondente ao caso de uso estabelecido;
- Responder as questões discursivas:
  - A matriz de relacionamento entre os requisitos do módulo construída representa um exemplo de rastreabilidade de requisitos. Para que serve esse tipo de rastreabilidade?
  - Ao identificar os componentes que implementaram determinada funcionalidade, vocês estão realizando o que é chamado de rastreamento de "recuperação" (*backward tracing* - rastreamento que é realizado a partir do código). Para que serve esse tipo de rastreamento?
  - Digamos que ao longo do desenvolvimento do projeto, a equipe do JabRef tivesse registrado os componentes que implementavam cada requisito. Poderíamos dizer que a equipe estaria fazendo o rastreamento de "construção" (*forward tracing* – rastreamento que é realizado a partir dos requisitos). Para que serve esse tipo de rastreamento?

De acordo com o professor da disciplina, comparando com as atividades propostas no estudo de caso sobre testes, disciplina pela qual também era responsável, as atividades nesse estudo foram mais simples e em uma quantidade também menor. Dessa forma, foram mais adequadas, visto que, os estudantes eram mais imaturos.

> *"(...) a princípio, eles realmente tiveram atividades mais simples, mas... provavelmente deve... deve ter sido... bom isso, porque tão começando (...) talvez*

> *a coisa tenha sido certa também, nesse aspecto... uma quantidade de atividades práticas um pouco menor, em comparação com DS III, mas pra alunos que realmente tavam no começo, imaturos. Eu acho que também ficou de bom tamanho"* (P2).

Ao longo do andamento do trabalho, realizamos três encontros presenciais com a turma. Os objetivos e datas de cada encontro, bem como o material disponibilizado aos estudantes, encontram-se detalhados nas tabelas 8.2 e 8.3, respectivamente. Convém ressaltar que além dos encontros, ficamos em contato constante com os estudantes através do correio eletrônico, dando suporte e esclarecendo dúvidas.

Tabela 8.2 Encontros presenciais da pesquisadora com os estudantes no estudo EC3.

Encontro	Objetivo	Data
1	Auto-apresentação. Apresentar o projeto de pesquisa, identificar os estudantes que participariam da pesquisa, obter a assinatura dos TCLEs, levantamento da experiência dos estudantes, apresentar o JabRef	07/04/2016
2	Discussão sobre o modelo conceitual e casos de uso do JabRef. Definição de qual módulo do software seria trabalhado por cada equipe. Orientações para a segunda parte do trabalho.	28/04/2016
3	Discussão sobre as questões discursivas da primeira parte do trabalho e as demais atividades da segunda parte do trabalho.	17/05/2016

Tabela 8.3 Material disponibilizado no estudo EC3.

Encontro	Título	Data
1	Apresentação do Projeto Prático	Contextualizar e motivar o estudante para a execução do projeto. Descrever as atividades que seriam realizadas e como seria o andamento do projeto.
1	Apresentando o JabRef	Introduzir os objetivos do JabRef e destacar porque a utilização de um sistema de gerenciamento de referências é importante. Explicar como é o processo de desenvolvimento de um projeto de código aberto e mostrar ao estudante que pode ser interessante participar de um projeto desse tipo.
1	Configuração ambiente JabRef	Apresentar o passo-a-passo de como os estudantes criariam o ambiente de desenvolvimento do JabRef em seus computadores.
2	Instruções detalhadas para a segunda etapa	Instruções para realização da segunda parte do trabalho e questões que deveriam ser respondidas pelas equipes.

Sentimos algumas dificuldades ao longo da aplicação da abordagem. A primeira refere-se à definição de quais atividades seriam propostas, dada a novidade da perspectiva de trabalhar os requisitos sob o ponto de vista de um projeto existente. Ocorreram várias dúvidas sobre o que deveria ser sugerido, para complementar e ao mesmo tempo ser interessante para a aprendizagem dos estudantes, pela proximidade do que pode acontecer futuramente, quando os mesmos estiverem trabalhando em alguma empresa.

O alinhamento dos objetivos de aprendizagem ao planejamento da disciplina representou outra dificuldade. Para não sobrecarregarmos os estudantes, decidimos não trabalhar

alguns objetivos de aprendizagem relacionados a casos de uso e utilização de ferramentas (LO5[2] e LO10[3], respectivamente), inclusive por acreditar que o professor havia tratado esses assuntos suficientemente.

O uso do JabRef como projeto de código aberto a ser manuseado pelos estudantes também limitou os objetivos de aprendizagem que poderíamos trabalhar com a abordagem. Como não havia nenhum modelo ou documento de especificação dos requisitos, solicitamos que os estudantes simplesmente analisassem a documentação existente sem que nenhuma técnica fosse aplicada, consequentemente, não trabalhamos os objetivos LO186 (Descrever técnicas para verificação e validação dos demais artefatos gerados pelo projeto além do código) e LO9 (Conduzir uma revisão de um conjunto de requisitos de software para determinar a qualidade dos mesmos, no que diz respeito às boas características de requisitos (consistência, validade, completude e viabilidade)).

Ademais, para não interferirmos muito no andamento da disciplina, realizamos apenas três encontros presenciais. Como aproveitamos esses encontros para discutirmos as soluções para as atividades propostas e todo o trabalho foi realizado no final do período letivo, tivemos que enviar eletronicamente os comentários específicos sobre cada trabalho.

Por último, visto que realizamos esse estudo paralelamente ao estudo descrito no Capítulo 7, sentimos a mesma dificuldade de mensurar a complexidade das atividades propostas, particularmente considerando à atividade que exigia a manipulação do código do OSP.

8.1.4 Coleta de Dados

O método de coleta de dados foi idêntico ao aplicado no estudo de caso de testes de software. Então, seguindo o mesmo procedimento descrito no Capítulo 7, aplicamos os questionários, realizamos as entrevistas e a análise de documentos.

Aplicamos o primeiro questionário (LER), logo no primeiro encontro com os estudantes (ver tabela 8.2), a fim de identificar a experiência dos estudantes com projetos reais. O segundo (QR1) foi repassado após o primeiro encontro, com o objetivo de capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a requisitos de software, antes da aplicação da abordagem. Por fim, ministramos o terceiro questionário (QR2) depois que encaminhamos os comentários dos trabalhos, com a intenção de obter a percepção dos estudantes sobre o seu conhecimento e habilidades sobre requisitos, após o emprego da abordagem, bem como, a contribuição do projeto para aquisição desse conhecimento e o desenvolvimento dessas habilidades. No mesmo questionário, ainda buscamos detectar a percepção do estudante: (i) sobre a utilização do projeto de código aberto como uma experiência real de desenvolvimento de software; (ii) se as atividades realizadas permitiram que o mesmo fizesse a conexão entre teoria e prática; (iii) sobre a contribuição do projeto para o desenvolvimento de competências relacionadas à prática profissional em engenharia de software; (iv) se a participação no

[2]Listar os principais componentes de um caso de uso ou descrição similar para algum comportamento requerido de um software.

[3]Demonstrar a capacidade de usar ferramentas de análise de requisitos em suporte ao desenvolvimento de um software de médio porte.

projeto fez com que o mesmo se sentisse mais preparado para o mercado de trabalho, e, por fim, (v) se a participação no projeto contribuiu para a conscientização do mesmo sobre a importância de alguns aspectos da engenharia de software.

As estruturas dos questionários LER, QR1 e QR2 encontram-se nos apêndices E, F e G, respectivamente. Enquanto o LER foi aplicado presencialmente para os estudantes que aceitaram participar da pesquisa, o QR1 e o QR2 foram enviados e respondidos eletronicamente. Para que fosse possível aplicarmos a análise pretendida, solicitamos a identificação dos respondentes em todos os questionários. Na conjuntura desse estudo, dos 15 estudantes que responderam o QR2, apenas nove responderam também o QR1. Portanto ao compararmos as respostas de QR1 e QR2, para as questões cabíveis, somente nove questionários foram considerados.

Para as entrevistas, como os módulos do software foram divididos entre as equipes formadas, procuramos entrevistar ao menos dois membros de cada equipe, porém, enquanto o estudante SR4, apesar de não ter sido convidado, voluntariou-se, um membro da equipe ER4 não compareceu. Quanto à equipe ER5, nenhum membro participou efetivamente do projeto. A Tabela 8.4 caracteriza os estudantes entrevistados por equipe e experiência declarada. Da mesma forma do estudo de caso sobre testes (Capítulo 7) , ao convidarmos os estudantes para as entrevistas, também procuramos cobrir proporcionalmente a diferença de experiência prévia com projetos reais apresentada pelos mesmos. Contudo neste caso, o único estudante da turma mais experiente não aceitou participar da pesquisa, consequentemente cobrimos apenas, as situações de 'alguma experiência[4]' e 'sem experiência[5]'.

Tabela 8.4 Caracterização dos estudantes entrevistados no estudo EC3.

Equipe	**Módulo do Software**	**Estudante**	**Experiência prévia**
ER1	Operações sobre bases de dados	SR2	Sem experiência
		SR4	Sem experiência
		SR5	Sem experiência
ER2	Operações sobre as entradas	SR1	Alguma experiência
		SR3	Sem experiência
ER3	Organização das entradas em grupos	SR6	Sem experiência
		SR7	Sem experiência
ER4	Operações sobre bases de dados	SR8	Alguma experiência

Cabe lembrar que como empregamos os mesmos procedimentos do estudo sobre testes, as entrevistas foram realizadas após o término das aulas, todavia, antes de encaminharmos os comentários sobre os trabalhos e antes que os estudantes respondessem o questionário QR2, finalmente, que entrevistamos o professor, ao término de todas as atividades.

Ainda como fonte de informação, exploramos os documentos entregues pelos estudantes. O propósito era examinar a aprendizagem demonstrada pelos mesmos.

[4]Estudantes que participaram de até dois projetos anteriormente.

[5]Estudantes que até o momento não tinham tido nenhuma experiência com projetos reais.

8.1.5 Análise de dados

No que diz respeito a análise de dados, para os dados qualitativos, aplicamos a análise descrita no Capítulo 4. Para os dados quantitativos, dado o pequeno número de participantes, utilizamos estatísticas descritivas para a análise. Somente para compararmos a aprendizagem antes e depois da aplicação da abordagem, recorremos ao teste de postos sinalizados de Wilcoxon para amostras pareadas, uma vez que os dados não apresentavam distribuição normal.

Cabe realçar que para a análise da contribuição do projeto para o alcance dos objetivos de aprendizagem sendo tratados (que representam algumas das questões presentes no QR2), utilizamos somente as respostas dos nove estudantes que também responderam ao QR1. Tomamos esta decisão com vistas a que pudéssemos chegar a outras conclusões, através da triangulação entre essas informações, conforme pode ser constatado ao longo da discussão sobre os resultados.

8.2 RESULTADOS

Nesta seção, apresentaremos os resultados obtidos com a realização desse estudo de caso. As evidências de acordo com os objetivos traçados para a pesquisa estão assim organizadas: (i) experiência real de desenvolvimento de software (Ob7); (ii) conexão teoria e prática (Ob6); (iii) objetivos de aprendizagem alcançados (Ob4); (iv) sentimento de preparação para o mercado de trabalho (Ob8); (v) consequências para a aprendizagem(Ob5) e, por último, (vi) lacunas na indústria de software (Ob9).

8.2.1 Experiência Real de Desenvolvimento de Software (Ob7)

A fim de obtermos objetivamente a percepção do estudante sobre a utilização de um projeto de código aberto como uma experiência real de desenvolvimento de software, empregamos as mesmas quatro afirmações usadas no estudo de caso sobre teste, as quais foram: (i) Você acredita que o projeto empregado (JabRef) permitiu que você tivesse uma experiência de mundo real (Q1R1); (ii) Você acredita que o projeto empregado (JabRef) representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software (Q2R2); (iii) O uso do projeto de código aberto (JabRef) permitiu que você visualizasse a complexidade e as dificuldades existentes em um projeto real de software (Q3R3); e (iv) A participação no projeto permitiu que você compreendesse melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software (Q4R4). A Figura 8.1 sumariza os resultados obtidos.

Através da figura, verificamos que todos os estudantes concordaram que a utilização do projeto de código aberto (JabRef) permitiu que eles tivessem uma experiência de mundo real; visualizassem a complexidade e as dificuldades existentes em um projeto real; e finalmente, que compreendessem melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software. Apenas para o aspecto do JabRef representar um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software houve uma discordância, o que mesmo assim, representa um nível de concordância de 93,33%. Cabe ainda ressaltar que 60% dos estudantes concordaram

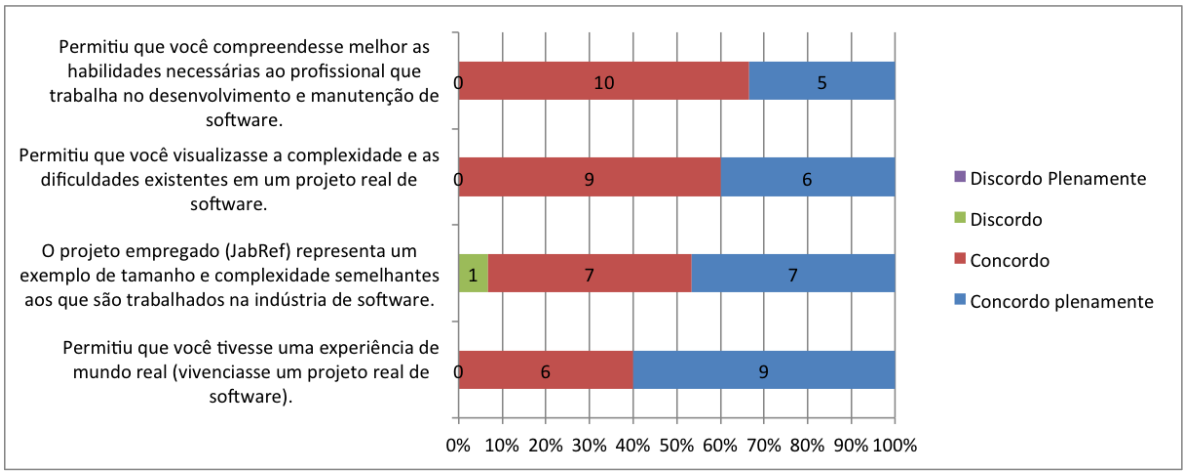

Figura 8.1 Níveis de concordância sobre a abordagem como uma experiência real de desenvolvimento de software no estudo EC3.

plenamente quanto à utilização do OSP como uma experiência de mundo real.

Qualitativamente, capturamos a percepção dos estudantes a respeito do uso do projeto de código aberto como experiência de mundo real sobre dois aspectos: proximidade dos softwares existentes nas empresas e proximidade das atividades realizadas na indústria. A Figura 8.2 detalha as características reveladas para esses aspectos. Essas características somadas às dificuldades encontradas em lidar com esse tipo de projeto contribuíram para que os estudantes elaborassem uma visão sobre as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software, também retratada na Figura 8.2. A seguir discorreremos sobre todos esses aspectos.

Proximidade dos softwares existentes nas empresas

Identificamos algumas características do OSP utilizado que o aproximam dos softwares existentes nas empresas e com os quais os engenheiros de software precisam saber trabalhar, são elas: (i) escassez de documentação; (ii) complexidade; (iii) estrutura confusa e (iv) não representar uma simplificação.

No que diz respeito à escassez de documentação apresentada pelo projeto empregado, os estudantes consideraram como principal problema, a falta de documentação do código fonte, seja pelos nomes dos métodos serem pouco intuitivos, seja pela ausência de comentários em geral.

> *"E quando foi no código em si, vendo aquele código ali, não estava muito documentado em questão de implementação. Ele estava muito completo em relação a como utilizar, mas como documentação de implementação, não. Você olhava as funções e você tinha que tentar entender a função, porque não dizia, realmente, a função, muitas delas o próprio nome não dava para identificar"* (SR1).

> *"Realmente muito mal documentado, sem comentário no código-fonte (...)*

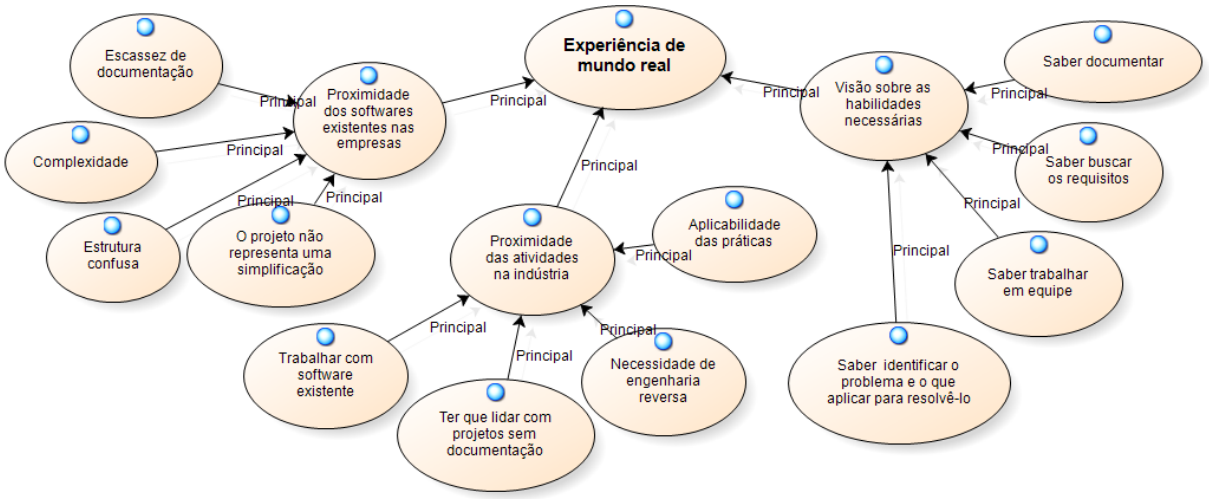

Figura 8.2 Percepção dos estudantes sobre a proximidade do projeto a uma experiência real de desenvolvimento e manutenção de software no estudo EC3.

> *o que essa classe faz, o que esse objeto... essa instância do objeto tá fazendo... ele não, não tem essa descrição, né... Nem, nem descrevendo em si os métodos, 'ah, esse método salva o arquivo', 'esse método salva as bases de dados abertas'. Isso... eu acho que em nenhum ponto do código que eu cheguei a olhar, existia esse tipo de... é... informação"* (SR8).

Contudo, um dos estudantes acrescentou que esse fato não será muito distinto do que vão encontrar futuramente.

> *"(...) não vai ser muito diferente do que vai ser encontrado... no mercado... Não vão ser todos os projetos que vai estar tudo claro, tudo documentado de forma clara, que auxilie não só o usuário e também... como também o desenvolvedor"* (SR5).

Quanto à complexidade do software, um estudante considerou que a complexidade do JabRef não era adequada para estudantes sem experiência.

> *"(...) é um programa muito complicado (...) pra quem não tem uma certa... prática, fica difícil entender (...)"* (SR8).

Inclusive esse mesmo estudante, comparando com softwares com os quais ele já está trabalhando em seu estágio, considerou o JabRef muito mais complexo.

> *"Eu estou estagiando agora, já tenho contato com os sistemas lá da empresa e assim... eu achei a estrutura do sistema da empresa muito mais fácil de entender... e muito mais fácil de operar, modificar, documentar, do que o... o JabRef"* (SR8).

Em contrapartida, outro estudante, mesmo sem ter experiências anteriores com projetos reais, considerou essa complexidade adequada.

> *"(...) achei na medida... não achei muito complexo não"* (SR7).

Possivelmente este estudante espera encontrar softwares ainda maiores e mais complexos futuramente nas empresas, uma vez que foi o único estudante que discordou em relação ao JabRef representar um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software (Q2R2), ao responder ao questionário. Essa perspectiva de complexidade para futuros softwares é compartilhada por mais estudantes.

> *"Eu acredito que seria mais complexo... um pouquinho mais..."* (SR6).

Outro estudante também não considerou o software tão complexo e complementou que a complexidade existente ocorre em virtude da falta de comentários no código mencionada.

> *"Não achei ele complexo, não... agora, apenas a parte da codificação dele... que quando foi aberta a parte para fazer a questão dos métodos, ficou um pouco complicado, por causa da falta dos... até de comentários, que poderiam nortear (...) dificultou muito"* (SR5).

Porém para o estudante que considerou o software mais complexo do que os que ele trabalha, além da documentação ser ruim e não existirem comentários no código fonte, o tamanho do software e a estrutura confusa do código colaboram para esta complexidade.

> *"Eu diria que ele é mais complexo... por causa da, da questão da quantidade de classes e métodos que ele tem... e... um pouco... um pouco da... posso dizer assim... da... organização... a estrutura do sistema, que eu achei um pouco confusa em relação aos projetos que eu trabalho"* (SR8).

Os estudantes podem ter considerado a estrutura do software confusa porque ela foge do usual, uma vez que as telas são construídas dinamicamente de acordo com as preferências do usuário, como descrito no extrato a seguir.

> *"(...) ele é bem complexo mesmo, porque eu sempre sou acostumado a fazer um software e criar apenas uma classe de interface. Já nele lá tem várias classes e aí a própria interface dele vai se modificando de acordo com o que o usuário põe. Ali foi muito complicado pra identificar essa parte, porque ele vai criando os botões de acordo com o que o cliente escolhe lá, o que ele marcou nas opções"* (SR1).

A última característica que aproxima o OSP dos softwares existentes nas empresas é a de não representar uma simplificação. Usualmente os estudantes estão acostumados a lidar com projetos simples ao longo das disciplinas.

> *"(...) são trabalhos bem mais simples, da faculdade (...)"* (SR1).

> *"Os projetos que eu desenvolvi nas disciplinas (...) são projetos simples, como construir um jogo, ou então uma ferramenta para conversar com o banco de dados"* (SR3).

Após a experiência com o OSP, este mesmo estudante constatou que um projeto real é mais complexo porque existem inúmeros detalhes que precisam ser tratados e que quanto mais funcionalidades ele contiver, mais complicado será.

> *"E é bem diferente, a complexidade da ferramenta é muito superior, muito superior. Tem muitos detalhes que devem ser tratados, então é bem, é bem mais simples o meu projetozinho aqui, do que a ferramenta, com certeza (...) E outros tipos de ferramenta que fazem coisas muito utilizadas, que trazem mais funcionalidades, pronto... com certeza vão ser mais, mais complicadas"* (SR3).

Por fim, o primeiro estudante citado que admitiu que os projetos trabalhados nas disciplinas são bem mais simples externou a sua preocupação em não estar preparado para lidar com projetos reais, exatamente por não lidar com esse tipo de projeto ainda na graduação.

> *"(...) é preocupante você não estar preparado diante desses softwares que você vai encontrar no mercado, porque até agora eu nunca tive nenhum contato realmente..."* (SR1).

Proximidade das atividades na indústria

Sobre a proximidade das atividades executadas no trabalho, às atividades que são realizadas na indústria, encontramos as seguintes características (ver Figura 8.2): (i) trabalhar com software existente; (ii) ter que lidar com projetos sem documentação; (iii) necessidade de realizar engenharia reversa, e (iv), aplicabilidade das práticas.

A partir de sua participação no projeto, um dos estudantes apercebeu-se que na indústria, nem sempre o profissional vai se envolver apenas com projetos iniciados do zero. Ele poderá ter que dar manutenção em softwares existentes. Portanto, é necessário aprender o que fazer nessas situações.

> *"(...) 'jogue numa empresa e dê manutenção nesse software', eu nunca cheguei a fazer isso. E depois de ter essa experiência vi que a coisa vai ser séria. É o que o mercado espera então é preciso se atualizar, não ficar só nesse método tradicional de se elaborar desde o começo"* (SR1).

Ao ter que trabalhar com um software existente, o estudante esperava que o mesmo estivesse adequadamente documentado. Contudo, a participação no projeto permitiu-lhe constatar que isso é muito difícil de acontecer. Na realidade, o profissional precisa lidar com projetos sem documentação.

> *"Se fosse dar manutenção, teria uma documentação bem-feita, e foi visto na prática que isso não existe"* (SR1).

Consequentemente, a engenharia reversa executada no projeto representa o que muitas vezes os estudantes precisarão executar quando estiverem trabalhando na indústria de software.

> *"E em si essa engenharia reversa que fomos vistos [vimos] que é o que acontece no mercado, na qual a gente vai pegar produtos... é... prontos no mercado e vamos estar sujeitos a fazer manutenções no que outros programadores já fizeram, cheio de remendos, e não bem organizado (...) quem seguir a carreira de programação no mercado, provavelmente vai se bater muito com isso"* (SR1).

Outro estudante ainda ressaltou que os métodos utilizados no decorrer das atividades com OSP correspondem exatamente ao que ele precisa fazer no estágio e o que precisará fazer futuramente no ambiente de trabalho, demonstrando assim a aplicabilidade das práticas executadas.

> *"(...) e essa questão dessa documentação, assim... mais aprofundada... Então, eu acho que tive a chance de poder realmente pegar uma ferramenta que é o que realmente eu estou usando no estágio, é o que eu vou usar quando sair daqui da universidade, e os métodos são os métodos que eu vou ter que usar no, no ambiente de trabalho"* (SR8).

Dificuldades

A vivência de um projeto com características semelhantes às encontradas nos softwares existentes nas empresas e às atividades que são realizadas nessas empresas geram dificuldades a serem enfrentadas pelos estudantes. Classificamos tais dificuldades em dificuldades provenientes da utilização de projetos reais e dificuldades provenientes da utilização de projetos de código aberto.

Dificuldades provenientes da utilização de projetos reais

Todas as vezes que nos defrontamos com um projeto novo, ocorre algum impacto. Contudo a dimensão desse impacto depende do tamanho do desafio proposto, ou seja, do tamanho e complexidade do software e das dificuldades para a realização das atividades. Consequentemente, a utilização de projetos reais sempre gerará algum tipo de impacto. Nesse estudo de caso, os estudantes relataram exatamente a dificuldade desse impacto inicial, ao se depararem com o projeto de código aberto proposto.

No extrato a seguir, o estudante deu a entender que após o impacto inicial, ele conseguiu transpor as dificuldades.

> *"Eu acho que... foi um pouco complicado de entender no começo... deu um pouco de trabalho... mas... fora assim... aquele impacto inicial..."* (SR8).

Outro estudante relatou o impacto, como a dúvida sobre a capacidade em realizar a tarefa solicitada.

> *"(...) pegar a ferramenta... ache os estudos de caso [casos de uso] dela e faça o modelo de... de... modelo conceitual. Aí foi aquele impacto... que você não conseguiria achar (...)"* (SR1).

Por outro lado, um terceiro estudante considerou a novidade de utilizar o OSP como o impacto, não a questão de utilizar um projeto real, apesar que, ele não tinha experiência com projetos reais anteriormente.

> *"(...) a questão daquele primeiro contato do... o susto, né, que leva a pessoa quando vê assim, um projeto open source... que... acredito que muitos, ali foi o primeiro contato mesmo com um projeto open source... E foi o novo... chega a assustar, né, o novo (...) A questão do JabRef foi... acho que a dificuldade foi o novo mesmo"* (SR5).

Por último, o professor também percebeu essa dificuldade inicial por parte dos estudantes, todavia a associou a falta de experiência dos mesmos, uma vez que as atividades solicitadas não eram muito complexas.

> *"(...) uma aplicação já desenvolvida, então aí já tem a dificuldade de olhar um código e tal. Aí é onde eles tiveram algum choquezinho, porque como são novinhos, não tem experiência, então tiveram essa dificuldade... Mas aí então, aí já não foi uma coisa tão grande. Eu acho que a maior dificuldade foi só essa, deles... 'ah, o que é isso?', 'aquele negócio'..."* (P2).

Vários estudantes relataram como dificuldade a própria novidade da abordagem de engenharia reversa associada à utilização de um projeto real, proposta.

> *"E pegar do zero assim, e você ver o que tá funcionando, o jeito que ele funciona e depois tentar descrever ele, é algo complexo"* (SR2).

> *"(...) como foi... o novo pra gente... é... e com isso surgiram dúvidas, tinham dúvidas a respeito de como fazer, porque era o primeiro contato com esse tipo de projeto..."* (SR5).

O extrato da entrevista com um dos estudantes também ilustra essa dificuldade.

> Pesquisadora: *"Por que você disse que foi complicado?"*
>
> SR7: *"Porque eu nunca tinha feito... tinha feito essa... como é que chama... é... programação reversa."*
>
> Pesquisadora: *"Engenharia reversa."*
>
> SR7: *"Essa engenharia reversa (...) Dificultou muito por não ter experiência., né... Nunca tinha pegado um código... completo daquele jeito. Tinha que olhar classe por classe, tinha que pegar cada requisito e pegar a funcionalidade daquele requisito no código, onde ele está implementado. Como isso é a primeira vez que eu fiz!! Isso dificultou muito."*

Inclusive, outro estudante considerou tal dificuldade como o maior desafio.

> *"A maior dificuldade primeiro é que você se debateu com uma novidade, que você nunca pegou. A gente nunca pegou um software para trabalhar em cima dele, a gente sempre partiu de algo para resolver o problema do zero, implementar o problema do zero, sem reaproveitar alguma coisa, ou então ver onde ele tá e procurar, a gente nunca fez isso. Acho que a maior dificuldade foi essa, pegar um código pronto e tentar identificar as coisas que estão implementadas nele. Em si, essa foi a maior dificuldade"* (SR1).

Contudo para o estudante anterior, a maior dificuldade foi entender o código desenvolvido por terceiros.

> *"A primeira dificuldade foi entender o código... essa foi a maior dificuldade (...) Acho que foi mais na questão de que eu é que não tinha escrito o código..."* (SR7).

Dificuldade sentida por mais um estudante.

> *"A interpretação do código fonte teve muita dificuldade"* (SR6).

Dentre as dificuldades para entender o código de antemão citadas, realçamos a falta de documentação, destacada por vários estudantes.

> *"(...) nunca peguei nada que não tenha documentação, não tinha nada. Eu procurei, não tinha absolutamente nada"* (SR2).

> *"Então... realmente foi a questão da ausência de informação, da documentação"* (SR3).

> *"Com relação à documentação foi um pouco mais difícil... é... localizar algumas informações..."* (SR5).

Outra dificuldade apontada pelos estudantes foi a de entender as funcionalidades disponibilizadas pelo software, o que muitas vezes pode acontecer quando projetos reais são utilizados.

No extrato a seguir, o estudante declarou que o software era pouco intuitivo

> *"Eu procurei entender, ele é pouco intuitivo (...) De início foi o entendimento da aplicação, que eu achei um pouco complexa (...) a gente foi pesquisando e olhando até as configurações do JabRef. Foi esclarecendo mais e acabou... facilitando no final"* (SR4).

Sentimento compartilhado por mais um estudante.

> *"Então, assim, Eu esperava que, que ele fosse mais fácil de usar. Eu tive algumas dificuldades pra... até pra alterar... fazer algumas edições... eu acho que não tava muito claro, não tava fácil de encontrar algumas... aplicações dele. Então, podia ser mais fácil. (...) acho que algumas aplicações estão meio que escondidas, mas ele não é um... um sistema muito complexo"* (SR6).

Um terceiro estudante admitiu que essa dificuldade era oriunda da quantidade de funcionalidades existente e da documentação inadequada.

> *"E também um pouco de dificuldade para entender alguns detalhes da ferramenta, como são muitas funcionalidades, às vezes eu demorava... Depois eu encontrava, mas eu demorava um tempo para entender o que aquela funcionalidade faz e como às vezes nem tudo existe na documentação, aí a pessoa tinha que sair fuçando mesmo a ferramenta, testando e verificando... Tinha que ser na força bruta... não teve outro jeito"* (SR3).

Utilização de projetos de código aberto
Além das dificuldades mencionadas para a utilização de projetos reais, especificamente em virtude da utilização de projetos de código aberto, um dos estudantes relatou a dificuldade de obter informações, diferentemente dos projetos existentes nas empresas.

> *"(...) pela questão dele ser open source... Não tinha muita informação, assim, disponível para dar um caminho a seguir (...) eu digo no sentido de informação disponível... até mesmo pra encontrar o... o núcleo de desenvolvimento, o fórum pra tirar as dúvidas... pra... pegar alguma indicação de alguma informação... fica um pouco... foi o que... onde se deu a maior dificuldade no trabalho. O que em projetos que... vamos supor que não sejam open source têm mais facilidade na informação"* (SR5).

Porém ao questionarmos se houve interação com a comunidade do projeto, para esclarecer qualquer tipo de dúvida, o estudante respondeu:

> *"Houve mais leitura nos fóruns. Mas foram poucas, não foram muitas não. Foram poucas"* (SR5).

Supomos que houve certo receio em interagir diretamente com a comunidade e pouca motivação para leitura do histórico de andamento do projeto.

Visão sobre as habilidades necessárias
A experiência proporcionada pelo projeto propiciou que os estudantes construíssem uma visão sobre algumas das habilidades necessárias aos engenheiros de software. As habilidades mencionadas foram (ver Figura 8.2): (i) saber buscar os requisitos; (ii) saber documentar; (iii) saber trabalhar em equipe, e (iv), saber identificar o problema e o que aplicar para resolvê-lo.

8.2.2 Conexão teoria e prática (Ob6)

Outro objetivo da pesquisa foi perscrutar se a prática com um projeto de código aberto permite que o estudante compreenda como o conhecimento teórico sobre requisitos é aplicado na prática em um projeto real e se a prática em projetos reais pode ser importante para a aprendizagem dessa teoria. A Figura 8.3 revela as percepções dos estudantes sobre esses aspectos.

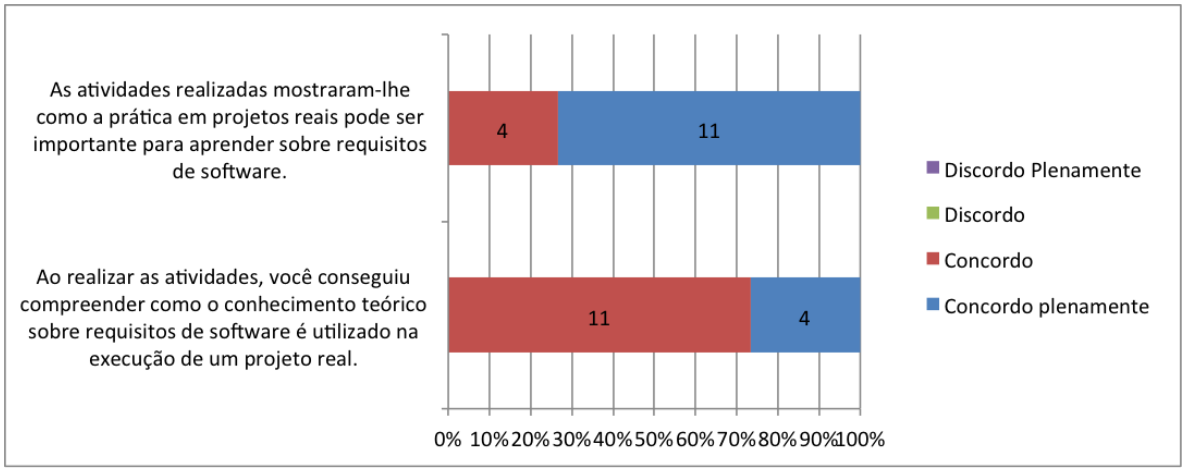

Figura 8.3 Compreensão dos estudantes sobre a conexão entre a teoria e a prática em projetos reais no estudo EC3.

As evidências mostram que a execução das atividades do projeto fez com que todos os estudantes compreendessem como o conhecimento teórico sobre requisitos de software é utilizado na execução de um projeto real. Todos também concordaram que essas atividades revelaram como a prática com projetos reais pode ser importante para a aprendizagem sobre requisitos de software, 73,33%, inclusive, concordaram plenamente com essa questão.

Sob a perspectiva qualitativa, identificamos que os estudantes reconheceram com a participação no projeto, como a prática é importante para a teoria, como a teoria é necessária para a prática, ou mesmo, como esses dois elementos são fundamentais para a aprendizagem. A Figura 8.4 detalha a percepção dos estudantes, sobre a qual, discorreremos a seguir.

Importância da prática para a teoria

Sobre a importância da prática para a teoria (ver Figura 8.4), capturamos que a prática: (i) ajuda a entender a teoria; (ii) concretiza a teoria; (iii) consolida o conteúdo; (iv) leva à participação ativa do estudante; (v) ajuda na retenção do conhecimento; (vi) permite aprender como fazer; (vii) permite enxergar o problema realmente; (viii) permite visualizar a aplicabilidade da teoria e, finalmente, (ix) possibilita a discussão. A seguir apresentaremos os extratos que levaram à essa categorização.

Vários estudantes relataram que fazer o projeto proporcionou um 'melhor entendimento sobre o conteúdo' repassado durante as aulas.

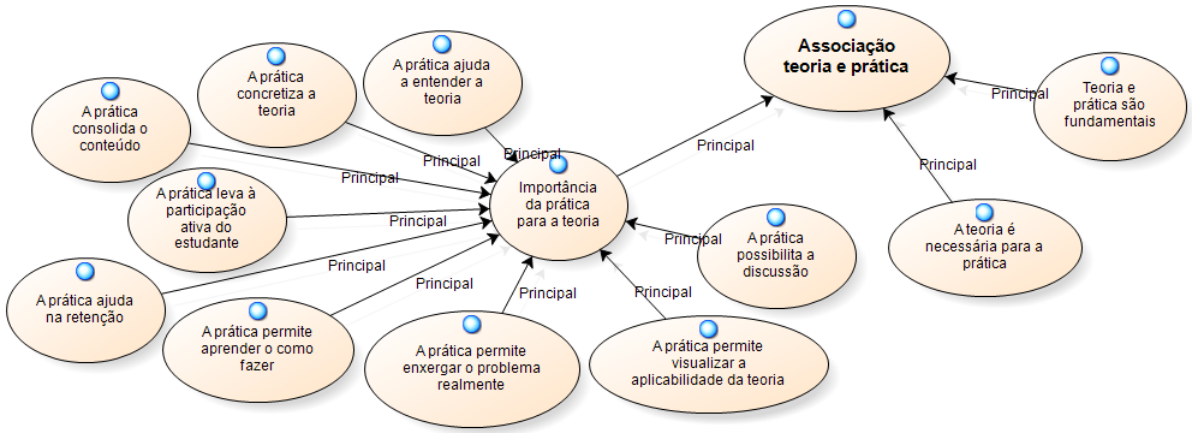

Figura 8.4 Percepção dos estudantes sobre a conexão teoria e prática no estudo EC3.

> *"Foi algo que melhorou muito meu entendimento e acho que dos meus amigos também, pelo menos no meu grupo, sim"* (SR2).

> *"(...) o JabRef... eu acho que deu pra gente aplicar bastante coisas da disciplina, que era o objetivo principal.... e destrinchar muitas coisas que às vezes a gente não tava conseguindo entender só no slide, na aula, e colocar em prática (...) como eu falei, tinham coisas que não estavam claras, coisas da disciplina mesmo que não estavam tão claras. E aí com o projeto, (...) buscando fazer... realizar o trabalho, deu para esclarecer"* (SR4).

> *"Pegar aquilo e ver realmente se... se realmente você entendeu aquilo... se você pegou os conceitos"* (SR8).

Ou mesmo, como no caso de outro estudante, verificar que seu entendimento inicial sobre o conteúdo não estava totalmente correto.

> *"(...) tinham vários assuntos na teoria que eu tinha um entendimento, quando ia pra prática e colocava aquele entendimento, a gente percebia que não era bem aquilo... Isso facilitou a entender a teoria"* (SR7).

Para alguns estudantes, a prática permite 'concretizar o que está sendo estudado na teoria', isto é, o conteúdo visto deixou de ser imaginação e tornou-se real.

> *"Eu achei bom porque... a gente conseguiu colocar realmente em prática o que a gente estava estudando..."* (SR4).

> *"Como eu já tinha dito, não ficou apenas na... o conhecimento não ficou apenas na teoria. Ele... vamos dizer que ele saiu da cabeça e aconteceu de fato"* (SR5).

> *"Porque o que foi feito (...) saiu do abstrato e foi pro real, entendeu?"* (SR2).

> *"(...) a gente ficou muito na teoria do conteúdo, porque DS 1 é muito teórico (diagrama é isso, estudo de caso [caso de uso] é isso, requisito é isso). E a gente conseguiu ver, identificar um requisito no próprio software, viu ele realmente implementado... e toda a sua... a rastreabilidade dele"* (SR1).

Este mesmo estudante ao ser questionado sobre o que achou interessante na execução das atividades, dentre outras coisas, destacou essa concretização dos requisitos no projeto.

> *"(...) você poder ter uma nova visão de DS I, de você não ficar só aprendendo a teoria e realmente poder ver na prática, os requisitos e sua implementação"* (SR1).

Outro estudante ao ser questionado sobre a contribuição do desenvolvimento das atividades para a sua aprendizagem, apontou a importância da conexão entre o conteúdo e a realidade proporcionada, para 'consolidar o conhecimento'.

> *"Foi, foi. Foi eu poder fazer... ligar, ligar os pontos... ligar os pontos e... preencher as lacunas, por assim dizer... consolidar o conhecimento (...) Eu acho que ficou mais... vamos dizer assim... concretizado. Eu pude realmente... consolidar esse tipo de informação"* (SR8).

Percepção compartilhada por outros estudantes, como pode ser constatado nos extratos a seguir.

> *"(...) a gente vê muito assunto, e entendemos, mas... com o projeto eu acho que consolida mais a questão do conteúdo"* (SR3).

> *"E achei interessante... não só pelo trabalho que teve, mas pelo resultado que ele proporcionou... que ele... gerou um conhecimento mais sólido do que... é... uma simples aula, assim, através de slides, numa sala de aula sentado, só recebendo informação (...) e essa experiência de trabalhar com o JabRef (...) criou um conhecimento... eu diria que solidificou mesmo o conhecimento (...) O fato de ter trabalhado com ele, acredito que solidificou mesmo, o conhecimento que estava, digamos assim, acumulado só na teoria"* (SR5).

No início deste último extrato, o estudante indicou outro fator importante à consolidação do conhecimento que foi a participação ativa do estudante. O estudante ainda complementou que 'a prática leva a participação ativa', consequentemente à maior retenção do conhecimento.

> *"(...) quando se vai para a sala de aula ficar olhando... apenas slides, ficar recebendo só a informação... e você não ter nenhuma atividade que consolide aquilo que você está recebendo, aquilo que você está vendo, com... eu acredito que com certeza... aquele conhecimento vai ficar vago em determinado tempo. Vai chegar uma época que você não vai mais lembrar... o que você viu, o que você escutou, se você não consolidar através de uma prática... você tirar... da teoria e botar na prática aquilo que você tá vendo, tá ouvindo, que você tá aprendendo, e acho que isso contribuiu bastante no conhecimento"* (SR5).

Mais um estudante realçou a questão da participação ativa para a sua aprendizagem, inclusive salientou a necessidade de buscar mais informação para atender ao que foi solicitado e demonstrou insatisfação com o modelo de ensino tradicional utilizado em sala de aula.

> *"Porque era muito assunto pra decorar e fazer a prova, praticamente foi só decoreba. E isso aí eu botei em prática o que eu aprendi, mesmo estudando em casa aprendi, fui atrás e consegui aprender muito mais do que na aula em si. (...) Foi muito importante esse projeto porque a gente podia fazer mesmo com nossas próprias mãos o que a gente aprendeu na sala ou com terceiros, entendeu? É muito melhor você aprender fazendo do que só ouvindo (...) Porque uma coisa é você passar slides e ficar lá falando, falando, falando... Porque tem horas que você tá lá e tem horas que a gente viaja... comigo acontece muito isso. Mas nem é nossa culpa nem a culpa é do professor. É porque o modelo aplicado na sala é totalmente diferente de você fazer. Você aprende muito mais fazendo do que você vendo"* (SR2).

Finalmente, outro estudante destacou o esforço exigido, dado a participação ativa, porém reconheceu a relevância desse esforço para a aprendizagem.

> *"(...) dá muito trabalho, mas... eu acho que é esse trabalho que faz com que o aluno aprenda mais"* (SR3).

Esses mesmos estudantes relataram a importância da participação ativa por meio da prática no projeto para a retenção do conhecimento. Como veremos nos extratos a seguir, enquanto no primeiro, o estudante ressalta a 'importância da prática para a retenção', comparando-a com o método tradicional de ensino. No segundo, o estudante destaca a fixação do conteúdo e a satisfação com a abordagem, apesar do esforço despendido. Por fim no terceiro extrato, o estudante aponta o processo cognitivo interno que ocorre com a prática, de construção do conhecimento.

> *"É como eu falei, aprendi muito mais do que na própria sala de aula... do que em prova... do que em... porque em prova praticamente foi decoreba, né? Porque era muito assunto pra decorar e fazer a prova, praticamente foi só decoreba (...) [comentando sobre o projeto] Porque eu tive que fazer mesmo, não recebi pronto (...) É uma coisa que não vou esquecer tão fácil"* (SR2).

> *"Foi aquela questão da prática, né. Às vezes a gente vê a teoria, mas quando a gente vê a prática fixa mais o conceito (...) Dá trabalho mas... dá muito trabalho, mas quando se vê o resultado e como fixa na cabeça aquilo, é bastante gratificante"* (SR3).

> *"E com a prática, ele... é como se ele criasse uma caixinha e fixasse no conhecimento da pessoa, no cérebro, que... torna aquilo ali já... como... uma coisa já conhecida, não é mais nada novo pra você, e... no conhecimento da disciplina... enriqueceu bastante"* (SR5).

Dois estudantes ainda revelaram o valor da prática para 'aprender o como fazer'.

> *"Pelo menos eu não tinha essa habilidade. (...) a gente teve que correr atrás de coisas que a gente não sabia e botar em prática no projeto"* (SR2).

> *"Então a prática ajudou muito a entender como é que faz o diagrama e tal"* (SR7).

A prática é que desenvolve a habilidade do saber fazer. Porque é na tentativa de fazer que surgem as dúvidas, que 'enxergamos realmente os problemas envolvidos', particularmente quando estamos trabalhando com projetos reais e não simplificações, ou seja, a prática com projetos reais permite visualizar que nem sempre, tudo é tão simples, como muitas vezes na teoria apresenta-se.

> *"(...) a gente vê como é complicado, aquele negócio de levantamento de requisitos... saber os casos de uso e tudo... saber a visão um pouco do que o cara que desenvolveu teve que saber... É bastante legal, porque às vezes a gente vê a teoria, mas quando vê a prática, é quando vê que o negócio é mais difícil do que o que diz na teoria (...) é uma coisa que a gente pega o conteúdo e acha que sabe, mas quando vai ver na prática, aí surge a dúvida... aí ajudou muito"* (SR3)

> *"Eu achei que seria mais fácil... colocar em prática..."* (SR6).

Por outro lado, a prática também 'permite visualizar a aplicabilidade da teoria'. Quando questionado sobre o que achou interessante na realização das atividades do projeto, um estudante descreveu exatamente o casamento do que tinha sido estudado com a prática.

> *"Primeiro eu acho que foi a questão de colocar em prática aquele conhecimento... de sala de aula (...) se os conceitos estão... realmente casando com... com a prática, com o que... com o trabalho feito"* (SR8).

Pensamento corroborado por mais um estudante, ao afirmar que o conteúdo visto foi aplicado realmente no projeto.

> *"Contribuiu porque a gente foi pra prática realmente. A gente partiu para a prática e percebeu que realmente são aplicadas... todo o conceito, toda a teoria que foi vista em sala de aula"* (SR1).

Ademais, a prática com um projeto real cria uma experiência comum, o que 'possibilita a discussão' sobre possíveis soluções. No extrato abaixo, o estudante descreveu esta situação, onde por meio da discussão criada em sala de aula, o mesmo pode comparar a sua solução com a de seus colegas.

> *"E também nós podemos ver se o que nós fizemos está correto, porque às vezes é o que acontece, nós fazemos o trabalho, tá feito, mas não temos certeza se está correto. E com a discussão, vendo as opiniões de outras pessoas, também é interessante porque às vezes eles levam por pontos que nem nós percebemos, e que deveriam estar lá... Aí, eu achei bastante interessante"* (SR3).

Como o projeto não é simplificado, ele pode inclusive servir de exemplo para outras discussões, conforme reportado por outro estudante .

> *"E também com as discussões com o outro professor... isso ajudou legal"* (SR7).

A teoria é necessária para a prática

Todavia o conhecimento teórico prévio, antes da prática com projetos reais, também é necessário. No extrato a seguir, o estudante reportou que os diagramas foram elaborados para o projeto, seguindo o que foi feito em sala de aula.

> *"(...) foram elaborados os diagramas e os casos de uso, como foi feito em sala de aula, aí foi bem detalhado de acordo com nosso aprendizado que tinha tido na disciplina"* (SR1).

Inclusive, quando esse conhecimento prévio não é suficiente, ele precisa ser complementado, tal como foi relatado por outros dois estudantes. Ratificando a 'necessidade do conhecimento teórico para a realização da prática'.

> *"Coisas que eu vi na sala... poucas, mas eu consegui colocar no projeto. Muitas outras eu tive que pegar de terceiros, em sites e artigos, para poder fazer no projeto. Tive que pegar uma complementação sobre isso, para poder implementar no projeto"* (SR2) .

> *"E aí com o projeto, a gente colocando em prática, pesquisando mais e buscando fazer"* (SR4).

Teoria e prática são fundamentais

Neste estudo, detectamos a preferência de alguns estudantes pelas atividades práticas, levando em conta a contribuição para o seu aprendizado.

> *"Para falar a verdade, eu aprendi mais praticando no projeto do que propriamente nas aulas (...) porque você vai aprender... o aluno vai aprender muito mais praticando e fazendo, mesmo que quebre a cabeça, tenha dúvida, reclame, ele vai aprender muito mais, do que aprender só em teoria"* (SR2).

> *"(...) a parte prática, com certeza, contribuiu muito... para o aprendizado. Claro que a teoria também... mas acredito que a prática... foi o que contou mais"* (SR5).

Contudo, neste último extrato, apesar da preferência pela prática, o estudante reconhece a importância da teoria. Ponto de vista reforçado por mais estudante, ao afirmar que a prática é um complemento e que no final, os dois elementos são fundamentais para a aprendizagem.

> *"Ainda é um complemento. Os dois são fundamentais e trazem mais aprendizado pro aluno"* (SR3).

8.2.3 Objetivos de Aprendizagem Alcançados (Ob4)

Após discorrermos sobre como os estudantes enxergaram a abordagem de utilização de um projeto de código aberto como uma experiência de mundo real e as consequências para a aprendizagem, dada a associação da teoria com a prática em um projeto real, cabe investigarmos os resultados diretos para a aprendizagem, isto é, quais objetivos de aprendizagem podem ser alcançados pela abordagem. Como o foco desse estudo de caso foi aplicar a abordagem de utilização de OSP para o ensino de requisitos de software, selecionamos os objetivos de aprendizagem relacionados a essa área de conhecimento estabelecidos no Capítulo 3. Analisamos quais seriam mais adequados a serem tratados pelo estudo de caso, uma vez que estes precisavam estar também de acordo com o programa e planejamento do professor da disciplina, e ainda, com as limitações impostas pelo projeto de código aberto utilizado, no caso, o JabRef, como mencionado em 8.1.3. Além dos objetivos de aprendizagem específicos relacionados à área de conhecimento citada, selecionamos também objetivos ligados ao desenvolvimento de competências associadas à prática profissional que poderiam ser beneficiadas pela aplicação da abordagem. Consequentemente, dividimos a apresentação dos resultados em aprendizagem de aspectos técnicos e desenvolvimento de competências da prática profissional.

Aprendizagem de aspectos técnicos)
Investigamos a aprendizagem de aspectos técnicos a partir de quatro perspectivas: (i) comparação, antes e depois da aplicação da abordagem, da percepção dos estudantes sobre a sua capacidade com relação aos objetivos de aprendizagem selecionados para o estudo; (ii) percepção dos estudantes sobre a contribuição do projeto para o alcance desses objetivos de aprendizagem; (iii) análise dos documentos entregues pelos estudantes e (iv) relatos dos estudantes ao longo das entrevistas.

Os objetivos de aprendizagem associados a requisitos de software que perscrutamos encontram-se enumerados na Tabela 8.5. A tabela também apresenta a comparação

entre a percepção dos estudantes sobre a sua capacidade considerando estes objetivos de aprendizagem após o emprego do método tradicional de ensino, com aulas expositivas, exercícios e prova, com a sua percepção após a execução das atividades práticas com o emprego do OSP. Confrontando os resultados, verificamos mudanças de moda para quatro objetivos de aprendizagem, sendo que três positivas e uma negativa, ou seja, para três objetivos de aprendizagem ocorreu uma melhoria em relação à concordância e para um objetivo aconteceu uma piora para o nível de concordância. Simultaneamente detectamos diferenças positivas significativas para três objetivos[6].

Por outro lado, para todos os sete objetivos de aprendizagem investigados diretamente, a maioria dos estudantes concordou que o projeto contribuiu para a melhora de suas capacidades (ver Figura 8.5).

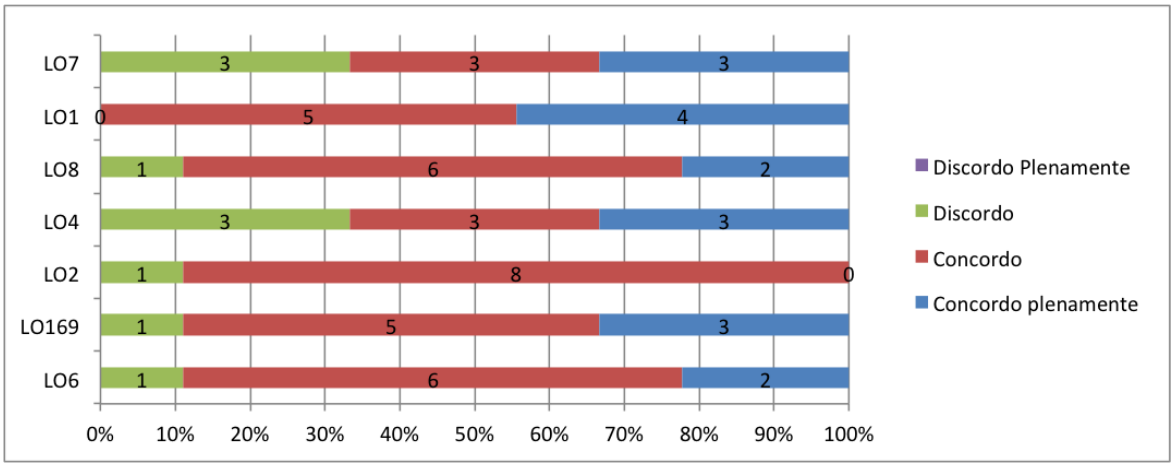

Figura 8.5 Contribuição da abordagem para a aprendizagem sobre requisitos de software.

Analisando estes resultados de maneira mais detalhada, verificamos que obtivemos melhoras significativas, conforme a Tabela 8.5, para os objetivos LO2 (capacidade de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais), LO8 (capacidade de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos) e LO7 (capacidade de diferenciar o rastreamento de 'construção' e 'recuperação', e explicar seus papeis no processo de validação dos requisitos). Como ao definirmos as atividades que seriam executadas, procuramos idealiza-las de modo que os estudantes visualizassem como ocorre o processo de engenharia de requisitos e gerenciamento de requisitos tendo como base um projeto real, particularmente um OSP, onde tais processos ficam de algum modo registrados, bem como, a realização de fato do rastreamento dos requisitos desse software. Supomos que o exemplo concreto empregado seja o responsável pela melhoria obtida nessa aprendizagem. Todavia, os 33,33% de discordâncias para o objetivo de aprendizagem LO7 (Figura 8.5) denotam que melhorias ainda podem ser sugeridas, principalmente gerando exemplos mais concretos de rastreamento de construção e de recuperação. Situação similar

[6]Dado que os dados não apresentavam uma distribuição normal, aplicamos o teste de postos sinalizados de Wilcoxon para amostras pareadas.

Tabela 8.5 Comparação entre a aprendizagem após o método tradicional de ensino e após o emprego da abordagem com OSP (EC3).

Id.	**Objetivos de aprendizagem específicos sobre requisitos de software**	**Moda**		**Soma de postos** positivos	***Valor-p***
		Trad.	OSP		
LO6	Capacidade de modelar e especificar os requisitos de software de médio porte, usando métodos comuns (ou seja, não formal/matemático).	-1 e 1	1	10,5	0,198
LO169	Capacidade de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte.	1	1	8,5	0,097
LO2	Capacidade de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.	-1	1	10	0,023*
LO4	Capacidade de comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.	1	1	6	0,353
LO8	Capacidade de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos.	-1 e 1	1	10	0,030*
LO1	Capacidade de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.	2	1	12	0,370
LO7	Capacidade de diferenciar o rastreamento de 'construção' (realizado a partir dos requisitos) e 'recuperação' (realizado a partir do código) e explicar seus papeis no processo de validação dos requisitos.	-1	-1	10	0,023*

(*) Representa resultado significativo, para nível de significância de 0,05, unidirecional.

será necessária para o alcance do objetivo LO4 (capacidade de comparar as abordagens direcionada por planos e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma), onde, apesar de termos utilizado estratégia semelhante de comparação com o processo que ocorre em projetos de código aberto e o tópico ter sido bastante discutido em sala de aula (da mesma forma que o LO2 e LO8), não conseguimos obter uma melhora significativa e 33,33% dos estudantes discordaram da contribuição do projeto para o alcance desse objetivo de aprendizagem (ver Figura 8.5). Analisando as respostas apresentadas nos documentos entregues, em geral, os estudantes apresentaram as características dos modelos, contudo não discutiram os riscos

e os benefícios de cada um. Apenas a equipe ER4 fez uma comparação no que diz respeito à volatilidade dos requisitos, validação e proximidade do usuário. Nossa suposição é que talvez a avalição crítica dos riscos e benefícios de cada modelo seja prematura para estudantes ainda imaturos profissionalmente. No caso da equipe ER4, dois membros possuíam previamente alguma experiência com projetos reais.

Quanto à capacidade de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software (LO1), ao mesmo tempo que obtivemos uma piora para a moda (Tabela 8.5), todos os estudantes concordaram que o projeto contribuiu para a melhoria dessa competência (Figura 8.5). Sob a perspectiva dos relatos dos estudantes ao longo das entrevistas, capturamos que alguns estudantes apontaram de fato dificuldades nessa identificação, como exemplificado no extrato a seguir.

> *"Trabalhar com requisitos, no caso, funcional e não-funcional (...) eu me sinto muito segura sobre os requisitos, no caso funcionais, e os não funcionais, eu tenho algumas dúvidas"* (SR6).

Entretanto, outro estudante, mesmo tendo dúvidas, reconheceu que a atividade contribuiu para a aprendizagem do conceito.

> *"Porque quando eu... eu fui definir o que era funcional e não funcional, ali sim, ali sim, eu vi, eu tive... o conceito pegou mais quando eu tive que sair olhando cada requisito 'isso é funcional', 'isso não é funcional', 'eita, esse é o quê? Esse eu tô em dúvida, é funcional ou não funcional?', aconteceu muito isso, fiquei em dúvida"* (SR3).

Por fim, um outro estudante explicou a razão das dúvidas que surgiram em relação à identificação dos requisitos não funcionais. O requisito não funcional terminava tornando-se funcional, porque muitas vezes representava a implementação de um comportamento diferente para a funcionalidade. Isto porque, as regras de funcionamento do JabRef são configuradas de acordo com as opções escolhidas pelos usuários do software.

> *"Na segunda parte já foi bem mais difícil. Primeiro, esse quesito de requisito funcional e não funcional, aquele não funcional mas é funcional, ele acaba sendo implementado, tava gerando muita dúvida disso, foi muito debatido"* (SR1).

Por conseguinte, apesar dos exemplos concretos proporcionados, o projeto de código aberto escolhido para a realização dessa atividade não foi o mais adequado, dada sua característica de autoconfiguração que terminou por confundir os estudantes. No entanto, convém salientar que, analisando os trabalhos entregues, a atividade relacionada gerou resultados satisfatórios.

Por último, os objetivos LO6 (capacidade de modelar e especificar os requisitos de software de médio porte) e LO169 (capacidade de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte) estão ligados diretamente à competência do 'saber fazer' da área de requisitos de software. Mesmo obtendo uma ligeira melhora

da moda para o objetivo LO6 (Tabela 8.5), a melhora após o emprego da abordagem de uso de OSP não foi significativa para nenhum dos dois objetivos de aprendizagem. Avaliando os modelos entregues pelas equipes, concluímos que não estavam de fato satisfatórios (LO6). Apresentavam muitos problemas, inclusive na utilização da notação UML (LO169). Em seus relatos, os estudantes apontaram dificuldades na elaboração desses modelos, em especial o modelo conceitual.

> *"Eu achei que seria mais fácil (...) encontrar alguns diagramas..."* (SR6).

> *"(...) até eu estava em dúvida sobre o diagrama conceitual... eu me atrapalho muito com o DER"* (SR1).

> *"(...) a gente teve as aulas sobre os diagramas e a gente colocou esses diagramas em prática... Na primeira parte do trabalho, a gente teve muita dificuldade, a questão do diagrama conceitual... Mas depois quando a senhora passou aquela discussão e tal, isso ficou bem claro (...)"* (SR7).

Embora o último estudante tenha comentado que colocaram em prática os diagramas em sala de aula, outro estudante indicou a falta de prática anterior como principal problema para a elaboração dos diagramas.

> *"Teve um diagrama... os diagramas foram o principal... o principal... problema. Porque a gente não praticava, a gente só via a teoria e não praticava. Porque o principal é a prática, né? Porque vê teoria é diferente de praticar, basicamente isso..."* (SR2).

Sondando os exercícios repassados pelo professor da disciplina, concluímos que exemplos foram repassados, contudo, talvez a quantidade não tenha sido suficiente. Sendo assim, a dificuldade de elaboração de diagramas não foi proveniente exclusivamente da utilização de um projeto real, mas também da falta de mais exercícios anteriores.

Em contrapartida, os estudantes, em sua grande maioria, concordaram com a contribuição do projeto para a melhoria de suas capacidades de modelagem (LO6) e utilização da notação UML (LO169) (ver Figura 8.5), situação confirmada ao longo das entrevistas. Dos oito estudantes entrevistados, seis comentaram sobre a contribuição do projeto para aprender a elaborar os diagramas (LO6) e um, sobre o uso da notação UML (LO169). Os extratos seguintes ilustram este fato.

> *"(...) a gente aprendeu muito a fazer os diagramas"* (SR1).

> *"(...) fazer alguns diagramas... acho que... contribuiu um bocado"* (SR6).

> *"Porque eu não tinha a menor ideia de como fazer aquele diagrama de sequência... não tinha a menor ideia. E com esse projeto, consegui saber que era uma besteira, consegui fazer muito mais fácil. Simples, uma coisa tão simples e eu tendo tanta dificuldade"* (SR2).

> *"(...) eu tinha pouquíssima experiência com UML, eu tinha mexido um pouco com DFD... e eu tinha pouquíssima experiência com UML... e essa questão dessa documentação, assim... mais aprofundada..."* (SR8).

> *"É... a aplicação até tinha coisas que eu já sabia... a questão dos diagramas, (...) e aí pude praticar e até aperfeiçoar mais"* (SR4).

Uma possível explicação para que não tivéssemos obtido melhoras tão significativas para os objetivos de aprendizagem de forma geral, foi a ausência de avaliações individualizadas. Como as atividades foram propostas para as equipes, na maioria das vezes, as mesmas foram subdivididas entre seus membros, de maneira que, cada membro ficou responsável por executar individualmente a atividade. Conforme constatado através das entrevistas, somente a equipe ER2 realizou as atividades e discutiu as questões em grupo.

Além da habilidade relacionada à construção de diagramas, capturamos o desenvolvimento de outras habilidades por parte dos estudantes: (i) saber trabalhar com requisitos; (ii) realizar engenharia reversa; (iii) identificar casos de uso; (iv) modularizar o software de acordo com os requisitos, e (v) fazer a rastreabilidade de requisitos. A seguir apresentaremos as respectivas evidências.

Com o projeto, os estudantes admitiram que aprenderam a trabalhar com requisitos, como buscá-los, reconhecê-los e documentá-los.

> *"E eu aprendi também a questão dos requisitos, como trabalhar com requisitos... isso me ajudou bastante..."* (SR7).

> *"(...) isso de, de ir buscar os requisitos, já vou ter uma facilidade maior"* (SR6).

> *"(...) eu acho que eu tô conseguindo reconhecer melhor os requisitos... e documentar em si..."* (SR8).

Sem se dar conta, no extrato a seguir, o estudante descreveu a realização de uma engenharia reversa, reconhecendo o aprendizado logrado.

> *"(...) porque todos os projetos que eu já peguei, eu já tinha documentação pronta. Eu só fui lá e programei, não tive esse trabalho de pegar o programa pronto e transformar ele em uma documentação. Foi praticamente, algo novo, que me fez aprender muito mais"* (SR2).

Enquanto que outro estudante admitiu que aprendera a fazer a engenharia reversa, ao mesmo tempo que reconheceu a importância desse aprendizado para a realização de futuras manutenções.

> *"E em si essa engenharia reversa que fomos vistos [que vimos] (...) a gente aprende esse método que com certeza vai ser o mais utilizado, (...) A gente tinha ideia de como recolher os requisitos, mas a gente não tinha ideia de como com os requisitos já elaborados, como fazer a busca deles no código (...) porque é preciso saber identificar os requisitos e como eles estão implementados para poder dar manutenção neles"* (SR1).

Baseando-nos nesses relatos, podemos dizer que indiretamente alcançamos o objetivo de aprendizagem LO124 (aplicar engenharia reversa em um software de médio porte).

Ademais, no extrato abaixo, o estudante expõem que o projeto ajudou-lhe a identificar efetivamente os casos de uso do software.

> *"(...) a gente saber o que é caso de uso, o que não é... assim, a gente via lá, talvez abrir o banco de dados, a gente não pensaria que fosse um caso de uso apenas... uma entrada, uma nova entrada ou uma nova referência, editar, então, e aí a gente acabou tendo... esse... esse encaixe, tanto da teoria quanto na hora da prática"* (SR4).

Já outro estudante acrescentou à identificação dos casos de uso e à construção de diagramas, o panorama sobre a engenharia de requisitos e, ainda mais importante, a visão sobre a modularização do software de acordo com os requisitos. Demonstrando assim, uma introdução ao desenvolvimento da capacidade de pensar o problema em vários níveis de detalhe e abstração (LO273). Convém salientar que tal capacidade não está ligada unicamente à área de requisitos de software, mas representa o desenvolvimento de uma competência da prática profissional.

> *"Eu aprendi mais sobre... sobre a questão da engenharia de requisitos, que é fundamental. E de como é que eles... podem separar o projeto para ser desenvolvido... separar por partes, os tópicos que são fundamentais para enfim desenvolver... é selecionar os casos de uso... fazer os diagramas para poder ter uma visão geral"* (SR3).

Finalmente, no próximo extrato, o estudante destacou a capacidade de rastrear os requisitos na perspectiva do código. Desse modo, rastreabilidade deixou de ser um conceito apresentado em sala de aula, para se tornar algo compreendido e executado na prática.

> *"(...) aprendi a rastrear os requisitos desde as funcionalidades do aplicativo até sua implementação, até achar os métodos dele, as classes de interface"* (SR1).

Além do desenvolvimento de habilidades, a abordagem de uso do OSP possibilitou a conscientização sobre a importância da documentação para a manutenção de software (objetivo de aprendizagem LO114), conforme retratado a seguir. Inclusive, o estudante salienta esta relevância até mesmo para o profissional que desenvolveu o software.

> *"(...) não ter documentação, também é ruim, porque... mais pra frente, até você mesmo que for tentar modificar a ferramenta, você não vai nem lembrar, porque você já produziu várias, várias ferramentas... e um... um software complexo, pronto! (...)". "A importância também da documentação... a gente sabe que é importante, mas quando vê lá aquele negócio que a pessoa tem que alterar ou então que outra pessoa tem que modificar o programa, aí que a pessoa se toca, que realmente a documentação nesse caso é boa. Porque a gente nunca imagina que o negócio não vai dar errado, sempre vai dar certo"* (SR3).

Como esperado, ocorreu também algum aprendizado sobre projetos de código aberto. Os extratos revelaram que essa aprendizagem foi considerada bastante interessante por parte dos estudantes, gerando até mesmo o interesse deles pelo tema.

Questionados sobre como foi a experiência de aprendizagem realizando as atividades do trabalho, dois estudantes destacaram que o ponto de partida foi a aprendizagem sobre projetos de código aberto. Foi algo que lhes chamou a atenção por representar algo novo.

> *"Então, a aprendizagem surge desde o começo, quando eu fui apresentado a uma ferramenta open source, que eu não tinha a mínima ideia do que era"* (SR1).

> *"A questão... só o fato de... de ter um projeto... dele ser open source... ele já proporcionou algo novo do que a gente viu até, até aquele momento"* (SR5).

Outros dois estudantes corroboraram a novidade do aprendizado sobre projetos de código aberto.

> *"(...) até o entendimento assim do que é realmente um projeto open source, que eu não tinha nem noção da... até da colaboração de outras pessoas (...)"* (SR4).

> *"Eu aprendi muito sobre projetos open source, eu não tinha muito conhecimento sobre isso"* (SR7).

Dentre esses quatro estudantes, o primeiro ainda complementou como uma das coisas mais interessantes dentre tudo que foi feito.

> *"O mais interessante... o aprendizado com open source foi um"* (SR1).

Por último, um estudante que não foi entrevistado, ao responder a questão aberta do questionário, onde solicitamos que expressassem críticas e sugestões de melhorias, espontaneamente, demonstrando certa satisfação, destacou a utilização de um projeto real e o aprendizado sobre projetos de código aberto como pontos que mais lhe chamaram a atenção.

> *"O que mais chamou minha atenção no projeto foi o contato com um projeto real, e conhecer como funciona o gerenciamento de requisitos para projetos open source. Eu fiquei realmente curioso em como seria ter a experiência de participar no desenvolvimento de algum projeto desses"* (SR13).

Após a discussão sobre todos esses tópicos, para o julgamento final se os objetivos de aprendizagem relacionados aos aspectos técnicos foram ou não alcançados, aplicamos as heurísticas e o algoritmo definidos em 4.5,. A Tabela 8.6 apresenta o julgamento para cada um dos objetivos. As dimensões empregadas para avaliação da aprendizagem aparecem em cada coluna. Portanto, a coluna 'melhora significativa' retrata o resultado positivo ou não do teste estatístico aplicado (ver Tabela 8.5). A coluna 'análise dos

trabalhos' expõe alguns comentários que sumarizam pontos discutidos ao longo do texto sobre os trabalhos apresentados. A coluna 'concordância com a contribuição' denota o percentual do nível de concordância dos estudantes se a abordagem contribuiu para o alcance do objetivo de aprendizagem (representa a soma dos percentuais de 'concordo' e 'concordo plenamente' na Figura 8.5). A coluna 'relato dos estudantes' apenas indica se ocorreram um ou mais relatos dos estudantes sobre o alcance do objetivo em questão. Neste caso, acrescentamos ainda comentários quando algum relato negativo ocorreu. Cabe destacar que utilizamos um hífen ('-') para expressar quando não analisamos qualquer uma dessas dimensões para o objetivo sendo tratado. Finalmente, a coluna 'alcance do objetivo' aponta o julgamento final para cada objetivo de aprendizagem, após a aplicação das heurísticas citadas.

Tabela 8.6: Julgamento do alcance dos objetivos de aprendizagem associados aos aspectos técnicos para o estudo de caso sobre requisitos de software (EC3).

Id.	**Objetivos de aprendizagem**	**Melhora Significativa**	**Análise dos trabalhos**	**Concordância com a contribuição**	**Relato dos estudantes**	**Alcance do objetivo**
LO6	Capacidade de modelar e especificar os requisitos de software de médio porte, usando métodos comuns (ou seja, não formal/matemático).	Não	Não estavam satisfatórios	88,9%	Sim	Parcial
LO169	Capacidade de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte.	Não	Não estavam satisfatórios	88,9%	Sim	Parcial
LO2	Capacidade de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.	Sim	-	88,9%	Sim	Sim
LO4	Capacidade de comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.	Não	Apenas uma equipe discutiu alguns benefícios, os demais a descreveram apenas os modelos	66,7	-	Parcial

LO8	Capacidade de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos.	Sim	-	88,9%	-	Sim
LO1	Capacidade de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.	Não	Trabalhos satis-fatórios	100,0%	Relataram o apren-dizado, mas que tiveram dúvidas	Sim
LO7	Capacidade de diferenciar o rastreamento de 'construção' e 'recuperação' e explicar seus papeis no processo de validação dos requisitos.	Sim	Trabalhos satis-fatórios	66,7%	-	Sim
LO124	Aplicar engenharia reversa em um software de médio porte	-	-	-	Sim	Sim
LO114	Perceber a importância da documentação para a manutenção de software	-	-	-	Sim	Sim

Desenvolvimento de Competências da Prática Profissional

Competências diretamente investigadas

Como mencionamos anteriormente, além das habilidades técnicas, também perscrutamos a contribuição do projeto para o alcance de alguns objetivos de aprendizagem relacionados à competências da prática profissional, identificados no Capítulo 3. A Tabela 8.7 enumera os objetivos explicitamente investigados.

Tabela 8.7 Objetivos de aprendizagem explicitamente investigados no estudo EC3.

Id.	**Descrição**
LO233	Aceitar e responder críticas de terceiros
LO235	Avaliar o trabalho desenvolvido por terceiros
LO268	Ter experiência com código desenvolvido por terceiros.
LO236	Habilidade de ser proativo
LO237	Habilidade de ser criativo.
LO266	Ser capaz de solucionar problemas.
LO267	Gerar soluções alternativas para os problemas.
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).
LO251b	Desenvolver a habilidade de sumarizar material técnico (código fonte e documentação).
LO246	Escrever documentação, clara, concisa e acurada

A Figura 8.6 expressa a percepção dos estudantes sobre a contribuição da aborda-

gem para o desenvolvimento das competências examinadas. Tendo um visão global, a maioria concordou com o desenvolvimento das habilidades listadas, com destaque para LO268, LO233, LO235 e LO272. Todavia, obtivemos uma quantidade significativa de discordâncias para os objetivos LO237, LO266 e LO251b. A seguir discutiremos sobre essas situações.

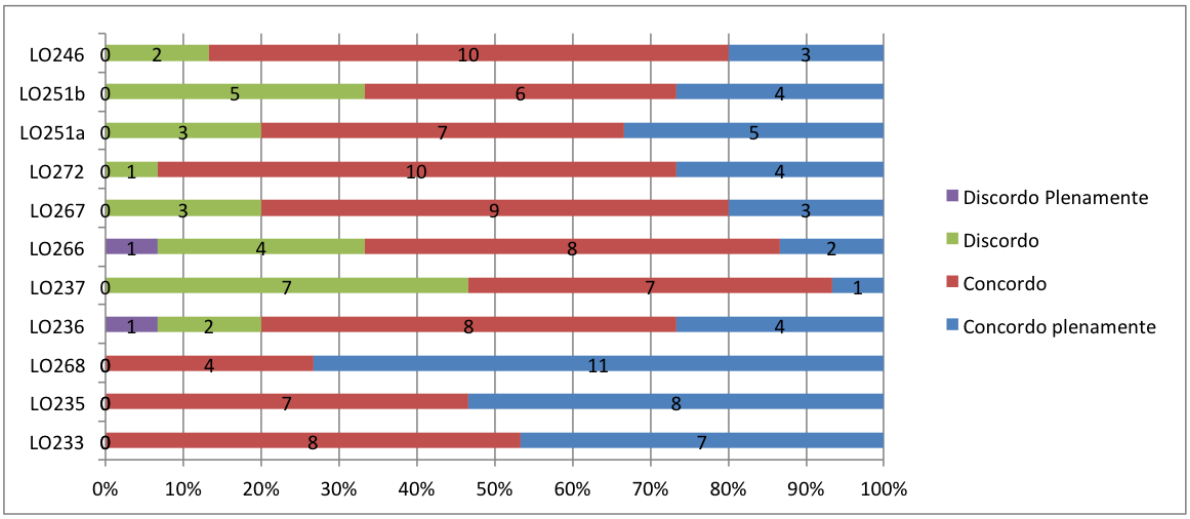

Figura 8.6 Contribuição do projeto para o alcance de objetivos de aprendizagem relacionados à prática profissional de engenharia de software no estudo EC3.

Como podemos observar na Figura 8.6, 100% dos estudantes concordaram com a contribuição das atividades para que os mesmos tivessem uma experiência com código desenvolvido por terceiros (LO268), assim como, devolvessem as habilidades de avaliar o trabalho desenvolvido por terceiros (LO235) e de aceitar e responder críticas de terceiros (LO233). Inclusive com muitos deles concordando plenamente com essa contribuição. Por conseguinte, a abordagem de uso de OSP aparece como propícia ao desenvolvimento dos objetivos LO268 e LO235, uma vez que é muito pouco provável que o código do projeto de código aberto selecionado tenha sido desenvolvido pelos próprios estudantes. Ao mesmo tempo, parece favorecer o aprimoramento da habilidade de aceitar e responder críticas de terceiros (LO233), importante, para quem futuramente precisará trabalhar como membro de uma equipe.

O maior número de discordâncias no que diz respeito à contribuição do projeto para o desenvolvimento das capacidades de solucionar problemas (LO266), gerar soluções alternativas para os problemas (LO267), ser proativo (LO236) e ser criativo (LO237) retrata que alguns estudantes não enxergaram as atividades propostas como algo que exigisse a solução de problemas e ainda que os mesmos precisassem ser proativos ou exercitar sua criatividade. Segundo a nossa percepção, ao analisarmos os trabalhos entregues e as dúvidas enviadas durante a sua execução, os estudantes realmente não desenvolveram estas habilidades no potencial que esperávamos.

No que concerne às habilidades ligadas à documentação escrita, 80% dos estudantes consideraram que o projeto contribuiu para que desenvolvessem a habilidade de com-

preender material técnico (código fonte e documentação) (LO251a), 66,67%, para que fossem capazes de sumarizar material técnico (código fonte e documentação) (LO251b) e, finalmente, 86,67%, para que fossem capazes de escrever documentação, clara, concisa e acurada (LO246). Explicações possíveis para as discordâncias ao objetivo LO251a seriam os estudantes considerarem que o software não estava documentado ou porque somente uma questão exigia realmente que eles lidassem com o código . Quanto ao maior número de discordâncias para o objetivo LO251b, interpretamos que os estudantes consideraram que não precisaram modificar o código fonte, ou mesmo que reconheceram as deficiências da documentação que eles próprios elaboraram. Os dois estudantes que discordaram quanto ao aperfeiçoamento da capacidade de escrever documentação, clara, concisa e acurada (LO246) também discordaram do aprimoramento em sua capacidade de sumarizar material técnico (LO251b), demonstrando consistência em suas percepções. Finalmente, analisando os trabalhos entregues, identificamos problemas na redação das respostas para algumas equipes, como erros de concordância e pontuação.

Por último, 93,33% dos estudantes concordaram que a execução das atividades do projeto propiciou uma visão do processo como um todo do desenvolvimento de software, isto é, uma visão holística do desenvolvimento de software (LO272).

Competências detectadas a partir da análise qualitativa

Analisando qualitativamente as respostas das entrevistas, percebemos que outros objetivos de aprendizagem foram também cobertos por meio da abordagem. A Tabela 8.8 lista esses objetivos. A seguir discorreremos sobre a conjuntura de desenvolvimento dessas competências.

Tabela 8.8 Outras competências também desenvolvidas pela abordagem no estudo EC3.

Id.	Descrição
LO227	Habilidade de trabalhar efetivamente como membro de equipes.
LO270	Compreender como teoria e prática influenciam uma a outra.
LO232	Expressar opiniões próprias.
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.
LO203	Compreender as expectativas de trabalho (habilidades esperadas).
LO269	Ter vivenciado pelo menos um projeto real.

Habilidade de trabalhar em equipe

Dois estudantes ao serem questionados sobre o que tinham aprendido realizando o projeto, dentre outras coisas, destacaram a habilidade de trabalhar em equipe.

> *"O que eu aprendi... trabalhar em grupo. Antes era dividir... dividir pra conquistar, agora foi reunir para conquistar"* (SR2).

> *"Primeiro... não que aprendi, mas vamos dizer que... reviver o trabalho em conjunto com a equipe"* (SR5).

Já outro estudante quando questionado sobre como foi sua experiência de aprendizagem realizando o trabalho também destacou o trabalho em equipe.

"Eu achei boa, tanto o trabalhar em equipe (...)" (SR4).

Entretanto, tais estudantes eram membros de uma mesma equipe, a ER1. A explicação para que eles realçassem esse tipo de aprendizagem está na forma que a equipe trabalhou para enfrentar o desafio que lhes foi apresentado, descrita por um de seus membros.

> *"Com relação a esse projeto, eu acredito que aconteceu algo novo, que eu já debati algum tempo com os colegas, que não acontece... que até então não tinha acontecido aqui, na universidade em si. Porque quando se é solicitado um projeto, geralmente se divide em tarefas, e no final, une todos os resultados e entrega. Nesse projeto aconteceu diferente (...) era o primeiro contato com esse tipo de projeto... foram realizadas reuniões (...) onde foi discutido o como... o como fazer em si, não só de uma parte, de cada indivíduo, mas sim, no sentido de um realmente ajudar o outro, de... esclarecer dúvida que o colega tinha na questão, na tarefa a ser feita, e... houve... houve o debate pra... pra que fosse feita a questão, pra não ficar dúvidas, pra um ajudar realmente o outro (...) pra poder trabalhar realmente em todas as questões e não apenas ficar naquele paradigma que cada um faz um e no final une tudo e entrega"* (SR5).

O estudante ressaltou o trabalho com responsabilidade individualizada, mas ao mesmo tempo colaborativo. Diferente do que estava acostumado a praticar na Universidade, onde o trabalho em grupo termina sendo individual. Consequentemente, o estudante só aprende a atividade pela qual ficou responsável. De acordo com o exposto pelos demais estudantes, foi exatamente o que aconteceu em suas equipes. Um dos membros da equipe ER2 justificou esse comportamento, de modo geral, em virtude da sobrecarga de atividades e informou que somente as tarefas mais trabalhosas eram assumidas por dois membros, também para diminuir a sobrecarga. Tanto este estudante quanto um membro da equipe ER4 comentaram que as atividades eram distribuídas a depender da afinidade de cada membro.

Através desses relatos, identificamos que, das duas formas, há o trabalho em equipe. Porém enquanto na primeira forma, ressaltamos a colaboração necessária para enfrentar os grandes desafios, na segunda, o trabalho pode ser executado de forma mais rápida, desde que os membros estejam preparados para executar as atividades assumidas. É relevante que, qualquer que seja a situação, o estudante precisa ter comprometimento.

Portanto, podemos dizer que houve o desenvolvimento da habilidade de trabalhar efetivamente como membro de equipes (LO227). Contudo, só foi percebido por uma das equipes, dada a abordagem diferente que os estudantes adotaram. Contudo, do ponto de vista do uso de atividades em grupo como recurso para a aprendizagem, somente a situação colaborativa deveria existir, para que verdadeiramente ocorra a aprendizagem do todo.

Compreender como teoria e prática influenciam uma a outra

Quando verificamos se a abordagem de utilização de projetos de código aberto permite que o estudante faça a conexão do que é visto na teoria com a prática na Subseção 8.2.2, discutimos um conjunto de características detectadas a partir da percepção dos estudantes, de como a prática é importante para a teoria e de como a teoria é necessária para a prática. Vimos, além dos diversos aspectos onde a prática favorece a aprendizagem da teoria, que os estudantes reconheceram como a prática permite comprovar a aplicabilidade da teoria e que, caso eles não tenham o conhecimento do conteúdo suficientemente para executar a prática, precisaram buscá-lo antes de realizar a prática. Portanto, nesse contexto, certificamo-nos que os estudantes puderam compreender como teoria e prática são fundamentais e como influenciam uma a outra, consequentemente, que foi possível alcançar o objetivo LO270.

Expressar opiniões próprias e Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias

Ainda referenciando a Subseção 8.2.2, vimos que a discussão realizada após a execução da prática possibilitou que diferentes perspectivas pudessem ser expostas e que, assim, os estudantes conhecessem a opinião uns dos outros. Desse modo, a vivência de um projeto real comum propiciou a realização da discussão, que por sua vez permitiu que os estudantes 'externassem suas opiniões', escutassem as opiniões dos demais colegas e assim, 'desenvolvessem o pensamento crítico' e 'ampliassem a sua visão' sobre o tópico sendo discutido. Agregamos aos extratos expostos na subseção citada, os relatos a seguir,, a fim de evidenciar o desenvolvimento da capacidade de expressar opiniões próprias (LO232), da habilidade de demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias (LO241).

> *"(...) quando a gente ia pra aula e escutava as opiniões de todo mundo, chegava em um consenso. Isso foi bom para o aprendizado"* (SR7).

> *"Achei bom, porque muita coisa, assim, que eu tinha entendido de uma forma, e aí outra pessoa acabou explicando de outra forma, e aí a gente acaba tendo um entendimento melhor (...)"* (SR4).

> *"É aquele negócio... quando se está trabalhando com projeto não necessariamente... uma pessoa vai ter, vai ser a dona da verdade. Quando você expande... dá espaço pra outras pessoas e abre essa discussão, você consegue até, vamos dizer assim, cobrir pontos que... foram falhos na sua percepção do projeto... Algo que... você tem sua ideia concebida e aí, um colega seu comenta alguma coisa que: 'ops, é mesmo, realmente... realmente deveria ser feito desse jeito, dessa forma'. Pode alterar aquilo que eu pensei, adaptar um pouco mais para poder incorporar aquilo que o colega falou"* (SR8).

Finalmente, o extrato de outro estudante ainda expressa o desenvolvimento da autocrítica, excelente ponto de partida para a aprendizagem. O reconhecimento do erro é o primeiro passo para a procura pela solução correta.

> *"Essas atividades em sala serviram principalmente para a gente tirar algumas dúvidas, né? Porque a gente fez coisas lá que ... estavam erradas... e você falando na sala, explicando como era, foi tipo uma luz no fim do túnel. A gente entendeu o que realmente era pra fazer e o que a gente fez. Aí foi uma autocrítica entre os alunos e o que a gente fez... até pra saber se pode melhorar... Basicamente uma autocrítica do que você fez e você pode aprender muito mais com isso"* (SR2).

Vivenciar um projeto real e Compreender as expectativas de trabalho (habilidades esperadas)

Ao investigarmos a percepção dos estudantes sobre o contato com o projeto de código aberto, como uma experiência real de desenvolvimento de software em 8.2.1, verificamos que eles identificaram características que aproximam o projeto de código aberto à realidade dos softwares existentes nas empresas. Tais características aliadas às atividades que precisavam ser executadas, também similares às realizadas nas empresas, geraram uma lista de dificuldades para a execução das mesmas. Essa conjuntura permitiu que os estudantes visualizassem um conjunto de habilidades necessárias aos profissionais de engenharia de software. Desse modo podemos assumir que os estudantes, a partir da vivência de um projeto real (LO269), compreenderam as expectativas de trabalho para a profissão (LO203).

Julgamento sobre o alcance dos objetivos de aprendizagem associados às Competências da Prática Profissional

Para julgar se os objetivos de aprendizagem relacionados às competências da prática profissional foram ou não alcançados, aplicamos as heurísticas e o algoritmo definidos em 4.5, do mesmo modo que fizemos para julgar os objetivos associados aos aspectos técnicos. A única diferença é que neste caso, não aplicamos a dimensão relativa aos testes estatísticos, já que estes não foram realizados. Sendo assim, a Tabela 8.9 apresenta o julgamento para cada um dos objetivos. Com exceção da coluna 'melhora significativa', todas as demais colunas que representam as demais dimensões aparecem. A coluna 'análise dos trabalhos' expõe alguns comentários que resumem a nossa percepção a respeito do desenvolvimento de determinadas habilidades por parte dos estudantes com a realização do trabalho, discutidas ao longo do texto. A coluna 'concordância com a contribuição' denota o percentual do nível de concordância dos estudantes se a abordagem contribuiu para o alcance do objetivo de aprendizagem associado à competência da prática profissional (representa a soma dos percentuais de 'concordo' e 'concordo plenamente' na Figura 8.6). A coluna 'relato dos estudantes' apenas indica se ocorreram um ou mais relatos dos estudantes sobre o alcance do objetivo em questão. Cabe destacar que utilizamos um hífen ('-') para expressar quando não analisamos qualquer uma dessas dimensões para o objetivo sendo tratado. Finalmente, a coluna 'alcance do objetivo' aponta o julgamento final para cada objetivo de aprendizagem, após a aplicação das heurísticas citadas.

Tabela 8.9: Julgamento do alcance dos objetivos de aprendizagem associados às competências da prática profissional para o estudo de caso sobre requisitos de software (EC3).

Id.	Objetivo de aprendizagem	Análise dos trabalhos	Concordância com a contribuição	Relato dos estudantes	Alcance do objetivo
LO233	Aceitar e responder críticas de terceiros		100,0%		Sim
LO235	Avaliar o trabalho desenvolvido por terceiros		100,0%		Sim
LO268	Ter experiência com código desenvolvido por terceiros.		100,0%		Sim
LO236	Habilidade de ser proativo	Não demonstraram essa habilidade na execução do trabalho	80,0%		Parcial
LO237	Habilidade de ser criativo.	Não demonstraram essa habilidade na execução do trabalho	53,3%		Não
LO266	Ser capaz de solucionar problemas.	Não demonstraram essa habilidade na execução do trabalho	66,7%		Parcial
LO267	Gerar soluções alternativas para os problemas.	Não demonstraram essa habilidade na execução do trabalho	80,0%		Parcial
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.		93,3%		Sim
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).		80,0%		Sim
LO251b	Desenvolver a habilidade de sumarizar material técnico (código fonte e documentação).		66,7%		Parcial
LO246	Escrever documentação, clara, concisa e acurada	Muitos problemas de redação no texto	86,7%		Parcial
LO227	Habilidade de trabalhar efetivamente como membro de equipes.			Sim	Sim
LO270	Compreender como teoria e prática influenciam uma a outra			Sim	Sim
LO232	Expressar opiniões próprias.			Sim	Sim
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.			Sim	Sim
LO203	Compreender as expectativas de trabalho (habilidades esperadas)			Sim	Sim
LO269	Ter vivenciado pelo menos um projeto real.			Sim	Sim
LO273	Ser capaz de pensar o problema em vários níveis de detalhe e abstração			Sim	Sim

8.2.4 Sentimento de preparação para o mercado de trabalho (Ob8)

Buscamos investigar se o uso do projeto de código aberto permitiu ao estudante sentir-se preparado para o mercado de trabalho, a partir da percepção do estudante sobre três aspectos: (QSP1) confiança em realizar atividades semelhantes em outro projeto; (QSP2) contribuição para a melhoria do seu futuro desempenho na vida profissional e (QSP3) contribuição para que o mesmo se sentisse mais preparado para a realização das atividades de requisitos no mercado de trabalho. A Figura 8.7 apresenta os níveis de concordância obtidos.

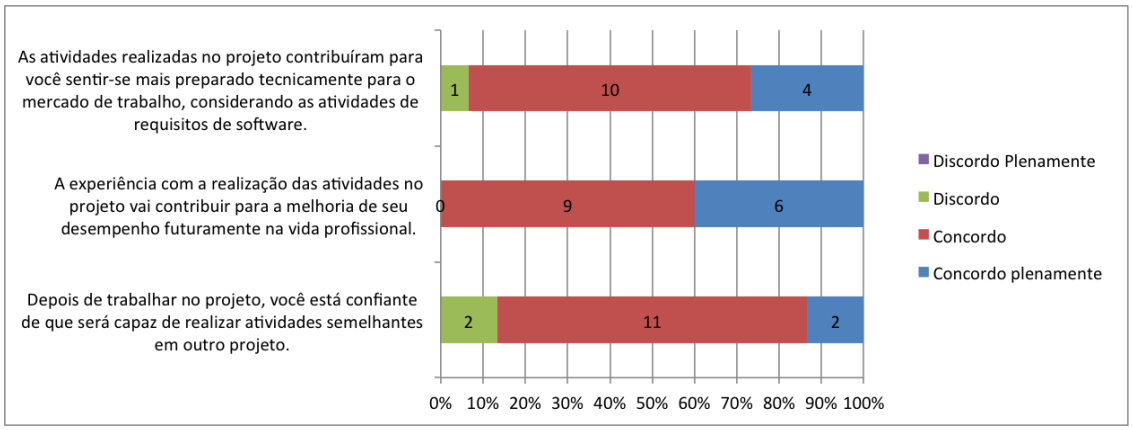

Figura 8.7 Níveis de concordância com relação à preparação para o mercado no estudo EC3.

Observamos que o projeto contribuiu em todos os aspectos para a maioria absoluta dos estudantes. Expressando em números, 86,67% dos estudantes estão mais confiantes que serão capazes de realizar atividades semelhantes em outro projeto (QSP1), 100%, concordam que as atividades no projeto contribuíram para a melhoria do seu desempenho profissional futuramente (QSP2) e, finalmente, 93,34%, consideraram que estariam mais preparados tecnicamente para o mercado de trabalho, considerando as atividades de requisitos de software, após as atividades do projeto (QSP3).

Do ponto de vista qualitativo, o projeto colaborou para que os estudantes sentissem-se preparados segundo os seguintes aspectos: (i) capacitação proporcionada; (ii) vivência de um projeto de software; (iii) visão profissional e (iv) desenvolvimento da autoconfiança.

No que diz respeito à capacitação proporcionada, um estudante apontou a contribuição da experiência com o projeto para o caso de precisar dar continuidade em algum software já existente, futuramente.

> *"O JabRef vai... vai facilitar muito... em caso de... conseguir dar continuidade ao código. Eu mesmo tinha esse problema, quando conversava com amigos e pegava um código e tentava... continuar de onde ele parou. Com o JabRef isso facilitou muito... essa compreensão" (...) "Talvez quando eu for para o mercado de trabalho, eu tenha que continuar um projeto já criado (...) Isso contribuiu muito"* (SR7).

Enquanto que outro estudante destacou que agora já tem noção do que precisará fazer, caso seja necessário dar manutenção em um software.

> *"Agora das atividades em si... esse novo mecanismo" (...) "saber modificar agora um software, né... que ainda não sabe, mas pelo menos já tem uma ideia... de opa, 'como muda essa funcionalidade aqui'... já tem uma ideia de qual o percurso fazer, em buscar lá na ferramenta"* (SR1).

Do ponto de vista de um terceiro estudante, a prática com o projeto permitiu-lhe ter um olhar mais detalhado, inclusive sobre os requisitos.

> *"Acho que foi o próprio entendimento também do projeto... assim, um olhar mais clínico... que a gente não tinha... até mesmo na hora de um requisito (...) a gente saber o que é caso de uso, o que não é"* (SR4).

Para a maioria dos estudantes, foi a primeira vez que 'vivenciaram um projeto real'.

> *"Foi meu primeiro contato"* (SR5).

> *"Bom, eu nunca desenvolvi algo tão completo assim, como o JabRef, sabe? Algo tão complexo, vamos dizer assim"* (SR7).

Por sua vez, um estudante, ao ser questionado sobre o que aprendeu ao realizar as atividades do projeto, enfatizou a experiência de lidar com um projeto existente.

> *"O que eu aprendi. Foi pegar... conseguir um pouco de experiência... receber um projeto já pronto, quase praticamente pronto, e fazer uma documentação sobre ele, que é uma coisa importante"* (SR2).

Posteriormente, afirmou ser essa experiência, a contribuição das atividades executadas para a sua formação profissional.

> *"Experiência, no caso. Porque eu tive que fazer mesmo, não recebi pronto. Tive que fazer, botar a mão na massa pra isso. Isso é experiência que você tem para o futuro, né"* (SR2).

A vivência do projeto de código aberto, consequentemente de um projeto real, permitiu o desenvolvimento de uma 'visão mais profissional'. Dois estudantes conseguiram enxergar que um projeto de software pode não ser tão simples quanto eles esperavam.

> *"Só que se for olhar hoje em dia, assim... depois do JabRef mesmo, eu acho que tem muito mais por trás do, do software, do que... do que a gente apenas tinha visto" (...) "Pela complexidade que assim... eu tava com uma visão muito... simples do que era realmente um projeto (...) Como os... os requisitos configuráveis mesmo, a gente nunca imaginaria que aquilo realmente era um requisito... que passaria a ser documentado ou algo do tipo"* (SR4).

> *"(...) verificando a ferramenta a fundo, não sendo mais usuário, e sim vendo a parte do desenvolvimento, a questão dos requisitos, aí sim eu percebi o quanto é ela é complexa. Porque às vezes nós vemos a ferramenta e não sabemos o quão complicada ela é. Só por estarem nos disponibilizando aquela funcionalidade 'legal, bacana, funciona', mas para ver como é que cada coisa foi implementada..."* (SR3).

Este último ainda vislumbrou que o desenvolvimento de software exige um esforço razoável, inclusive considerando o levantamento de requisitos. Visão distinta de quando se trabalha apenas com projetos pequenos, como ocorre tradicionalmente nas disciplinas. Por fim, concluiu que está mais maduro, uma vez que já sabe o que pode encontrar futuramente, principalmente, considerando a questão dos requisitos.

> *"E a partir do projeto eu pude perceber mais ainda sobre a dificuldade que é desenvolver um software" (...) "Só com a parte de requisitos... Eu percebo que... dá trabalho, porque fazer o levantamento de requisitos de um software que você vai construir... sempre pode ter algum requisito a mais que o usuário queira, então para você ter todos os requisitos é difícil! A gente só de sair selecionando e vendo, já achamos muitos, imagine uma coisa que vai incrementando cada vez mais que o usuário achar uma coisa: 'acho que deveria ter mais isso'... Aí pronto, já é de se esperar que é muita coisa... no mercado, com certeza, principalmente com essas ferramentas grandes" (...) "Bom... amadureci mais um pouco e já sei... o que me espera. Já sei que realmente vai ser muita coisa para se fazer... a gente pensa que um software é pouca coisa, porque às vezes a gente faz uns projetos aqui... mas é uma coisa pequena... aí a gente não tem essa maturidade de... de... A gente pensa que vai ser... a gente sabe no fundo que é grande, mas não tem a realidade. A gente sabe 'é vai ser grande', mas quando a gente vê, como eu vi na disciplina, que a gente vai... em cada detalhezinho ali..." (...) "A gente já sabe que... eu pelo menos já sei que quando eu for pro mercado... já sei que é muita coisa para se fazer. Só de requisitos"* (SR3).

Ainda, considerando o sentimento de preparação para o mercado, alguns estudantes informaram que estão mais seguros.

> *"Mas agora que teve o JabRef, que eu comecei a trabalhar com os requisitos, eu me sinto muito mais seguro para o mercado de trabalho"* (SR7).

Inclusive, outro estudante contou que havia tentado um estágio antes de cursar a disciplina, mas que estava *"com receio de entrar lá e não saber"*, mas agora, após a disciplina, ele já tem *"um caminho andado já... o projeto ajudou bastante"*.

No caso do estudante a seguir, este demonstrou confiança, ao ser questionado como se sentia se for preciso executar atividades de requisitos de software no mercado de trabalho, após participar do projeto.

> *"Eu acho que vou conseguir executar melhor essas atividades." (...) "antes de entrar na UFS, eu fiz o curso técnico na área de computação (...) eu tinha uma noção muito, muito vaga da questão de requisitos e tal (...) peguei a matéria, consegui puxar mais alguns elementos que me faltaram daquela noção inicial que eu tinha e... aí com o trabalho, eu consegui firmar esse conhecimento"* (SR8).

Estes relatos remetem ao desenvolvimento da autoconfiança, consequentemente ao resultado positivo quanto ao sentimento de confiança de que serão capazes de realizar atividades semelhantes em outros projetos (ver Figura 8.7). Porém, conforme apresenta esta figura, nem todos os estudantes estão confiantes sobre essa capacidade. Um desses estudantes,durante a entrevista, admitiu que está confiante para encontrar os requisitos funcionais, mas não ocorre o mesmo para os requisitos não funcionais (lembrando que esta dificuldade foi discutida ao longo do tópico 'Aprendizagem de aspectos técnicos' em 8.2.3).

> *"(...) eu acho que me sinto mais confiante em... em acreditar que eu consigo encontrar requisitos funcionais. E os requisitos não funcionais, por exemplo, eu acho que deu pra... melhorar... deu para ter ciência de que era preciso melhorar... na busca de requisitos não funcionais... Eu acho que... ajudou nesse sentido"* (SR6).

Ele ainda concluiu que o projeto foi importante para perceber *"que seria necessário, se aprofundar"* (SR6).

Outros estudantes também tiveram a mesma percepção, de que foi dado apenas o primeiro passo.

> *"Eu só senti que eu subi só [um lance de] uma escada... de uma escadaria enorme. Foi um avanço, um pouco de experiência, mas mesmo assim, tenho muito caminho para andar ainda, né? para chegar no mercado e ser um bom profissional"* (SR2).

Um estudante acrescentou que, após a realização do projeto, o impacto não será tão grande, porque teve pelo menos uma experiência ainda na Universidade.

> *"Agora estamos um pouco preparados, né, não vamos receber aquele impacto que com certeza a gente receberia, indo no mercado e... opa, 'agora é isso, eu não vi isso na graduação'. A gente se sente acho que mais preparado para enfrentar o mercado, que agora a gente pelo menos tem uma noção do que realmente a gente vai fazer no mercado"* (SR1).

Por sua vez, outro estudante também ressaltou a situação de que não haveria mais um impacto tão grande ao se defrontar com um desafio semelhante.

> *"Eu não me sinto totalmente pronto... pro mercado, ainda, de trabalho, mas com as atividades realizadas... no projeto em si e até na disciplina... eu acredito que... estaria... como é que vou dizer... mais ou menos preparado. Não vou chegar... não iria chegar de forma... cega, vamos dizer assim... com os olhos vendados, né... já teria um conhecimento bom, para realizar essas tarefas e eu não... acredito que não me assustaria tanto quanto chegasse ao mercado de trabalho"* (SR5).

Todos os extratos expostos corroboram os resultados quantitativos apresentados no início desta subseção, visto que todos os estudantes concordaram com a contribuição do projeto para o seu desempenho futuro (QSP2), mas ocorreram algumas discordâncias quanto à capacidade de realizar atividades semelhantes em outro projeto (QSP1) e ao sentimento de preparação técnica para o mercado de trabalho (QSP3). Como a disciplina onde aplicamos o estudo de caso representa a primeira em uma sequência de três disciplinas obrigatórias na área de engenharia de software, a sensação de os estudantes se considerarem ainda parcialmente preparados é, de certa maneira, esperada. Todavia, para o que se propõe para uma disciplina introdutória e tendo em vista tudo o que foi exposto, bem como a satisfação demonstrada pelos estudantes, julgamos que a abordagem atingiu seu objetivo.

8.2.5 Consequências para a aprendizagem (Ob5)

Paralelamente à investigação de objetivos específicos de aprendizagem, procuramos também explorar, de modo mais geral, quais as consequências para a aprendizagem de engenharia de software por parte do estudante, quando projetos de código aberto são empregados. Intuitivamente, através de uma perspectiva qualitativa, surgiram os temas que, de certo modo, já foram discutidos nas seções anteriores, mas que serão complementados por temas não apresentados ou por pontos de vista ainda não discutidos. Sendo assim, dividimos o conteúdo a ser apresentado nesta subseção nos seguintes tópicos: (i) projeto de código aberto como situação autêntica para a aprendizagem; (ii) mudanças de atitude; (iii) dificuldades a serem enfrentadas; (iv) escolha do projeto e (v) visão geral sobre a aprendizagem proporcionada.

Projeto de código aberto como situação autêntica para a aprendizagem
O projeto de código aberto por representar um software já existente possibilita uma abordagem diferente, particularmente considerando a área de requisitos de software. Tradicionalmente, os estudantes trabalham os requisitos a partir da construção de uma nova aplicação, como confirmado por um estudante.

> *"Eu nunca tinha pegado uma ferramenta, ao contrário, a gente sempre pega a ferramenta e constrói do começo"* (SR1).

Com o OSP, os requisitos podem ser recuperados a partir das funcionalidades providas e do código existente, o que representa uma abordagem complementar interessante para a aprendizagem do assunto.

> *"É... uma nova forma de ver... o trabalho assim... como documentação, vamos supor, ou realizar diagramas... é... porque não foi uma coisa que a gente pegou do zero e veio fazendo, realizando, desenvolvendo a aplicação e já documentando. A gente foi ao contrário agora... a gente pegou a aplicação já pronta e veio documentando... Ter esse contato, assim, ao reverso dessa tarefa... acredito que contribuiu bastante pra consolidar realmente o conhecimento"* (SR5).

Tal abordagem permitiu que os estudantes visualizassem que nem sempre o engenheiro de software trabalha com um software, construindo-o desde o início.

> *"Foi bem... foi bem benéfico para a gente essa aprendizagem, porque acredito que a maioria da turma nunca chegou a se deparar com uma situação dessa, todo mundo esperava realmente que o software seria construído do começo"* (SR1).

Sobretudo, a disponibilidade do código em projetos de código aberto é que permite a aplicação desse tipo de abordagem.

> *"Talvez se não tivesse o JabRef, a gente não iria trabalhar com essa teoria [engenharia] reversa"* (SR7).

Segundo um estudante, o OSP também permite que o estudante fuja do usual, ao ter que lidar com um software que não foi criado por ele.

> *"(...) até a questão também da gente sair um pouco do... da construção da gente, de uma ideia da gente, e também praticar num projeto"* (SR4).

Reconhecidamente, trabalhar com um software que representa um software real gera interesse nos estudantes.

> *"O que mais chamou minha atenção no projeto foi o contato com um projeto real"* (SR13).

Para outro estudante, a experiência com o projeto de código aberto acrescentou ao método comumente aplicado, a possibilidade de enxergar de outra maneira o conteúdo estudado, com uma visão próxima da realidade.

> *"Foi... foi um complemento. Nós vimos tudo em sala de aula, os conceitos, fizemos atividades práticas no laboratório, fazendo os diagramas e tudo. E com a ferramenta [referindo-se ao JabRef]... pudemos ver de outra forma o que nós estudamos... pudemos ver como é na realidade as coisas" (...) "é uma forma de visualizar de outra forma o que foi visto em sala de aula... ver real"* (SR3).

A proximidade com a realidade, também foi percebida por mais estudantes.

> *"Então foi... trabalhou muito a realidade do que vai se encontrar... no mercado... no futuro"* (SR5).

De acordo com o relato a seguir, esta proximidade também favoreceu que o estudante reconhecesse o fato relevante da aprendizagem ser contextualizada, onde o estudante não precisa transpor o conteúdo estudado para um caso real, como acontece usualmente.

> *"Eu acho que foi bom... foi bom, porque... basicamente... a gente vê muito aqui na UFS, teoria, teoria, teoria, só que quando a gente sai daqui... a gente vai demorar um certo tempo, para conseguir realmente moldar aquilo que a gente viu aqui, pra o mercado, pra o que a empresa precisa, né..."* (SR8).

O projeto de código aberto exibe outras características que os tornam ainda mais próximos ao que os estudantes irão encontrar futuramente ao trabalhar na indústria de software. Como mencionamos anteriormente, o código do OSP é desenvolvido por terceiros, por conseguinte exibe diferentes estilos de programação. O extrato a seguir exprime exatamente essa situação.

> *"(...) uma outra pessoa tinha escrito o código e ela escreve da maneira dela, entendeu? E eu tenho a minha maneira de escrever o código"* (SR7).

Por representar um exemplo 'concreto' de projeto de software, os estudantes podem extrapolar e enxergar como conceitos, práticas ou processos que estão sendo estudados poderiam ser aplicados no mesmo. O relato de um estudante retrata essa situação.

> *"Supondo que não tivesse o projeto do JabRef, a gente ia sair só a questão de... ia sair só sabendo a teoria de cada processo, não é isso? ... com o JabRef, no caso... no meu caso, eu comecei a... por exemplo, via o código e lia o manual e já começava a pensar em como aquele projeto poderia ser aplicado em um... em um processo de software... por exemplo, no Scrum ou então... quais as vantagens que eu poderia... quais as vantagens que eu teria utilizando o projeto a partir daquele ponto no Scrum"* (SR7).

No caso do estudante a seguir, comparando o processo ágil ao processo de desenvolvimento do projeto de código aberto, ele reconheceu que a discussão baseada no exemplo concreto do projeto permitiu que realmente enxergasse algumas similaridades entre esses processos, diferentemente se a explicação fosse puramente teórica. Ou seja, o uso do projeto permitiu que sua visão fosse ampliada.

> *"(...) na última discussão, que até compararam o ágil com, com a ferramenta [referindo-se ao processo utilizado no JabRef] ... É uma questão que realmente... tem uma similaridade mas a pessoa não vê! Por isso que eu acho também importante a discussão... tipo, realmente a falta de documentação mais a questão de... de priorizar o desenvolvimento, lembra muito o movimento ágil... Sei que nós como alunos aprendemos tudo isso na teoria, aí*

> *quando vai ver a prática... quando tá lá na prática também não consegue exatamente 'pô cara, eu vi isso'. Realmente isso... isso se aplica a essa ferramenta... ao projeto open source... É bastante, bastante interessante porque abre muito a mente das pessoas (...)"* (SR3).

Finalmente, os estudantes também se aperceberam do ambiente de aprendizagem disponível, não só para aprender sobre projetos de código aberto, mas para aplicar o conteúdo estudado, poder visualizar os erros e acertos e, enfim, ganhar mais experiência.

> *"(...) com o entendimento que a gente teve, a gente pode até... com outros projetos ou até mesmo com o JabRef ... pegar para... ter uma noção... ter um conhecimento maior dos projetos open source e até também a experiência que teve... de colocar em prática a teoria que a gente estudou.. e ter uma experiência melhor"* (SR4).

> *"Eu tenho muita deficiência na questão de programar ainda, e eu encontrei no projeto open source uma forma de aprendizado, uma forma de autoaprendizado. Olhar o código e ver... os acertos e os erros daquele código e com isso pode ajudar... pode contribuir para eu escrever o meu"* (SR7).

Um dos estudantes comentou que anteriormente pensou em contribuir com um OSP, porém teve receio, dada a falta de experiência. Após a realização das atividades do projeto compreendeu que existem outras formas de contribuição como é o caso de criar documentação, isto é, não é tão difícil participar do projeto.

> *"Até pensei em participar de algum, só que minha experiência não... não é tanta como eles necessitam, né. Então, pretendo praticar muito mais, isso que você me explicou, como pegar... por exemplo... eu tenho um projeto lá, (...) que é uma plataforma de um sistema da universidade (...) que eu até salvei lá, para poder fazer do mesmo jeito que eu fiz em DS I, agora... que era pegar e fazer uma documentação para ele... que ele não tem documentação"* (SR2).

Portanto a utilização do projeto de código aberto no processo de ensino-aprendizagem de engenharia de software, ao mesmo tempo que foge do método usual de ensino, propicia outras oportunidades de aprendizagem e a sua proximidade com a realidade faz com que concluamos que representa uma situação autêntica para a aprendizagem. Na conjuntura específica para a aprendizagem sobre requisitos, representa uma maneira complementar, mas também relevante, uma vez que muitas vezes os profissionais precisam trabalhar com softwares existentes que não estão documentados.

Mudanças de Atitude

A partir da abordagem de emprego do OSP também capturamos a conscientização dos estudantes sobre diversos aspectos, entre eles: (i) importância da documentação; (ii) importância da modelagem; (iii) importância das práticas propostas pela Engenharia de

Software; (iv) importância do rastreamento entre requisitos e (v) esforço necessário para construir o software.

Um estudante conscientizou-se sobre a importância da documentação do software após enfrentar as dificuldades de lidar com o projeto, que não estava adequadamente documentado.

> *"(...) tem que fazer uma boa documentação, porque, sofri, deu pra sofrer. Sem documentação, sem organização, sem tempo definido, não vai pra lugar nenhum"* (SR2).

Enquanto que outro estudante compreendeu a importância da modelagem ao deparar-se com um software maior e mais complexo.

> *"Toda vez que eu queria fazer um projeto, eu pensava no DER do projeto, sempre era no DER. Fazia primeiro uma modelagem no DER, que eu já fazia no banco de dados e depois eu jogava a aplicação por cima dele. Eu nunca pensei em poder fazer primeiro o levantamento de estudos de casos [casos de uso], detalhar ele, o modelo conceitual para ter uma noção do negócio, para depois sim elaborar uma DER, eu já partia direto para o DER. Até porque os projetos eram pequenos e dava para fazer isso né, mas com um projeto muito grande, você sabe que se partir para DER, não vai dar certo." (...) "eu nunca elaborava esses diagramas e eu não sabia da importância realmente deles (...)"* (SR1).

Por sua vez, um terceiro estudante, comentando sobre as habilidades necessárias ao profissional da indústria de software, manifestou que a realização das atividades deu-lhe ciência sobre as várias recomendações, práticas ou métodos propostos pela Engenharia de Software e a importância de aplicá-los.

> *"Eu não imaginava que tinham tantas regras. Assim, não é regra porque não é obrigado... o povo usar, mas... para ser um bom profissional é bom segui-las, é aconselhável segui-las, e eu não imaginava que tinham tantos detalhes que nós temos que saber. Eu realmente não imaginava isso (...)"* (SR3).

Outro estudante, ao falar sobre a contribuição do projeto para a sua formação profissional, demonstrou estar ciente da importância do mapeamento de interdependência entre os requisitos, isto é, do rastreamento entre requisitos.

> *"Eu acho que uma das atividades realizadas que... que eu acho que mais contribuiu foi a questão do... o mapeamento de requisitos... aquele cruzamento, onde eu verifico qual requisito influencia no outro, ou qual requisito influencia no caso de uso, ou caso de uso que tá relacionado a um certo requisito. Acho que esse foi um dos pontos mais... importantes"* (SR8).

Por último, tendo que lidar com um exemplo real de software, um estudante conscientizou-se sobre o esforço de fato necessário para construir um software, passando a melhor valorizar o trabalho dos engenheiros de software.

> *"E... dar valor também ao trabalho do engenheiro de software... também, porque a gente às vezes vê a coisa lá pronta, 'tá fácil, tá bom', às vezes até reclama quando o negócio não funciona direito, 'ave, não tá funcionando', mas não sabe o trabalho que dá fazer aquilo! Aí, a partir do... do projeto é que a gente sente mais no couro mesmo como é, como é trabalhoso fazer, desde a questão, questão teórica, dos levantamentos, até a prática que é a implementação, porque verificar o código também... dá trabalho, imagine implementar, quem tiver que implementar... com certeza. Aí dá esse choque de realidade, dá, realmente com certeza dá! Porque como eu tava fazendo, só os levantamentos dos requisitos, só de um topicozinho deu várias páginas, eu pensei 'rapaz, imagine quem teve que fazer de todo o sistema'. Realmente, implementar, mais do que isso ainda"* (SR3).

Quantativamente, procuramos identificar a percepção dos estudantes sobre a contribuição da abordagem para que eles compreendessem a importância dos seguintes aspectos: (i) especificação de requisitos (QA1); (ii) documentação do software (QA2) e (iii) engenharia de requisitos (QA6).

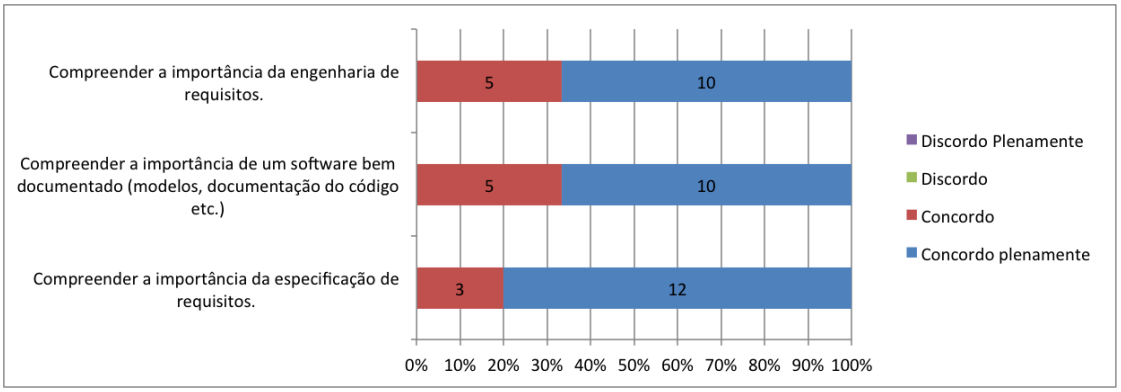

Figura 8.8 Níveis de concordância com relação à contribuição do projeto para conscientização dos estudantes no estudo EC3.

A Figura 8.8 apresenta que a grande maioria dos estudantes concordaram plenamente que a abordagem contribuiu para que os mesmos compreendessem a importância de todos os aspectos questionados. Estes resultados corroboram os resultados qualitativos mencionados.

Dificuldades a serem enfrentadas

Ao longo de qualquer que seja o processo de ensino-aprendizagem, dificuldades surgem, sejam provenientes da própria abordagem ou recurso sendo aplicado, sejam provenientes do contexto de aplicação da mesma. Como essas dificuldades influenciam direta ou indiretamente a aprendizagem dos estudantes, decidimos descrevê-las ao longo do texto, a exemplo das dificuldades associadas ao emprego de projetos reais e projetos de código aberto, discutidas em 8.2.1. Nesta subseção, discorreremos sobre as dificuldades oriundas do contexto de aplicação da abordagem.

Identificamos as seguintes dificuldades associadas à aplicação da abordagem: (i) ter dificuldades com programação; (ii) entendimento das atividades solicitadas (iii) falta de participação de todos os membros da equipe; (iv) tempo curto e (v) sobrecarga do final do período. Como veremos todas essas dificuldades influenciaram a execução das atividades por parte dos estudantes.

Apesar de apenas uma das atividades solicitadas exigir que os estudantes lidassem com o código do software propriamente dito, um estudante admitiu a sua deficiência com programação como fator dificultante.

> *"(...) eu não tive... um bom aprendizado nas matérias de programação, isso pode ter dificultado muito"* (SR7).

Nesse aspecto, concluímos que para lidar com projetos reais em nível de código, é essencial um conhecimento prévio sobre programação bom, caso contrário o desafio tornar-se-á muito grande.

Quanto à dificuldade de entendimento das atividades solicitadas, um estudante apontou-a como uma das principais dificuldades a serem enfrentadas na execução das atividades.

> *"Acho que interpretação... pra entender o que a questão pedia..."* (SR6).

Inclusive em outro momento, admitiu ter sido essa a razão pelas diversas dúvidas encaminhadas.

> *"(...) a gente depois mandou dúvidas, né, mandou algumas perguntas, mas... acho que talvez se a gente tivesse tido mais tempo para interpretar as questões, talvez a gente não tivesse mandado tantas dúvidas (...)"* (SR6).

Já outro estudante interpretou a dificuldade de entendimento da questão como a razão para que seus colegas não tivessem realizado as atividades adequadamente.

> *"Um pouco do conhecimento, às vezes o pessoal não tá entendendo muito bem o que a questão pedia e... Por exemplo, quando eu tava fazendo a revisão do trabalho, eu achei uma porrada de furos e tal, aí eu peguei, sentei e expliquei: 'olha, tô mexendo nessa parte aqui, essa parte é sua, tô mexendo, eu tô mexendo por isso. Eu entendi que a resposta seria mais próxima disso aqui, por isso, isso, e isso. Você concorda? Sim, dê uma olhada como tá o documento agora'. Aí o colega olhava e dizia: 'realmente, era isso mesmo, tal, eu não entendi muito bem isso, me explique direito como você chegou a essa resposta' "* (SR8).

Por outro lado, segundo o estudante a seguir, a explicação sobre o projeto foi adequada.

> *"Ficou muito claro. Eu saí da aula já sabendo o que era para fazer"* (SR7).

Para esse estudante, a dificuldade ocorreu devido à falta de comprometimento de alguns indivíduos, gerando sobrecarga de trabalho para os demais membros da equipe, ou seja, faltou a participação de alguns dos integrantes da equipe.

> *"Na primeira parte do trabalho... houve uma divisão. Mas teve um ou outro que não... se quer... nem... se comprometeu a fazer... isso sobrecarregou os outros. Nessa segunda parte do trabalho foi mais complicado porque éramos em quatro, e somente dois fizeram, entendeu? Só na hora da entrega do trabalho foi que a gente recebeu um e-mail dos outros dois dizendo que tinham desistido da matéria"* (SR7).

Talvez a falta de comprometimento tenha ocorrido pela dificuldade de interpretação das questões ou vice-versa. Outras explicações possíveis para a dificuldade de interpretação seriam a novidade da abordagem solicitada (execução da engenharia reversa) ou as questões discursivas não demandarem respostas diretas. De fato, exigiam alguma reflexão, síntese ou análise crítica por parte dos estudantes.

De acordo com os estudantes, outra dificuldade sentida na execução das atividades correspondeu ao tempo curto para executá-las. Inclusive dois estudantes chegaram a sugerir que o projeto fosse iniciado mais cedo, para que o prazo para realizá-lo fosse ampliado.

> *"Acho que faltou tempo também, acho que se tivesse mais tempo para fazer esse trabalho. Isso foi geral, assim, os quatro [referindo-se aos membros da equipe] tavam questionando isso, 'poxa talvez se a gente tivesse mais tempo talvez saísse melhor' "* (SR6).

> *"(...) se o trabalho fosse... começado antes, logo no início do período, quando a gente poderia dar uma estudada no código, antes de começar a realizar as atividades"* (SR7).

> *"(...) acho que seria mais proveitoso... poderia até tentar trazer mais no início do curso... logo depois que a gente viu o modelo poderia começar trabalhando, seria algo mais interessante. Pegamos da metade para o final, acho que esse foi um tempo muito curto e a correria... para poder desenvolver algo melhor. Mas no fim de tudo, gostei"* (SR1).

Para este último, o maior obstáculo foi a sobrecarga de atividades do final do período, onde os estudantes usualmente têm várias tarefas a cumprir das diversas disciplinas que estão cursando.

> *"Mas acho que a maior dificuldade foi na questão dessa segunda parte ser no final do período e aí você está sobrecarregado com outras coisas. Você quer resolver logo, para poder partir pra outra coisa... mas não dificuldade em si no projeto, depois que foi esclarecido [algumas dúvidas], deu para fazer"* (SR1).

Escolha do projeto

Outro fator que acarreta em consequências para a aprendizagem corresponde à escolha do projeto de código aberto a ser empregado na abordagem. Tal escolha influencia diretamente desde a própria aplicação da abordagem, por limitações impostas devido às especificidades do software, até as dificuldades a serem enfrentadas na execução das atividades, motivação e satisfação dos estudantes, como discorremos a seguir.

Ao discutirmos as dificuldades de aplicação da abordagem na Seção 8.1.3, mencionamos que a escolha do JabRef limitou os objetivos de aprendizagem que poderíamos trabalhar com a abordagem em virtude da ausência de modelos ou documento de especificação dos requisitos. Ao mesmo tempo que argumentamos em 8.2.3, que o software não foi muito adequado para a atividade de identificação dos requisitos funcionais e não funcionais (para o alcance do objetivo de aprendizagem LO1), dada sua característica de autoconfiguração que terminou por confundir os estudantes. Estas situações demonstram Concluímos assim que nem sempre o OSP vai atender a todas as necessidades didáticas da disciplina, fato de certa forma esperado, uma vez que, não representa um software criado especificamente para este fim.

Considerando as perspectivas de motivação e satisfação, que indiretamente influenciam a aprendizagem, detectamos que os estudantes consideraram o domínio do software útil e mais importante, faz parte do seu contexto, visto que já o utilizaram ou podem vir a utilizá-lo.

> *"Bom, eu já havia conhecido um pouco essa ferramenta, eu utilizei em um curso que fiz Latex, aí, e eu tive um pouco de experiência de como utilizar. Não cheguei a utilizar muito, mas já percebi que é bastante útil. Já utilizei na minha pesquisa para referenciar, citar algum autor. Aí já achei interessante"* (SR3).

> *"Em relação à ferramenta, pode ser uma ferramenta útil né, na construção do TCC"* (SR1).

Sob o ponto de vista da situação do código e da documentação do JabRef, descrevemos uma série de dificuldades em 8.2.1 relatadas pelos próprios estudantes, mas que também são comuns de ocorrerem quando os projetos existentes nas empresas precisam ser trabalhados. Apesar de os estudantes apontarem a complexidade do projeto, a estrutura confusa, a falta de documentação, entre outros, como obstáculos a serem superados, apenas um estudante demonstrou real insatisfação com o OSP selecionado. Para este estudante, a complexidade do JabRef não era adequada para estudantes que estavam cursando a primeira disciplina relacionada à engenharia de software.

> *"Assim, em relação... eu achei o conceito do projeto em si, interessante. Mas eu realmente não gostei do... programa selecionado pra isso, eu achei que é um programa realmente muito... pra quem tá começando (...) é um programa muito complicado (...) pra quem tá começando mesmo... terrível, terrível!"* (SR8).

Apesar de respeitarmos a opinião do estudante e que a mesma deva ser levada em conta nos próximos empregos da abordagem, vale lembrar que sua opinião não foi compartilhada pelos outros estudantes entrevistados, conforme discutimos em 8.2.1. Ademais, os extratos a seguir ilustram a satisfação dos estudantes com a abordagem.

> *"Acho que já falei tudo, sobre a experiência. Eu gostei"* (SR1).

> *"E pegar do zero assim, e você ver o que tá funcionando, o jeito que ele funciona e depois tentar descrever ele, é algo complexo, né, pela primeira vez, foi complexo. Mas foi divertido"* (SR2).

> *"Acho que colocar em prática a busca de requisitos, de, de classes, a, a ligação entre classes, o relacionamento no caso, né. Eu acho que colocar em prática o que a gente viu, o que a gente aprendeu na unidade anterior, colocar em prática nessa.. no projeto... eu achei interessante"* (SR6).

> *"Eu achei bastante gratificante. Dá trabalho mas... dá muito trabalho, mas quando se vê o resultado (...) é bastante gratificante"* (SR3).

Após a exposição desses fatos, concluímos que nem sempre um OSP vai atender a todas as necessidades didáticas da disciplina, fato de certo modo esperado, uma vez que não representa um software criado especificamente para este fim. Por outro lado, deve-se estudá-lo previamente para julgar quais objetivos de aprendizagem podem ser trabalhados e se o desafio proposto não será inapropriado ao nível de conhecimento dos estudantes.

Visão geral sobre a aprendizagem proporcionada
Para apresentar uma visão geral sobre esta aprendizagem, elaboramos a Figura 8.9. Portanto, através da análise qualitativa realizada, identificamos percepções sobre a aprendizagem obtida para curto e longo prazo. Na ótica de curto prazo, detectamos a compreensão de conceitos e o desenvolvimento de habilidades técnicas, a exemplo de entender o que é e como ocorre a engenharia de requisitos, e, como identificar casos de uso e construir diagramas. Sob o ponto de vista em longo prazo, destacamos: (i) o desenvolvimento de habilidades gerais ligadas à melhoria de competências da prática profissional; (ii) mudanças de atitudes, com a conscientização da importância da documentação, das práticas propostas pela Engenharia de Software e da valorização do trabalho do desenvolvedor de software, por exemplo, e, finalmente, (iii) a experiência profissional proporcionada, incluindo desde a vivência de um projeto, até o desenvolvimento da autoconfiança dos estudantes. Lembramos que discutimos cada um desses tópicos nesta e em subseções anteriores (8.2.3 e 8.2.4).

8.2.6 Lacunas na Indústria de Software (Ob9)

O último objetivo da pesquisa foi investigar se os objetivos de aprendizagem alcançados com esta abordagem ajudam a suprir algumas das lacunas identificadas pela indústria de software na formação dos estudantes. Portanto, retornando ao Capítulo 3, analisamos quais deficiências de formação foram direta ou indiretamente trabalhadas por meio

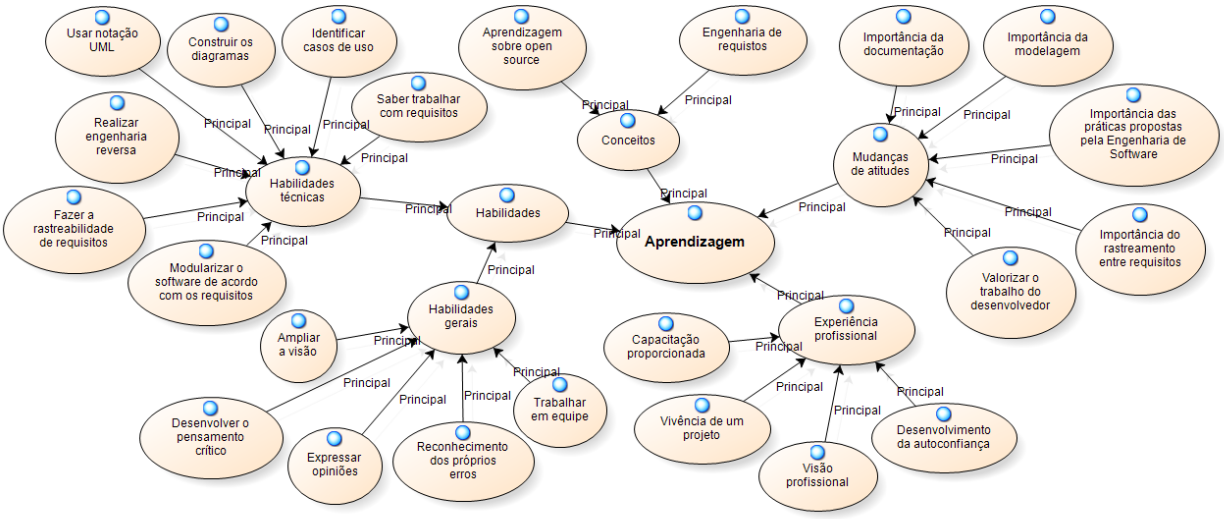

Figura 8.9 Contribuição da abordagem para a aprendizagem no estudo de caso sobre requisitos de software (EC3).

da abordagem, em virtude dos conteúdos e competências abordados pelas atividades do projeto. A Tabela 8.10 apresenta as deficiências relacionadas à área de requisitos de software e à prática profissional, bem como os objetivos de aprendizagem associados. A última coluna indica se a abordagem contribuiu, contribuiu parcialmente ou não contribuiu para o suprimento da deficiência, a partir dos objetivos de aprendizagem alcançados (Tabelas 8.6 e 8.9). Assinalamos as deficiências que não foram diretamente tratadas pela abordagem com um hífen ('-').

Considerando às lacunas referentes à área de requisitos de software, enumeramos apenas três deficiências na tabela. Porém, convém lembrar que apenas as deficiências com mais ocorrências foram listadas pelos estudos utilizados como referência para identificação dessas lacunas (ver Capítulo 3). Dentre os objetivos de aprendizagem alcançados para esta área de conhecimento (Tabela 8.6), somente o objetivo 'capacidade de modelar e especificar os requisitos de software de médio porte' (LO6) estava associado a uma dessas deficiências. Nesse caso específico, como o objetivo em questão foi somente parcialmente alcançado, podemos dizer que a contribuição para o suprimento da deficiência foi também parcial.

Cabe salientar que não trabalhamos na aplicação da abordagem os objetivos de aprendizagem associado às deficiências 'preparação e condução de entrevistas para a elicitação de requisitos' (LO89) e 'elicitação de requisitos' (LO3). Primeiro, porque precisaríamos de muito mais tempo ao longo da disciplina para propor o desenvolvimento de uma nova funcionalidade para o software e assim trabalharmos estes objetivos de aprendizagem. Segundo, porque a realização de entrevistas (LO89) corresponde a um método bastante incomum para o levantamento de requisitos em projetos de código aberto. A quantidade e a distribuição dos usuários, geralmente encontradas em tais projetos, inviabilizam o emprego dessa técnica de levantamento de requisitos.

Tabela 8.10 Contribuição da abordagem para o suprimento das deficiências apontadas pela indústria de acordo com o estudo de caso de Requisitos de Software (EC3).

Deficiência	**LO**	**Contribuição**
Requisitos de Software		
Preparação e condução de entrevistas para a elicitação de requisitos	LO89	-
Elicitação de Requisitos	LO3	-
Especificação de Requisitos	LO6	Parcial
Prática Profissional em Engenharia de Software		
Comportamento ético	LO199	-
Comprometimento com a aprendizagem contínua	LO326	-
Compreensão das expectativas que recaem sobre o profissional da área	LO203	Sim
Motivação e disposição	LO238	-
Reconhecimento e aprendizagem com os próprios erros, aceitar respostas de terceiros	LO233	Sim
Gerenciamento do tempo	LO234	-
Habilidades de liderança	LO240	-
Experiência com trabalho em equipe	LO227	Sim
Participação em equipes multidisciplinares	LO228	-
Pensamento crítico, geração e síntese de novas ideias	LO241	Sim
Solução de problemas	LO266	Parcial
Propor soluções alternativas	LO267	Parcial
Pró-atividade	LO236	Parcial
Paixão por tecnologia ou pelo domínio com o qual está trabalhando	LO239	-
Experiência com projetos reais	LO269	Sim
Habilidade de visualizar o projeto como um todo (visão do todo)	LO272	Sim
Habilidade de comunicação escrita para produção de documentação técnica	LO246	Parcial
Habilidade de comunicação oral (uso de jargões e habilidades de escuta)	LO327	-
Habilidade de apresentação	LO311	-
Hesitação em solicitar ajuda	LO249	Parcial
Articulação de argumentos e fornecimento de explicações claras	LO244	Não

No que concerne às deficiências de formação relacionadas à prática profissional, a Tabela 8.10 mostra que houve contribuição, mesmo que em alguns casos apenas parcial, para o suprimento das deficiências, de acordo com o alcance dos objetivos de aprendizagem associados (Tabela 8.9). Particularmente para as deficiências associadas aos objetivos LO244 e LO249, apesar de não termos investigado diretamente estes objetivos, chegamos a algumas conclusões. Para a deficiência em 'articular argumentos e fornecer explicações claras' (LO244), consideramos que não houve contribuição, tendo como base as respostas apresentadas nos trabalhos escritos, principalmente devido aos problemas de redação detectados nos documentos entregues por algumas equipes. Entretanto, convém destacar que o problema fundamental está na capacidade de escrita, que, por conseguinte foge do escopo da disciplina. Inclusive, ao longo das discussões presenciais realizadas, os estudantes expuseram suas opiniões e seus argumentos. Porém, como não utilizamos

nenhum instrumento para o registro do debate ocorrido, não pudemos analisar em mais detalhes, se houve contribuição para o suprimento desta deficiência. Para a deficiência de 'hesitar em solicitar ajuda' (LO249), verificamos que os estudantes procuraram ao longo da execução das atividades do projeto tirar suas dúvidas com a pessoa que propôs o trabalho, no caso, esta pesquisadora. No entanto, através do extrato da entrevista com o estudante SR5, percebemos algum receio de interagir com a comunidade do projeto (conforme exposto no tópico 'utilização de projetos de código aberto' na Subseção 8.2.1). Por essa razão, consideramos que a contribuição foi apenas parcial. Contudo, caberia uma investigação mais detalhada para identificar as causas dessa hesitação.

Capítulo

9

COMPARAÇÃO ENTRE OS ESTUDOS E DISCUSSÃO

Após aplicarmos e analisarmos individualmente os resultados da abordagem de uso de projetos de código aberto como recurso didático na aprendizagem de evolução e manutenção de software (EC1) (Capítulo 6), testes de software (EC2) (Capítulo 7) e engenharia reversa de requisitos (EC3) (Capítulo 8), resta-nos fazer a triangulação entre esses achados. Sendo assim, o objetivo desse capítulo é identificar as interseções e as particularidades dos resultados dos três estudos de caso, discutindo-as à luz dos objetivos traçados para a pesquisa e da literatura existente, particularmente considerando a síntese realizada das evidências reportadas pelos estudos relacionados (RS[1]) (Capítulo 5), a fim de que possamos chegar a conclusões mais apropriadas.

Dentre os resultados encontrados, reportamos também as dificuldades enfrentadas pelos estudantes devido à utilização de projetos reais, particularmente pelo emprego de projetos de código aberto, bem como as dificuldades provenientes da aplicação da abordagem, consequentemente, discutimos os desafios enfrentados pelo professor.

Convém ressaltar que apresentamos os achados por meio de tabelas, de maneira que os resultados de cada estudo estarão em uma mesma coluna, descrições em uma mesma linha corresponderão a interseções entre os estudos e um único resultado em uma linha representará uma especificidade do respectivo estudo.

9.1 EXPERIÊNCIA REAL DE DESENVOLVIMENTO DE SOFTWARE (OB7)

O principal objetivo em usar projetos de código aberto na educação em engenharia de software é prover mais realismo à abordagem didática. Ou seja, que a experiência do estudante com o projeto seja mais próxima do que irá encontrar futuramente no ambiente de trabalho. Porém, ao aplicarmos a abordagem, as primeiras dúvidas que surgem são: Como os estudantes enxergam o OSP? Eles enxergam o OSP como um projeto real? Como os estudantes enxergam as atividades que foram solicitadas? Eles enxergam as atividades como algo que é próximo ao que precisarão executar no mercado de trabalho?

[1]Sigla que utilizaremos para descrever resultados oriundos da síntese apresentada no Capítulo 5.

A percepção do estudante sobre a experiência com o projeto de código aberto como uma experiência de mundo real é relevante para que o mesmo entenda o objetivo do desafio proposto. Caso contrário, não haverá motivação para trabalhar com um projeto maior e muitas vezes mais complexo do que está usualmente acostumado. Por esta razão, decidimos investigar essa percepção.

Proximidade dos projetos de código aberto aos projetos existentes nas empresas

Nos estudos realizados, 93% (EC2 e EC3) e 89% (EC1) dos estudantes consideraram que o projeto empregado (JabRef) representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software, enquanto que 100% (EC2 e EC3) acreditaram que o JabRef permitiu que eles tivessem uma experiência de mundo real (essa questão não foi investigada quantativamente no estudo EC1). Semelhantemente, a síntese das evidências (RS) também confirmou a percepção dos estudantes que o projeto de código aberto 'provê uma experiência de mundo real' (ver Tabela 5.12).

A Tabela 9.1 apresenta as características descritas pelos envolvidos em cada um dos estudos, que aproximam o JabRef dos softwares que são trabalhados nas empresas.

Tabela 9.1 Características do JabRef que o aproximam dos softwares existentes nas empresas.

EC1	**EC2**	**EC3**
Tamanho.	Tamanho.	-
Complexidade.	Complexidade.	Complexidade.
Problemas na estrutura do código – Erosão da Arquitetura.	Problemas na estrutura de código.	Estrutura confusa.
Desenvolvido por várias pessoas.	Desenvolvido por várias pessoas.	-
Problemas com a documentação.	Escassez de documentação.	Escassez de documentação.
-	Exige configuração do ambiente.	-
Utilizado por várias pessoas.	Existe *feedback* do usuário.	-
-	Software implementado sem simplificações.	O projeto não é simplificado.
Difícil de entender.	-	-

Através da Tabela 9.1 vemos que há uma percepção praticamente homogênea dos envolvidos sobre as características que aproximam um projeto de código aberto dos projetos trabalhados pela indústria de software. Algumas características foram apontadas nos três estudos, outras em dois estudos. Apenas as características de 'exigir a configuração de ambiente' e a 'dificuldade de entendimento' foram apontadas por em único estudo. Veremos no decorrer desse capítulo que estas características ainda foram apontadas como dificuldades a serem enfrentadas.

Convém ressaltar que tais características não podem ser generalizadas. Isto é, do mesmo modo que podem existir projetos nas empresas bem estruturados, podem existir

projetos de código aberto bem estruturados, ou mesmo, que podem existir projetos de empresas e projetos de código aberto que são simples. Na verdade, elas representam características que são usualmente encontradas, mas que variam muito de projeto para projeto.

Salientamos uma particularidade no estudo EC3, no qual um dos estudantes assinalou que a complexidade do projeto não era adequada para estudantes iniciantes. Apesar dessa opinião não ter sido compartilhada por outros estudantes, deve ser realçada, uma vez que poderá ocorrer sempre que um projeto real for utilizado como recurso didático. Neste sentido, o balanceamento entre a complexidade do projeto e os objetivos de aprendizagem deve nortear a escolha do projeto (ou dos projetos) de código aberto a ser(em) utilizado(s).

Em contrapartida, estudantes nos estudos EC2 e EC3 destacaram a experiência proporcionada ainda na Universidade de lidar com projetos próximos do que irão encontrar no mercado de trabalho, que não representam simplificações, que não foram criados especificamente para fins didáticos, ou seja, que não correspondem aos denominados *'toy projects'* que são usualmente empregados nas disciplinas de engenharia de software. Como mencionado por um dos estudantes do estudo EC2, esse fato representa uma "quebra de paradigma" (ST7). Segundo um estudante do estudo EC3, o paradigma atual de uso de projetos simplificados gera a preocupação de não saber lidar com projetos reais, uma vez que não tratam com esse tipo de projeto na graduação (SR1).

Em dois estudos (EC1 e EC2), similarmente ao que ocorre na indústria, os estudantes atestaram o envolvimento de usuários com o software. No estudo EC2, o estudante lembra que existe o *feedback* dos usuários, enquanto que no EC1, os estudantes lembram que o software já é utilizado por várias pessoas. De acordo com o professor, neste último estudo, constatar a utilidade do software é importante para aumentar o interesse dos estudantes. Este fato é ratificado no estudo de Santore et al. (2010), no qual os estudantes ficaram entusiasmados por implementar um projeto real e útil.

Outros estudos também apontam essas características como próximas ao mundo real. Ellis et al. (2007) destacam a diversidade e distribuição dos 'vários desenvolvedores participantes', a interação com os 'usuários envolvidos', a mudança constante dos requisitos, ter que lidar com código com 'documentação deficiente' e 'escrito por terceiros' e, por fim, ter que atender a padrões profissionais de qualidade. Já Buchta et al. (2006) ressaltam o 'tamanho do software' e a contribuição individual no contexto de uma equipe.

Proximidade das atividades realizadas nas empresas

Sobre a proximidade entre as atividades realizadas no OSP e as atividades que são executadas no mercado de trabalho, 56% dos estudantes no estudo EC1 admitiram esta proximidade. Ao longo das entrevistas, os estudantes também reconheceram a proximidade, contudo, salientaram que as modificações em si, solicitadas pelo professor, eram bastante simples, embora exigissem a execução de todas as atividades do processo de mudança incremental estudado. A investigação qualitativa nos estudos EC2 e EC3 também demonstrou o reconhecimento da proximidade das atividades solicitadas com as atividades executadas nas empresas: 'testar software desenvolvido por terceiros' e 'necessidade de engenharia reversa', respectivamente. Na literatura, o estudo de Buchta et al. (2006) realça que o uso de projetos de código aberto de porte médio possibilitou o envolvimento

dos estudantes com solicitações de mudanças reais no software.

A Tabela 9.2 resume as características indicadas pelos estudantes em cada um de nossos estudos, que aproximam as atividades executadas no OSP com as que são realizadas nas empresas. Comparando os resultados de cada estudo, as percepções mais comuns por parte dos estudantes são as condições de ter que 'lidar com software existente', 'aplicabilidade das práticas' e 'ter que lidar com software sem documentação'. As demais condições foram reconhecidas apenas em um estudo.

Tabela 9.2 Proximidade das atividades realizadas nas empresas.

EC1	**EC2**	**EC3**
-	Ter que lidar com projetos sem documentação.	Ter que lidar com projetos sem documentação.
-	Ter que lidar com projeto de código aberto.	-
-	Exige mais autonomia.	-
-	-	Necessidade de engenharia reversa.
Aplicação do conteúdo estudado*.	A aplicação de fato do conteúdo estudado.	Aplicabilidade das práticas.
-	Testar software desenvolvido por terceiros.	-
Dar continuidade a um projeto existente.	Lidar com software pronto.	Trabalhar com software existente.

(*) associado ao tema 'Aprendizagem', no estudo de caso.

Quanto à condição de ter que lidar com software existente, em todos os estudos, os estudantes distinguiram esta situação, em que nem sempre o profissional envolve-se com projetos iniciados do zero. Para Carrington (2003), o uso de projetos de código aberto provê a possibilidade trabalhar a engenharia reversa e manutenção de software, o que representa uma experiência mais realista para os estudantes do que apenas se concentrarem no desenvolvimento de um novo projeto.

Outra característica que comprova a proximidade das atividades que precisaram ser executadas no OSP com as que são realizadas no mercado de trabalho, foi o relato dos estudantes de que já estavam aplicando as mesmas práticas nos estágios que estavam realizando (EC1 e EC3) ou mesmo no exercício profissional (segundo relato de um ex-aluno que participou do estudo EC2).

Apesar de ter sido detectada em apenas um estudo, a característica da abordagem de exigir mais autonomia dos estudantes é bastante relevante. Como lembrado por um dos estudantes do estudo EC2, ele precisou desenvolver esse comportamento ao longo de suas experiências profissionais, mas seus colegas, mesmo não estando acostumados, tiveram que transpor esse desafio para realizar as atividades no projeto. De acordo com Budd (2009), diferentemente de um curso tradicional, onde tudo que é necessário foi explicado anteriormente ou existe uma explicação em algum livro texto, qualquer que seja o OSP utilizado, em virtude da diversidade de ferramentas utilizadas ou habilidades que lhe são

necessárias, o estudante sempre precisará descobrir o que precisa saber e onde pode obter aquela informação. Por conseguinte, a abordagem exige que o estudante desenvolva alguma autonomia.

Dificuldades provenientes de uso de projetos reais

Em contrapartida, a proximidade com a realidade acarreta em um conjunto de dificuldades que os estudantes precisam enfrentar. 100% dos estudantes nos estudos EC2 e EC3 concordaram que o uso do OSP permitiu que eles visualizassem a complexidade e as dificuldades existentes em um projeto real de software. A Tabela 9.3 descreve as dificuldades identificadas pelos estudantes ao realizar as atividades durante os estudos EC1, EC2 e EC3. Dentre as dificuldades comumente encontradas, temos: o 'impacto inicial', a 'dificuldade de realização das atividades', o desafio de 'entender código desenvolvido por terceiros' e a 'falta de documentação'. Particularidades deram-se apenas no estudo EC3 em virtude da 'novidade da abordagem' (tratar os requisitos a partir de um projeto existente) e, consequentemente da necessidade de 'entender todas as funcionalidades' do software, o que, especificamente no caso do JabRef, gerou alguma dificuldade, porque podem ser configuradas pelo usuário.

Tabela 9.3 Dificuldades ao ter que lidar com um OSP.

EC1	**EC2**	**EC3**
Inicialmente intimidado*	Dificuldade com projetos grandes.	Impacto inicial
	Não ter experiência com projetos complexos.	
	Como começar a trabalhar em um projeto grande.	
Dificuldade de realização das atividades.	Dificuldades na execução dos testes.	Dificuldades na execução das atividades.
-	-	Entender algumas funcionalidades.
-	Entender código desenvolvido por terceiros.	Entender código desenvolvido por terceiros.
-	Falta de documentação.	Falta de documentação.
-	-	Novidade da abordagem diferente.

(*) Apesar de não ter sido explicitamente declarada pelos estudantes, inferimos por outra evidência, como apresentada ao longo do texto.

A novidade de lidar com um software desconhecido de tamanho e complexidade razoáveis gera um impacto inicial nos estudantes, do mesmo modo que poderá ocorrer quando se defrontarem com situações semelhantes em uma empresa. Um estudante do estudo EC1 declarou o sentimento de intimidação ao se deparar com este tipo de situação. Ter esta experiência ainda na universidade serve de aprendizado sobre como proceder e provê confiança de que conseguirá realizar o que foi solicitado, como discutiremos posteri-

ormente neste capítulo. Contudo, é preciso mensurar o tamanho do desafio a ser proposto aos estudantes para que não se sintam desmotivados e aconteçam desistências, como nos estudos EC2 e EC3.

Quanto à dificuldade na execução das atividades, esta é uma dificuldade esperada, dada a ausência de simplificações quando softwares reais são usados. O tamanho, a complexidade, os problemas de estrutura no código, a documentação incompleta, entre outros, geram dificuldades para a localização e realização da mudança (EC1), definição do que testar e como testar (EC2) e construção de diagramas que retratem o código da aplicação (EC3). Entretanto, também refletem as possíveis dificuldades que poderão vir a enfrentar no mercado de trabalho.

A síntese das evidências (RS) ratifica essas dificuldades (Tabela 5.13). A dificuldade na 'compreensão do projeto', resultado da síntese, pode ser associada à dificuldade em 'entender código desenvolvido por terceiros'. Mais especificamente, podemos dizer que esta última representa uma das explicações para a dificuldade de compreensão do projeto. A 'falta de documentação', que representa outra explicação para a dificuldade de compreensão, também surgiu na síntese, neste caso descrita como 'documentação incompleta'. Por fim, a 'dificuldade na execução das atividades solicitadas' representou uma dificuldade parcialmente confirmada pela síntese.

Se por um lado as evidências encontradas mostram que tais dificuldades aparecem com certa frequência, salientamos que não são exclusividade de projetos de código aberto. Elas podem ocorrer qualquer que seja o projeto real utilizado, exatamente por serem consequência das características presentes nesse tipo de projeto (ver Tabela 9.1).

Visão sobre as habilidades necessárias

Apesar das dificuldades, a participação no projeto permitiu que 93% dos estudantes no estudo EC2 e 100% dos estudantes no estudo EC3 compreendessem as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software. A Tabela 9.4 detalha as habilidades necessárias indicadas pelos estudantes nos estudos EC2 e EC3. Cabe a ressalva de que não investigamos diretamente essa questão e nenhuma categoria aflorou a esse respeito dos dados no estudo EC1.

A Tabela 9.4 mostra que os estudantes perceberam a necessidade de habilidades específicas ligadas à área de conhecimento tratada, como: 'determinar o que deve ser testado e como deve ser testado', 'ter a vivência dos erros mais comuns' e 'pensar em todas as possibilidades ao testar o software', para o estudo EC2 que trabalhou a área de testes de software, e 'saber buscar os requisitos', para o estudo EC3 que trabalhou a área de requisitos de software. Os estudantes também reconheceram a necessidade de habilidades gerais como 'escrever código e documentação compreensível por terceiros' e 'saber trabalhar em equipe', reconhecidas nos dois estudos (EC2 e EC3) e outras reconhecidas em apenas em um dos estudos, a exemplo de 'adaptar-se aos padrões mesmo informais utilizados pela equipe' e 'saber identificar o problema e o que aplicar para resolvê-lo', pelos estudos EC2 e EC3, respectivamente. Uma observação é que os estudantes do estudo EC2 identificaram mais características que os estudantes do estudo EC3. Supomos que isso aconteceu dada a maior maturidade dos primeiros, uma vez que os estudantes já se encontravam no final do curso.

Tabela 9.4 Habilidades necessárias percebidas pelos estudantes após a participação no projeto.

EC2	**EC3**
Ler código e documentação feita por terceiros.	-
Escrever código e documentação compreensível por terceiros.	Saber documentar.
Adaptar-se aos padrões, mesmo informais, utilizados pela equipe.	-
Saber trabalhar em equipe.	Saber trabalhar em equipe.
Ter uma base de conhecimento teórico.	-
Ter noção das ferramentas existentes.	-
Saber pesquisar e filtrar as informações resultantes.	-
Determinar o que deve ser testado e como deve ser testado.	-
Ter a vivência dos erros mais comuns.	-
Pensar em todas as possibilidades ao testar o software.	-
-	Saber identificar o problema e o que aplicar para resolvê-lo.
-	Saber buscar os requisitos.

Com esses resultados, concluímos que os estudantes perceberam a execução das atividades no projeto de código aberto como uma experiência de mundo real vivenciada por eles (Ob7). Como consequência, os estudantes puderam enxergar as dificuldades envolvidas e reconhecer as habilidades que eles próprios precisarão desenvolver. Acreditamos que esta situação é relevante porque pode gerar a conscientização e, por conseguinte, mudanças de comportamento quanto à busca do desenvolvimento dessas habilidades ainda durante o curso.

9.2 CONEXÃO TEORIA E PRÁTICA (OB6)

Influenciados pelo trabalho de Nandigam, Gudivada e Hamou-Lhadj (2008), no qual os autores acreditavam que a visão prática com o uso de OSP permitiria aos estudantes entender os princípios básicos da engenharia de software não como conceitos meramente acadêmicos, mas como algo realmente necessário para a prática, resolvemos investigar, dentre as possíveis consequências para a aprendizagem, especificamente se a adoção de projetos de código aberto permitiria ao estudante fazer a conexão do que é estudado na teoria com a prática (Ob6).

Procurando entender o que acontece quando há a conexão entre a teoria e a prática com a utilização de projetos reais, identificamos nos relatos dos estudantes a importância da prática para a aprendizagem da teoria, a importância da teoria para a realização da prática e, consequentemente, que tanto a teoria quanto a prática são fundamentais. A Tabela 9.5 descreve estes achados. Vemos que as categorias detectadas são similares

nos três estudos realizados, de modo que, das 16 categorias, apenas cinco são particulares a determinado estudo de caso. Discorremos sobre os achados que julgamos mais importantes.

Nos três estudos, os estudantes apontaram que a prática com o projeto proporcionou um 'exemplo concreto' de aplicação da teoria, de maneira que o objeto de estudo deixou de ser apenas um relato do professor. Diminuiu a abstração e tornou-se algo real.

Possivelmente por prover um exemplo concreto, encontramos evidências nos estudos EC2 e EC3 de que a prática ajudou a 'entender os conceitos' estudados, situação também relatada por Hepting et al. (2008).

Nos estudos EC1 e EC3, os estudantes enfatizaram que, a partir da prática, conseguiram 'internalizar a teoria' e 'consolidar o conteúdo', respectivamente,. Expressando que através do projeto prático, conseguiram "ligar os pontos e preencher as lacunas"(SR8) ou que a execução do projeto propiciou que se "criasse uma caixinha e fixasse no conhecimento da pessoa, no cérebro'"(SR5), sem se darem conta, os estudantes ilustraram o processo de assimilação e acomodação do conhecimento em sua estrutura cognitiva, conforme mencionado no Capítulo 2.

Novamente nos três estudos, os estudantes reconheceram a relevância da prática com o projeto para a 'retenção' ou 'fixação' do conteúdo. Enquanto nos estudos EC1 e EC2, os estudantes assinalaram que realizar o projeto era mais significativo do que simplesmente estudar para uma prova, onde o conteúdo é esquecido em pouco tempo, no estudo EC3, os estudantes indicaram como fator relevante para a retenção a necessidade da 'participação ativa', distinta da atitude passiva em uma aula tradicional, onde o estudante só recebe informação e o conhecimento fica "vago em determinado tempo" (SR5). Um dos estudantes chegou a afirmar: "você aprende muito mais fazendo do que você vendo" (SR2). Budd (2009) acrescentam ainda que a participação ativa, em qualquer OSP, demandará do estudante que se torne autodidata, porque sempre precisará aprender alguma ferramenta ou desenvolver alguma habilidade.

A Figura 9.1 ilustra os relacionamentos entre as categorias discutidas e, consequentemente, como ocorre a contribuição da prática com projetos para a aprendizagem de determinado conteúdo. Sob a perspectiva da fundamentação teórica sobre a aprendizagem discutida no Capítulo 2, as evidências comprovam: (i) a relevância da 'experiência concreta' para a aprendizagem significativa; (ii) a 'codificação semântica do conteúdo para a memória de longo prazo', em virtude do entendimento do conteúdo, ou seja, da ocorrência do 'processo de assimilação e acomodação do conhecimento' na 'estrutura cognitiva' do aprendiz; (iii) a influência do 'conteúdo ser significativo' para a retenção na memória de longo prazo e, finalmente, (iv) a importância do 'engajamento cognitivo' para a aprendizagem, através das atividades sendo executadas.

Paralelamente, outras categorias capturadas demonstram que a prática com projetos reais possibilita o desenvolvimento de competências técnicas e profissionais.

Não há dúvidas de que a prática é essencial para o desenvolvimento de habilidades. Ninguém aprende a andar de bicicleta, sem andar de bicicleta. Ninguém pode dizer que sabe fazer um bolo, apenas porque tem a receita, sem nunca ter feito um bolo. Do mesmo modo, os estudantes dos estudos EC2 e EC1 reconheceram que 'a prática é essencial para aprendizagem de alguns assuntos' ou que 'a falta da prática leva a uma

Tabela 9.5 Resumo das percepções dos estudantes sobre a conexão entre a teoria e a prática nos três estudos de caso.

EC1	**EC2**	**EC3**
Importância da prática para a aprendizagem da teoria		
A prática permite que o estudante enxergue um exemplo concreto.	A prática concretiza a teoria.	A prática concretiza a teoria.
A prática fortifica o aprendizado da teoria.	-	-
A prática internaliza a teoria.	-	A prática consolida o conteúdo.
-	-	A prática leva à participação ativa do estudante.
A prática fixa a teoria.	A prática ajuda na retenção.	A prática ajuda na retenção.
A prática permite visualizar quando a teoria é aplicada.	-	A prática permite visualizar a aplicabilidade da teoria.
A falta de prática leva a uma aprendizagem parcial.	A prática é essencial para aprendizagem de alguns assuntos.	-
-	A prática ajuda a entender a teoria.	A prática ajuda a entender a teoria.
-	A prática facilita o aprendizado da teoria.	-
-	A prática permite aprender o como fazer.	A prática permite aprender o como fazer.
-	A prática permite enxergar o problema realmente.	A prática permite enxergar o problema realmente.
O projeto foi usado como exemplo nas aulas teóricas.	A prática possibilita a discussão.	A prática possibilita a discussão.
Importância da teoria para a realização da prática		
A teoria alicerça a prática.	A teoria é necessária para a prática.	A teoria é necessária para a prática.
A teoria sistematiza a prática.	-	-
-	A teoria permite planejar o que deve ser feito.	-
Teoria e prática são importantes		
Teoria e prática são complementares.	-	Teoria e prática são fundamentais.

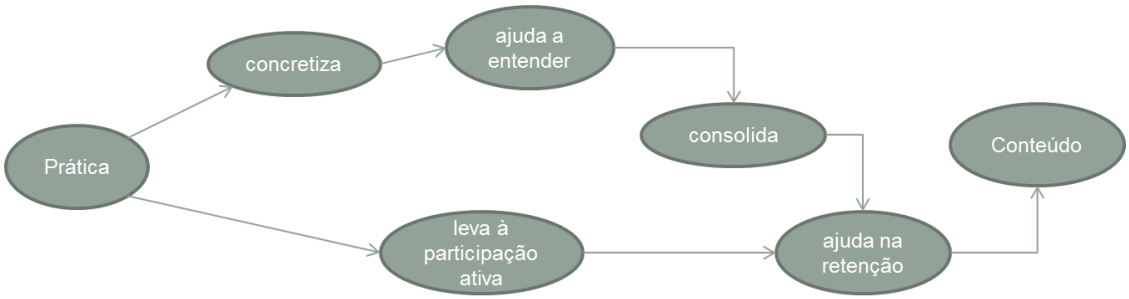

Figura 9.1 Contribuição da prática para o processo de aprendizagem.

aprendizagem parcial', respectivamente (rever Tabela 9.5). Ou seja, para 'aprender o como fazer' não basta saber a teoria sobre testes de software ou conhecer o processo de mudança incremental. É preciso exercitar as técnicas de teste e as atividades do processo de mudança, respectivamente. É exercitando que surgem as dúvidas. Como citamos no Capítulo 2, o aprendiz somente terá certeza se sabe ou não fazer, após experimentar (GENTRY, 1990). É trabalhando com projetos próximos à realidade, que os estudantes conseguem 'enxergar como é o problema realmente' (EC2 e EC3) e que nem sempre tudo é tão simples como parece, visualizando somente a teoria. Segundo Morelli et al. (2009), os estudantes veem que problemas desafiadores raramente rendem-se às soluções apresentadas em livros-texto. Na perspectiva de Horstmann (2009), os estudantes que não possuem experiência com projetos reais têm uma expectativa irreal com relação à qualidade do código fonte e ficam surpresos quando não encontram um código elegante como os apresentados nos livros-texto.

Por outro lado, a prática também permite que o estudante, de posse de um problema, consiga 'visualizar quando os conceitos e princípios teóricos devem ser aplicados' (EC1) ou propicia que os estudantes percebam que os conceitos e princípios estudados são realmente aplicados na prática, isto é, que confirmem 'a aplicabilidade da teoria' (EC3).

Exemplos concretos, uma visão mais realista dos problemas a serem enfrentados, o julgamento de quais técnicas devem ou não ser aplicadas, bem como a vivência de uma experiência (projeto) comum 'possibilitam a ocorrência de discussões'. Como mencionado anteriormente, a partir do projeto prático real, cada estudante tem a sua experiência e elabora o seu ponto de vista. Acrescentando-se que todos os estudantes experimentam um pouco do projeto, diferentes perspectivas podem ser expostas, favorecendo a discussão, sem contar que as discussões passam a ocorrer sobre exemplos concretos e não sobre exemplos abstratos, não vivenciados. Enquanto que no estudo EC1, o OSP foi utilizado algumas vezes como exemplo na discussão de tópicos teóricos, nos estudos EC2 e EC3, aulas específicas foram dedicadas à discussão sobre as atividades realizadas no projeto. Nestes estudos, vários relatos expõem a satisfação com a aprendizagem ocorrida em função dessas discussões.

Ilustramos a relação entre essas categorias na Figura 9.2. Aprender o como fazer leva ao desenvolvimento de habilidades técnicas, como fazer a engenharia reversa, testar ou modificar sistematicamente um software. Defrontar-se com um problema real pode

desenvolver a proatividade e a criatividade na procura por soluções alternativas. O contexto do software real propicia que os estudantes constatem os problemas como realmente são e possibilita que os estudantes visualizem a aplicabilidade da teoria, gerando reflexão e desenvolvimento do pensamento crítico. As discussões sobre o projeto permitem que os estudantes exponham suas opiniões e ampliem sua visão a partir da opinião de seus colegas. Assim, diversas habilidades ligadas à prática profissional podem ser desenvolvidas. Os resultados obtidos com relação ao desenvolvimento destas competências nos três estudos de caso serão comparados em 9.3.

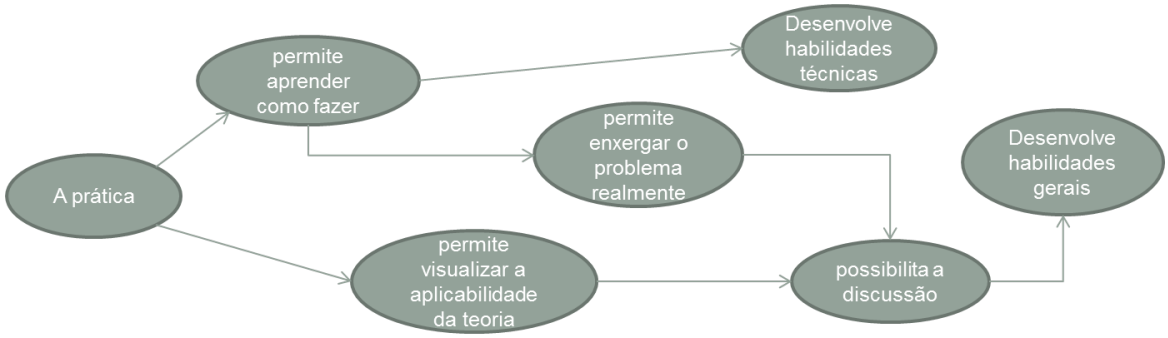

Figura 9.2 Contribuição da prática com projetos reais para o desenvolvimento de competências.

No que concerne à importância da teoria para a prática (Tabela 9.5), as categorias detectadas revelam que a 'teoria é necessária para a prática' (EC2 e EC3), porque 'alicerça' (EC1), permite ao estudante compreender o que está fazendo e porque está fazendo; porque 'sistematiza' (EC1), estabelece técnicas, métodos ou regras, possibilitando que a atividade seja mais produtiva e eficiente e, finalmente, porque permite 'planejar a prática' (EC2), visto que, 'o que' e 'o como deve ser feito' já são conhecidos. A Figura 9.3 expõe a nossa interpretação para o relacionamento entre essas categorias.

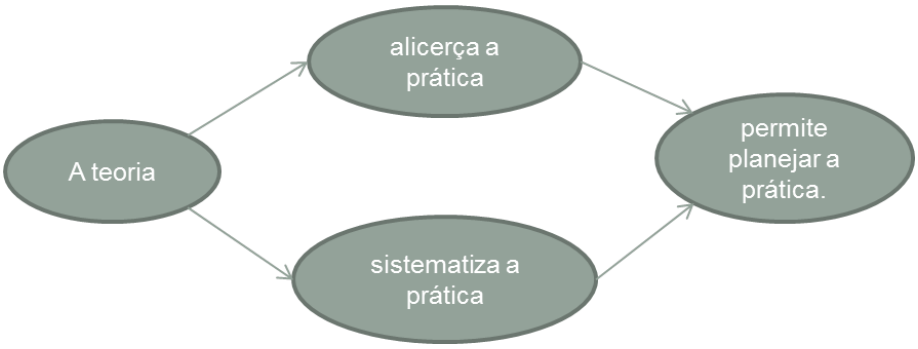

Figura 9.3 Importância da teoria para a prática.

Finalmente os estudantes reconheceram que teoria e prática são 'complementares' (EC1) e, por conseguinte, ambas são 'fundamentais' (EC3).

Do ponto de vista quantitativo, 93% dos estudantes no estudo EC2 concordaram que as atividades realizadas mostraram-lhes como a prática em projetos reais pode ser importante para aprender sobre testes de software. Enquanto que no estudo EC3, 100% concordaram com esta condição, porém, em relação a requisitos de software. Corroborando com esses achados, na síntese das evidências (RS) (Tabela 5.11), detectamos que 'o uso de projetos reais permite colocar em prática toda a teoria estudada'.

No que diz respeito à compreensão de como o conhecimento teórico é aplicado na prática em um projeto real, 86% dos estudantes concordaram que o uso do OSP propiciou esta compreensão no estudo EC2 e 100%, no estudo EC3. Já na síntese das evidências (RS) (Tabela 5.11) observamos que 'com a prática em projetos reais, os estudantes conseguem enxergar as razões de aplicar a teoria'.

Ellis et al. (2007) complementam que os preceitos e princípios da engenharia de software são incorporados ao projeto e tornam-se evidentes para os estudantes à medida que o projeto é executado, isto é, os princípios de engenharia de software são transmitidos pela experiência e não pela voz do instrutor. Para Nandigam, Gudivada e Hamou-Lhadj (2008), as atividades no OSP proveem um embasamento sólido para a disciplina de engenharia de software. Os autores destacaram que, após as atividades no projeto, as aulas expositivas e discussões sobre métricas de *design*, manutenibilidade do código, documentação e cuidado com a sincronização entre código e *design* tornaram-se mais significativas para os estudantes, inclusive com a articulação de pontos de vista próprios.

Isto posto, concluímos que: (i) a prática com a utilização de projetos abertos pode proporcionar uma aprendizagem mais significativa e, consequentemente, mais efetiva (melhorando a retenção) sobre o conteúdo apresentado; (ii) que os estudantes podem fazer a conexão da teoria com a prática e, assim, reconhecer a importância dos princípios, métodos e técnicas propostos pela teoria (Ob6); (iii) desenvolver habilidades técnicas e, por fim, (iv) desenvolver habilidades ligadas à prática profissional.

Nas próximas seções, discutiremos os resultados obtidos na realização desta pesquisa em relação ao desenvolvimento das habilidades técnicas e profissionais, e à conscientização da importância dos princípios, métodos e técnicas propostos pela engenharia de software.

9.3 OBJETIVOS DE APRENDIZAGEM ALCANÇADOS (OB4)

A fim de avaliarmos a eficiência da abordagem de uso de projetos de código aberto como exemplo de projeto de software a ser trabalhado pelos estudantes, propusemos investigar se certos objetivos de aprendizagem poderiam ser alcançados. Todavia, ao invés de utilizarmos como parâmetro de investigação os objetivos de aprendizagem específicos das disciplinas onde os estudos seriam realizados, para proporcionarmos uma visão mais abrangente e padronizada, preferimos utilizar como base o SWEBOK e os currículos de referência dos cursos de computação. Para o alinhamento com as necessidades da indústria de software, acrescentamos as deficiências apresentadas pelos estudantes recém-formados ao serem contratados, identificadas por estudos específicos. Assim, após a análise dos trabalhos relacionados e discussão com outro professor experiente na área de engenharia de software, estabelecemos um conjunto de objetivos que poderiam ser tratados por meio da abordagem (ver Capítulo 3).

Outros estudos também procuraram apurar os benefícios de utilização de projetos de código aberto com relação aos currículos de referência, SWEBOK e deficiências de formação apontadas pela indústria. Como o foco do estudo de Rekha et al. (2009) era o curso de ciência da computação, os autores utilizaram algumas habilidades estipuladas pelo currículo de referência desse curso, bem como um conjunto de habilidades esperadas pela indústria para os estudantes recém graduados, para direcionar sua investigação. Já Stamelos (2009), examinou as oportunidades de experiência prática que projetos de código aberto proveem para a educação em engenharia de software. Além de descrever como a utilização desses projetos poderia atender aos requisitos gerais comuns aos currículos de referência de computação propostos pela ACM e IEEE, o autor discutiu, para cada área de conhecimento do SWEBOK, como os OSP poderiam ser empregados, sugerindo inclusive um conjunto de atividades a serem desenvolvidas pelos estudantes e como estes poderiam participar das comunidades. O estudo ainda descreveu dois exemplos de disciplinas onde a abordagem foi aplicada, contudo não apresentou resultados quanto à aprendizagem dos estudantes.

A diferença principal da nossa pesquisa para esses estudos está no nível de detalhe e abrangência que buscamos perscrutar os objetivos de aprendizagem específicos da engenharia de software. Como descrito no Capítulo 3, a partir das visões da academia, da indústria e dos trabalhos relacionados, estabelecemos os objetivos de aprendizagem que poderiam ser tratados pela abordagem e, posteriormente, investigamos o alcance destes objetivos, um a um, dentro do escopo de cada estudo de caso que realizamos. Finalmente, acrescentamos, como consequência da síntese das evidências (RS), os objetivos alcançados por outros estudos. A seguir discutimos estes achados.

Objetivos de aprendizagem alcançados específicos da engenharia de software
Aplicamos a abordagem para a aprendizagem de áreas de conhecimento distintas da engenharia de software. Para cada uma delas, conseguimos alcançar vários objetivos de aprendizagem relacionados e alguns objetivos de outras áreas da própria engenharia de software, como apresentamos nas Tabelas 6.2, 7.8 e 8.6.

Vimos que os estudantes estudando sobre requisitos aprenderam sobre manutenção de software; estudando sobre testes, aprenderam sobre refatoração, uso de ferramentas de construção e gerenciamento de configuração de software; e, por fim, estudando sobre evolução de software, aperfeiçoaram-se no uso de ferramentas de construção e gerenciamento de configuração de software, bem como na execução de refatorações. Por conseguinte, a abordagem permite ampliar o escopo da aprendizagem integrando diversas áreas da engenharia de software, além da aprendizagem dos objetivos específicos.

Quanto aos objetivos de aprendizagem específicos, muitos dos que alcançamos são ligados às habilidades de 'saber fazer'. Os estudantes desenvolveram a habilidade de fazer uma mudança em um software de médio porte de modo sistemático, inclusive analisando o impacto desta mudança. Os estudantes desenvolveram a habilidade de criar testes automatizados para um software de médio porte, enfrentando os desafios que surgiram para a sua implementação. Desenvolveram a habilidade de usar ferramentas, tanto para criar estes testes, quanto para analisar a sua cobertura.

Particularmente no estudo de caso que aplicamos um projeto de código aberto para a

aprendizagem de requisitos de software, apesar de os estudantes reconhecerem a contribuição da abordagem, alcançamos apenas parcialmente os objetivos associados ao 'saber fazer', com exceção da capacidade de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software (LO1). Este fato pode assinalar que o uso de OSP pode não ser adequado em algumas situações. Porém, preferimos não chegar a conclusões precipitadas, uma vez que a proposta didática empregada foi totalmente nova e diferente do usual, inclusive para nós que a propomos. Ademais, a análise qualitativa apontou a necessidade de mais exercícios relacionados à modelagem dos requisitos, independente do uso ou não de projetos de código aberto. Este indício sugere a complementaridade da abordagem com uma didática convencional, onde exercícios de complexidade crescente ainda são necessários. Assim, acreditamos que a proposta didática como um todo deva ser primeiramente aperfeiçoada e novamente investigada, para que depois possamos chegar a alguma conclusão.

No que diz respeito aos objetivos relacionados às competências em nível de 'entendimento' (taxonomia de Bloom) e 'familiaridade' (CS2013), no estudo de caso EC1 (Evolução e Manutenção de Software), a abordagem permitiu que os estudantes fossem capazes de 'identificar as questões principais associadas com a evolução de software e explicar seu impacto no ciclo de vida do software' (LO113) e de 'descrever o processo de analisar e implementar mudanças no código de um projeto grande' (LO69). No estudo EC2 (Testes de Software), a abordagem contribuiu para a capacidade de: (i) 'descrever os conceitos e práticas relacionadas ao teste de software (LO90); (ii) 'descrever e distinguir entre tipos e níveis diferentes de testes' (LO95); (iii) 'entender a diferença entre testar pequenos programas e testar software de grande porte' (LO91); (iv) 'descrever como é um processo de gerenciamento de defeitos (LO105) e (v) 'ter algum conhecimento sobre ferramentas de integração contínua e testes de regressão' (LO110). E somente parcialmente para que os estudantes fossem capazes de: (i) 'descrever e distinguir entre as diferentes técnicas caixa-preta e caixa-branca para elaboração de casos de teste' (LO98a e LO98b); (ii) 'descrever o processo de testes de regressão e seu papel no gerenciamento de releases' (LO287) e (iii) 'descrever como selecionar bons testes de regressão e automatizá-los' (LO103). Finalmente, no estudo de caso EC3 (requisitos de software), o uso do projeto de código aberto contribuiu para os estudantes fossem capazes de: (i) 'descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais' (LO2); (ii) 'descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos' (LO8) e (iii) 'diferenciar o rastreamento de construção e recuperação e explicar seus papeis no processo de validação dos requisitos' (LO7). Por último, contribuiu apenas parcialmente para a capacidade de 'comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma' (LO4).

Considerando a síntese das evidências (RS), objetivos de aprendizagem de outras áreas como processos de software, qualidade de software, *design* de software e construção de software também foram alcançados.

Analisando cada um dos objetivos retratados na Tabela 5.19, verificamos que, dentre os objetivos especificamente ligados à engenharia de software, dez estão relacionados à competência de 'saber fazer'. Destes, sete foram alcançados e três foram parcialmente

alcançados. Entre os alcançados, encontramos habilidades fundamentais para o processo de desenvolvimento e manutenção de software, como modelagem (LO6 e LO38), programação (LO63), testes (LO66a), engenharia reversa (LO124), manutenção (LO118) e uso de ferramentas de modelagem (LO47). Entre os parcialmente alcançados temos as habilidades de depurar código (LO66b), corrigir *bugs* (LO100) e usar ambientes de desenvolvimento modernos (LO82), neste caso considerado parcial, porque não conseguimos capturar, a partir da evidência relatada, se os estudantes foram capazes de utilizar o ambiente em sua plenitude, inclusive as funcionalidades de testes e depuração disponibilizadas.

Quanto aos objetivos relacionados às competências em nível de 'entendimento' (taxonomia de Bloom) e 'familiaridade' (CS2013), a síntese identificou o alcance também de sete objetivos (Tabela 5.19), entre eles: (i) 'descrever os passos necessários para planejar e dar andamento em um projeto de software em um ambiente real' (LO157); (ii) 'enfatizar a importância do design do software' (LO52); (iii) 'descrever como desenvolver um software é diferente de um esforço individual de um programa' (LO156); (iv) 'reconhecer atributos de qualidade de software' (LO177); (v) 'perceber a importância da documentação para a manutenção de software' (LO114); (vi) 'entender as nuances da programação e os desafios envolvidos em escrever código limpo' (LO115) e (vii) 'compreender a importância da manutenção e evolução em um projeto de software, especificamente considerando o tamanho do esforço necessário nessas atividades' (LO117).

Comparando os resultados da síntese das evidências (RS) com os de nossos estudos, apuramos que o alcance de alguns objetivos de aprendizagem se repetiu, a exemplo de: (i) 'aplicar engenharia reversa em um software de médio porte' (LO124), em RS e EC3; (ii) 'realizar a manutenção de um software real de médio porte' (LO118), em RS e EC1 e (iv) 'perceber a importância da documentação para a manutenção de software' (LO114), em RS e EC3.

Podemos dizer que o alcance foi complementar para os objetivos que tratam das capacidades de: (i) 'aplicar várias estratégias de testes em programas simples' (LO66a) em RS e 'criar e documentar um conjunto de testes para um segmento de código de médio porte' (LO101) em EC2; (ii) 'modelar e especificar os requisitos de software de médio porte, usando métodos comuns' (LO6), alcançado em RS e parcialmente alcançado para EC3 e, por fim, (iii) 'construir, executar e depurar programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração visual', parcialmente alcançado em RS e alcançado em EC1 e EC2.

Todos esses achados confirmam a contribuição do uso de projetos de código aberto para o alcance de objetivos de aprendizagem específicos da engenharia de software, principalmente considerando o desenvolvimento de habilidades importantes para o exercício profissional, a exemplo de modelagem, programação, testes, engenharia reversa e modificação de software existente. Mostram também a abrangência da abordagem, dados os resultados positivos para diversas áreas da engenharia de software. Ademais, diferenças entre os resultados sugerem que os achados não podem ser generalizados, em virtude da diversidade do contexto: estudantes, professores, abordagem didática utilizada, programa da disciplina e OSP escolhido.

A seguir faremos uma comparação semelhante para os objetivos de aprendizagem re-

Tabela 9.6: Competências da prática profissional alcançadas.

Id.	Objetivo de aprendizagem	EC1	EC2	EC3
LO233	Aceitar e responder críticas de terceiros.	-	Sim	Sim
LO235	Avaliar o trabalho desenvolvido por terceiros.	-	Sim	Sim
LO268	Ter experiência com código desenvolvido por terceiros.	Sim	Sim	Sim
LO236	Habilidade de ser proativo.	-	Sim	Parcial
LO237	Habilidade de ser criativo.	-	Sim	Não
LO266	Ser capaz de solucionar problemas.	-	Sim	Parcial
LO267	Gerar soluções alternativas para os problemas.	-	Sim	Parcial
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.	-	Sim	Sim
LO251a	Desenvolver a habilidade de compreender material técnico (código fonte e documentação).	Sim	Sim	Sim
LO251b	Desenvolver a habilidade de sumarizar material técnico (código fonte e documentação).	-	Sim	Parcial
LO246	Escrever documentação, clara, concisa e acurada.	-	Parcial	Parcial
LO227	Habilidade de trabalhar efetivamente como membro de equipes.	Não	Sim	Sim
LO242	Habilidade de lidar com incertezas e ambiguidades.	-	Sim	-
LO250	Usar ferramentas de comunicação.	-	Sim	-
LO270	Compreender como teoria e prática influenciam uma a outra.	Sim	Sim	Sim
LO232	Expressar opiniões próprias.	-	Sim	Sim
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.	-	Sim	Sim
LO203	Compreender as expectativas de trabalho (habilidades esperadas).	-	Sim	Sim
LO269	Ter vivenciado pelo menos um projeto real.	Sim	Sim	Sim
LO273	Ser capaz de pensar o problema em vários níveis de detalhe e abstração.	-	-	Sim

lacionados à prática profissional.

Objetivos de aprendizagem alcançados relacionados à prática profissional

A Tabela 9.6 resume o alcance das competências relacionadas à prática profissional considerando os resultados obtidos nos estudos de caso sobre evolução de software (EC1), testes de software (EC2) e requisitos de software (EC3), apresentados nas Tabelas 6.2, 7.11 e 8.9, respectivamente.

Através da Tabela 9.6 observamos que várias competências relacionadas à prática profissional foram direta ou indiretamente desenvolvidas por meio da abordagem. Por corresponder a um software real, ser desenvolvido por diversas pessoas, os estudantes 'vivenciaram um projeto real' (LO269), tiveram que 'lidar com código desenvolvido por terceiros' (LO268) e desenvolver a habilidade de 'compreender material técnico' (código fonte e documentação) (LO251a). Consequentemente, o desenvolvimento destas competências está implicitamente ligado ao emprego do OSP. Semelhantemente, como o uso de projetos reais favorece que os estudantes 'compreendam como teoria e prática influenciam uma a outra' (LO270), conforme discutimos em 9.2, a utilização de OSP favorece o alcance deste objetivo. Analisando a tabela, verificamos que realmente esses objetivos foram alcançados nos três estudos de caso. Ocorrências ainda solidificadas na síntese das evidências (RS) (ver Tabela 5.19). Ademais, o uso de projetos reais e a experiência provinda das dificuldades (discutidas em 9.1) promoveram ainda a 'compreensão das expectativas de trabalho (habilidades esperadas)' (LO203), em EC2 e EC3. Este objetivo não foi investigado no estudo EC1.

Em contrapartida, existem outros objetivos para os quais não podemos prever que, ocorrendo determinada condição, o objetivo poderá ou não ser alcançado. Por exemplo, no estudo EC3, o OSP, por representar um software completo, proporcionou o desenvolvimento da 'capacidade de começar a pensar o problema em vários níveis de detalhe e abstração' (LO273), mas isso não quer dizer que há a possibilidade de este resultado ocorrer em outros casos. As características de corresponder a um software já existente, completo, sendo utilizado por vários usuários e que, consequentemente, exige a utilização de ambiente de produção, propiciaram o 'alcance de uma visão holística do desenvolvimento do produto de software' (LO272) nos estudos EC2 e EC3, mas que pode não ocorrer em outros estudos.

Em alguns casos, como o uso de projetos não simplificados possibilita que os estudantes enxerguem o problema como ele é realmente (Seção 9.2), algumas competências podem ser exigidas e, portanto, desenvolvidas como, por exemplo: (i) 'lidar com incertezas e ambiguidades' (LO242) em EC2; (ii) 'habilidade de ser proativo' (LO236), em EC2 e parcialmente em EC3; (iii) 'habilidade de ser criativo'(LO237), em EC2; (iv) 'ser capaz de solucionar problemas' (LO266), em EC2, RS e parcialmente em EC3 e, por fim, (v) 'gerar soluções alternativas para os problemas (LO267), em EC2, RS e parcialmente em EC3.

Observamos que em outras situações, o desenvolvimento das habilidades depende também das atividades solicitadas e a forma que serão exigidas por parte do professor. Por exemplo, se as atividades solicitadas demandarem que os estudantes façam algum tipo de documentação, podemos obter um melhor alcance para o objetivo 'desenvolver a habilidade de sumarizar material técnico (código fonte e documentação)' (LO251b) alcançado em EC2, mas somente parcialmente alcançado em EC3, bem como para o objetivo 'escrever documentação, clara, concisa e acurada' (LO246), alcançado em RS e parcialmente alcançado nos estudos EC2 e EC3. A forma que o professor conduz discussões em sala de aula sobre as atividades executadas e os resultados dessas atividades, como abordamos em 9.2, também pode favorecer o alcance dos objetivos 'expressar opiniões próprias'(LO232), 'aceitar e responder críticas de terceiros' (LO233) e 'demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias (LO241), todos alcançados nos estudos EC2 e EC3.

Uma situação particularmente dependente do que for estabelecido pelo professor refere-se ao desenvolvimento da habilidade de trabalhar efetivamente como membro de equipes (LO227). O professor pode determinar que os estudantes trabalhem individualmente ou em equipe, junto a um mesmo OSP. O professor pode estabelecer que as atividades resultem em contribuições para a comunidade do projeto ou não. Consequentemente, para os casos onde o propósito é contribuir para comunidade, a interação do estudante com os desenvolvedores da comunidade torna-se essencial. Outro fator importante que influencia o trabalho da equipe corresponde à existência de alguma forma de avaliação individualizada ou não. Nos estudos de casos que realizamos, os estudantes deveriam trabalhar em equipe, contudo, suas atividades não necessariamente gerariam contribuições para a comunidade. Somente no estudo EC2, existiu a avaliação individualizada para algumas das atividades. Neste contexto, conseguimos o alcance do objetivo LO227 nos estudos EC2 e EC3, todavia, a falta de avaliação individualizada influenciou

fortemente o estudo EC1, de maneira que não conseguimos alcançar este objetivo. Por outro lado, os resultados da síntese das evidências (RS) demonstraram que os estudantes foram capazes de trabalhar como equipe quando esta se resumia aos próprios colegas. Porém, quando precisaram trabalhar com profissionais externos, os resultados nem sempre foram interessantes devido à falta de receptividade de algumas comunidades. Assim, o objetivo foi considerado como parcialmente alcançado.

A síntese das evidências (RS) capturou ainda alguns resultados para os objetivos (Tabela 5.19): (i) 'interagir com outras pessoas (interagir profissionalmente com os *stakeholders*)' (LO243), parcialmente alcançado; (ii) 'distinguir comportamento profissional ético e não ético (LO199), parcialmente alcançado, por fim, (iii) 'coordenar o próprio trabalho com relação ao trabalho dos outros' (LO234), alcançado. De acordo com as evidências reportadas, como estes objetivos estão associados ao trabalho em equipe, supomos que a razão para que os dois primeiros objetivos tenham sido apenas parcialmente alcançados seja também a falta de receptividade de algumas comunidades.

Para os estudos de caso produtos desta pesquisa, cabe a observação que muitas destas competências relacionadas à prática profissional não foram diretamente perscrutadas, mas que afloraram da análise qualitativa realizada, particularmente considerando o estudo EC1, realizado ainda na fase exploratória da pesquisa. Contudo, ressaltamos que dada à relevância dessas competências para o exercício profissional e os resultados positivos obtidos quanto à contribuição de projetos de código aberto para o seu desenvolvimento, seria interessante buscar métodos de investigação mais apropriados para a sua avaliação, principalmente no que se refere à coleta de dados. Um exemplo seria executar o registro das observações das discussões realizadas em sala, para avaliar o desenvolvimento da 'habilidade de expressar opiniões próprias' (LO232) e do 'desenvolvimento do pensamento crítico e capacidade de gerar e sintetizar novas ideias' (LO241). Ao mesmo tempo, reconhecemos que este tipo de investigação não é uma tarefa fácil.

Todos esses achados ratificam a contribuição do uso de projetos reais, especificamente de projetos de código aberto, para o desenvolvimento de habilidades técnicas e gerais, conforme discutimos em 9.2 e retratamos na Figura 9.2. No que diz respeito ao desenvolvimento de habilidades técnicas, apesar dos resultados positivos alcançados, concluímos que a abordagem não descarta a execução de outros exercícios. O ideal seria aplicar um conjunto de exercícios de complexidade crescente até o complemento com as atividades no OSP. Quanto ao desenvolvimento de habilidades gerais, vimos que a abordagem de uso de OSP exige o desenvolvimento de algumas competências. Em outros casos, por representar situações reais, algumas condições podem ocorrer e assim desenvolver algumas competências. Por último, o desenvolvimento de algumas competências está sujeito às atividades solicitadas e às exigências de como estas atividades devem ser realizadas, estabelecidas pelo professor.

A seguir discutiremos as consequências da abordagem para a percepção do estudante quanto à preparação para o mercado de trabalho.

Tabela 9.7 Contribuição do OSP para preparação para o mercado capturada em cada estudo de caso.

EC1	**EC2**	**EC3**
Experiência com um projeto de porte razoável e mais complexo. Vivência da complexidade das atividades.	Vivência de um projeto de software.	Vivência de um projeto.
-	Visão profissional.	Visão profissional.
Capacitação proporcionada.	Capacitação proporcionada.	Capacitação proporcionada.
Desenvolvimento da autoconfiança.	Desenvolvimento da autoconfiança.	Desenvolvimento da autoconfiança.
Preparação satisfatória.	Preparação parcial.	Preparação inicial.

9.4 SENTIMENTO DE PREPARAÇÃO PARA O MERCADO DE TRABALHO (OB8)

Outro ponto que buscamos investigar como consequência da utilização da abordagem foi se a experiência proporcionada com um projeto real, especificamente um projeto de código aberto, contribui para que o estudante sinta-se mais preparado para o mercado de trabalho. Como resultado, capturamos além da própria percepção do estudante sobre sentir-se ou não preparado, um conjunto de circunstâncias provenientes da experiência profissional propiciada pelo projeto, que, a nosso ver, colabora para essa preparação. Conforme podemos constatar na Tabela 9.7, os resultados são recorrentes entre os estudos. A seguir discorreremos sobre cada uma das circunstâncias encontradas.

Vivência de um projeto de software

Segundo Ellis, Morelli e Hislop (2008b), estudantes tipicamente não apresentam o conhecimento e as habilidades necessárias para lidar com projetos grandes. Papadopoulos, Stamelos e Meiszner (2012) destacaram que trabalhar com o OSP, para muitos estudantes, representou a primeira vez que lhes foi solicitado que laborassem em um contexto real, envolvendo uma comunidade grande. Essa situação foi comprovada em nossos estudos. No estudo EC3 envolvendo estudantes que se encontravam na metade do curso, para a maioria deles, foi a primeira vez que vivenciaram um projeto real. Todavia, no estudo EC1, apesar de muitos estudantes já estarem próximos de concluírem o curso, a maioria afirmou que o OSP utilizado foi o maior projeto que eles manipularam até aquele momento. Ademais, no estudo EC2, onde os estudantes estavam prestes a se formar, sete dos quinze estudantes que participaram da pesquisa também não tinham tido nenhuma experiência com um projeto real anteriormente. Isto quer dizer que é comum o estudante concluir o curso e não ter vivenciado ao menos um projeto de software em um contexto real. Como vimos no Capítulo 3, esta situação extrapola nossos estudos, uma vez que a 'falta de experiência com projetos reais' é uma das deficiências apontadas pela indústria de software para os estudantes recém-formados (Tabela 3.5). Portanto, como a abordagem de uso de OSP propicia esta experiência, ela colabora para a preparação para o mercado. De acordo com Ellis et al. (2011), projetos de código aberto permitem que

os estudantes ganhem experiência de mundo real e em ambientes profissionais. Usualmente os projetos são razoavelmente grandes, provendo aos estudantes a compreensão da complexidade envolvida no desenvolvimento de um software real, que tem continuidade além do contexto acadêmico (ELLIS; MORELLI; HISLOP, 2008a). Em nossos resultados, verificamos que tanto no estudo EC1 quanto no EC2, os estudantes reconheceram a relevância de vivenciar as dificuldades provenientes de lidar com um projeto grande ainda na universidade, diferentemente do que ocorreu ao longo das demais disciplinas (EC1). Os estudantes passaram a ter uma visão de como as coisas ocorrem fora da academia (EC2) ou da complexidade que precisarão enfrentar futuramente (EC1). E, consequentemente, da importância da experiência para a formação profissional (EC3) e para a preparação para o mercado de trabalho (EC2). Como resultado da síntese das evidências (RS), os estudantes confirmaram a experiência de mundo real proporcionada (ver Tabela 5.10).

Visão profissional

Como consequência da vivência de um projeto de software, os estudantes ganham uma visão mais profissional da construção de software. Para Ellis et al. (2012), ter uma visão mais ampla do processo de desenvolvimento de software, é um resultado natural após estudantes trabalharem em um projeto HFOSS. De acordo com Nandigam, Gudivada e Hamou-Lhadj (2008), os estudantes passam a enxergar a diferença entre escrever um programa e desenvolver um projeto de software. Eles são capazes de descrever o impacto do tamanho e da complexidade nos métodos e técnicas empregadas para desenvolver o software (ELLIS et al., 2012).

No estudo EC3, os estudantes conseguiram enxergar que um projeto de software pode não ser tão simples quanto eles esperavam, em virtude dos projetos pequenos com os quais estavam acostumados a trabalhar. Vislumbraram que o desenvolvimento de software exige um esforço razoável. Um dos estudantes chegou a declarar que estava mais maduro porque, depois do projeto, já sabia o que poderia encontrar futuramente. Estudantes do estudo EC2 também admitiram que trabalhar com um projeto grande possibilitou a visão do que será realmente encontrado no mercado. Proporcionou uma visão mais profissional, porque introduziu algo próximo da realidade, diferentemente das outras disciplinas do curso. Um dos estudantes no estudo EC2 ainda revelou que, através do projeto, pôde visualizar outros aspectos do processo final de entrega do software. Complementando esta visão profissional, lembramos ainda a percepção proporcionada sobre as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software, como vimos em 9.1. Entre os resultados capturados na síntese das evidências (RS), a visão profissional foi provida pelos diversos papéis assumidos pelos estudantes e pela percepção de que o próprio trabalho encaixava-se em um contexto maior (ver Tabela 5.10).

Capacitação proporcionada

De modo geral, ao longo dos estudos de caso, observamos que os estudantes sentiram-se mais capacitados com relação: (i) ao processo sistemático de manutenção e evolução de software (EC1); (ii) saber o que testar (EC2); (iii) usar ferramentas (EC2); (iv) trabalhar em equipe (EC2) e (v) trabalhar requisitos a partir de um software existente (EC3).

Quantativamente, 93,3% dos estudantes no estudo EC2 e 100%, no estudo EC3, con-

cordaram que a experiência com a realização das atividades no projeto vai contribuir para a melhoria de seu desempenho futuramente na vida profissional[2], o que significa que a grande maioria dos estudantes reconheceu a contribuição das atividades realizadas no OSP para a sua capacitação para o mercado. Resultados da síntese das evidências (RS) destacaram esta capacitação (ver Tabela 5.10), principalmente no que diz respeito a ajudar a conseguir um emprego. Alguns autores interpretaram que o projeto propiciou maior profissionalismo (HISLOP; ELLIS; MORELLI, 2009) e maturidade (ELLIS et al., 2007) aos estudantes, ou mesmo que aprendessem a trabalhar em um ambiente de desenvolvimento profissional (ELLIS et al., 2012).

Desenvolvimento da autoconfiança

Colaborando também para o sentimento de preparação, investigamos se os estudantes sentiam-se confiantes em realizar atividades semelhantes em outro projeto. 80% e 86,67% dos estudantes nos estudos EC2 e EC3, respectivamente, concordaram que estavam confiantes[3]. Por meio das entrevistas, confirmamos o desenvolvimento da autoconfiança dos estudantes nos três estudos realizados, do mesmo modo que relatado por Rekha et al. (2009) e obtido na síntese das evidências (RS) (Tabela 5.10).

No estudo de Horstmann (2009), a maioria dos estudantes, antes de lidar com o OSP, nunca havia se defrontado com um projeto grande, de maneira que se sentiram inicialmente intimidados. Todavia, ao final da experiência, perderam o medo e seu nível de confiança aumentou bastante. Este fato foi também corroborado em nossos estudos. No estudo EC1, estudantes relataram o sentimento de insegurança inicial, que foi substituído pelo sentimento de capacidade e realização, já que conseguiram realizar as atividades solicitadas. No estudo EC2, os estudantes perceberam que são capazes de lidar com código razoavelmente grande, desenvolvido por profissionais experientes; já sabem por onde começar, o que devem fazer e que ferramentas podem utilizar; finalmente, que efetivamente terem testado um software deu-lhes maior segurança para o mercado de trabalho. Por último, no estudo EC3, os estudantes declararam estar mais seguros e confiantes para o mercado de trabalho.

Sentir-se preparado

Todavia, apesar de todas as circunstâncias mencionadas que colaboram para a preparação para o mercado de trabalho, ao questionarmos diretamente se as atividades realizadas no projeto contribuíram para que o estudante se sentisse mais preparado tecnicamente para o mercado de trabalho, apenas 53,3% dos estudantes no estudo EC2 concordaram, enquanto que no estudo EC3, foram 93,3%. A nossa percepção, ao longo das entrevistas, foi que os estudantes sentiram-se com uma preparação satisfatória no estudo EC1, parcial, no estudo EC2 e, por fim, inicial, no EC3. Como resultado da síntese das evidências (RS), no estudo de Ellis et al. (2007), os estudantes não reconheceram a experiência proporcionada pelo projeto sobre o assunto (ver Tabela 5.10).

Supomos que, como o emprego do OSP representa a primeira experiência com um

[2]Não realizamos este questionamento no estudo EC1.

[3]Também não realizamos este questionamento no estudo EC1.

projeto real para muitos estudantes, é normal que estes sintam-se apenas parcialmente preparados. Entretanto, destacamos que também corresponde ao primeiro e relevante passo, como reconhecido por estudantes em EC2 e EC3, e semelhantemente ressaltado por Sowe e Stamelos (2007).

Também associada à preparação para o mercado, está a contribuição da abordagem para suprir as deficiências de formação apontadas pela a indústria, como veremos a seguir.

9.5 LACUNAS NA INDÚSTRIA DE SOFTWARE (OB9)

Ao longo dos estudos, procuramos verificar se a aprendizagem detectada ajudaria a suprir as deficiências de formação dos profissionais recém-formados apontadas pela indústria de software (Ob9). Em geral, apenas analisamos o alcance dos objetivos de aprendizagem associados a tais deficiências (Tabela 3.5). Apesar de representar um método simples, como mencionamos em 4.6, permite-nos ter uma visão geral da contribuição da utilização de projetos de código aberto para atender as necessidades da indústria.

A Tabela 9.8 resume os resultados obtidos nos estudos de caso (Tabelas 6.3, 7.12 e 8.10) e na síntese das evidências (RS) (Tabela 5.21). As deficiências estão organizadas por área de conhecimento do SWEBOK. Para cada deficiência especificamos o objetivo de aprendizagem associado e o resultado em cada investigação. O hífen ('-') representa que a deficiência não foi diretamente tratada pela abordagem ou investigada no decorrer da pesquisa.

Observando a Tabela 9.8, vemos que a abordagem ajuda a suprir várias deficiências. Inclusive, em alguns casos, com resultados positivos em mais de uma investigação. Em outros casos, ocorreu alguma contribuição, mesmo que parcial. Claramente, constatamos a contribuição da abordagem para as deficiências relacionadas à prática profissional, dada à experiência profissional propiciada.

Lembramos que, em cada estudo, discutimos seus resultados particulares, e ressaltamos que nem todas as deficiências foram alvo da aplicação da abordagem ou investigadas diretamente. Assim, muitos destes resultados afloraram naturalmente das entrevistas com os estudantes. Se tivéssemos focalizado a aplicação da abordagem no suprimento de outras deficiências ou empregado métodos de investigação mais adequados, poderíamos aumentar o escopo de atendimento. Outro ponto a destacar é que apenas as deficiências apontadas com mais frequência foram analisadas (ver Capítulo 3). Isto quer dizer que outras deficiências existem e poderiam ser supridas pelos demais objetivos de aprendizagem alcançados, particularmente no que se refere às deficiências técnicas.

De modo geral, concluímos que o uso de projetos de código aberto auxilia no suprimento de muitas deficiências de formação indicadas pela indústria (Ob9), que este escopo é abrangente, considerando as diversas áreas de conhecimento da engenharia de software cobertas, e que os métodos de aplicação podem ser aperfeiçoados, a fim de que esta abrangência seja ainda maior.

Tabela 9.8: Contribuição para o suprimento das lacunas de formação apontadas pela indústria.

Deficiência	**LO**	**EC1**	**EC2**	**EC3**	**RS**
Requisitos de Software					
Especificação de Requisitos	LO6	-	-	Parcial	-
Design de Software					
Nível de detalhamento dos modelos	LO38	-	-	-	Sim
Construção de Software					
Comentar o código adequadamente	LO62	-	-	-	Parcial
Uso de ambientes integrados de desenvolvimento	LO82	Sim	Sim	-	Sim
Testes e	LO66a	-	-	-	Sim
localização do erro (debugging) no código	LO66b	-	-	-	Parcial
Teste de software					
Construção de testes de unidades não redundantes	LO99	-	Sim	-	-
Processo de testes e correção de erros sem que haja a reintrodução de erros antigos	LO100	-	-	-	Parcial
Uso de ferramentas de testes	LO107	-	Sim	-	-
Uso de ferramentas de localização de erros (*debugging*)	LO108	-	Sim	-	-
Uso de ferramentas de cobertura de testes	LO109	-	Sim	-	-
Exposição a ferramentas de integração contínua e testes de regressão	LO110	-	Sim	-	-
Manutenção de software					
Manutenção de software	LO118	Sim	-	-	Sim
Gerência de Configuração do Software					
Uso de ferramentas de gerenciamento de configuração de software	LO133	Parcial	Sim	-	-
Realização de merge entre códigos e/ou ramificação do código (*branching*)	LO134	Parcial	Sim	-	-
Prática Profissional em Engenharia de Software					
Comportamento ético	LO199	-	-	-	Parcial
Compreensão das expectativas que recaem sobre o profissional da área	LO203	-	Sim	Sim	-
Reconhecimento e aprendizagem com os próprios erros, aceitar respostas de terceiros	LO233	-	Sim	Sim	-
Gerenciamento do tempo	LO234	-	-	-	Sim
Experiência com trabalho em equipe	LO227	Não	Sim	Sim	Parcial
Pensamento crítico, geração e síntese de novas ideias	LO241	-	Sim	Sim	-
Solução de problemas	LO266	-	Sim	Parcial	Sim
Propor soluções alternativas	LO267	-	Sim	Parcial	Sim
Pró-atividade	LO236	-	Sim	Parcial	-
Experiência com projetos reais	LO269	Sim	Sim	Sim	Sim
Habilidade de visualizar o projeto como um todo (visão do todo)	LO272	-	Sim	Sim	-
Habilidade de comunicação escrita para produção de documentação técnica	LO246	-	Parcial	Parcial	Sim
Hesitação em solicitar ajuda	LO249	-	Parcial	Parcial	Parcial
Articulação de argumentos e fornecimento de explicações claras	LO244	-	-	Não	-

9.6 CONSEQUÊNCIAS PARA A APRENDIZAGEM (OB5)

Além de investigar quais objetivos de aprendizagem poderiam ser alcançados e quais deficiências de formação a abordagem de uso do OSP poderia ajudar a suprir, procuramos explorar as consequências para a aprendizagem de engenharia de software, ao aplicarmos esta abordagem (Ob5). Especificamente dentre as possíveis consequências, perscrutamos: se o estudante percebeu o seu contato com o projeto de código aberto como uma experiência de mundo real (Ob7); se a adoção de projetos de código aberto permitiu que o estudante fizesse a conexão do que é visto na teoria com a prática (Ob6) e, finalmente, se e como o uso do projeto de código aberto permitiu ao estudante sentir-se preparado para o mercado de trabalho (Ob8), resultados estes, já discutidos no decorrer deste capítulo. Nesta seção, discorremos sobre a percepção do projeto de código aberto como um ambiente autêntico de aprendizagem, mudanças de atitudes causadas pela experiência proporcionada, outras aprendizagens proporcionadas e encerramos com uma visão geral da contribuição da abordagem para o processo de aprendizagem do estudante.

Projeto de código aberto como ambiente autêntico de aprendizagem

Como mencionamos no Capítulo 2, para a experiência de aprendizagem gerar bons resultados, ela precisa ser interessante e estar conectada com que será necessário para o estudante futuramente (DEWEY, 1938). No caso de estudantes de graduação, deve ser útil para sua vida profissional. Como vimos em 9.1, projetos de código aberto exibem uma série de características que os aproximam dos softwares que são trabalhados nas empresas, de maneira que as atividades que precisam ser executadas são semelhantes. Buchta et al. (2006) também destacaram em seu estudo que, ao terem que realizar mudanças incrementais em projetos de código aberto, os estudantes precisaram lidar com software de tamanho realístico e configurações similares às que ocorrem na indústria. Portanto, a proposta de sua utilização para a aprendizagem de engenharia de software conecta o estudante com o que precisará realizar em um ambiente profissional.

Em nossos estudos, um achado merece destaque. Um dos estudantes do estudo EC3 relatou, como consequência para sua aprendizagem, o fato de o conteúdo ter sido aplicado de forma contextualizada, de maneira que ele não precisaria transpor o conhecimento adquirido para um caso real, como geralmente acontece. Assim, o uso de projetos de código aberto como exemplo de mundo real apresenta-se como uma solução para o problema de *situated learning*, conforme descrevemos no capítulo 2.

O projeto de código aberto ainda exibe algumas características que favorecem a aprendizagem sob aspectos distintos dos que são encontrados usualmente em disciplinas de engenharia de software.

Ao trabalhar com um OSP, os estudantes precisam tratar com um 'projeto existente', diferentemente de construir um software desde o início. Segundo Carrington (2003), esta mudança de foco provê uma experiência mais realística para os estudantes. Isto porque, muitas vezes, os profissionais defrontam-se com esta situação ao invés do desenvolvimento do software a partir do zero, situação com a qual os estudantes estão mais acostumados na academia. Ao longo de nossa investigação, os estudantes, além de reconhecerem essa mudança de foco como importante para a sua formação profissional (EC3), acrescenta-

ram que o OSP não constitui um software criado somente para atender as necessidades da disciplina (EC1). Consequentemente, foge do padrão (i.e., não foi criado nem pelo professor, nem pelos próprios estudantes) e não apresenta simplificações (EC2), assim como na indústria,.

Um dos diferenciais da utilização de projetos de código aberto como exemplo de projeto de software a ser trabalhado pelos estudantes em cursos de engenharia de software é a 'disponibilidade do código'. Estudantes, nos estudos EC1 e EC3, aperceberam-se deste benefício. O código de um projeto real está disponível para que seja estudado, analisado, modificado, testado por um estudante, por um grupo de estudantes ou por toda turma, sem que seja necessário qualquer processo burocrático ou preocupação com sigilo do código, já que não representa um software proprietário. Se a qualidade do que for produzido pelos estudantes não for satisfatória, não ocorrerão perdas financeiras. Se estudantes desistirem do projeto, o professor não precisará substituí-los a qualquer custo, a fim de cumprir determinado contrato.

A disponibilidade do código de um projeto de software, com muitas de suas funcionalidades já implementadas, permite a aplicação de uma 'abordagem diferente', complementar para alguns assuntos como requisitos de software ou essencial para outros, a exemplo de manutenção e evolução de software. No estudo EC3, estudantes realçaram que a experiência com o projeto de código aberto acrescentou ao método comumente aplicado (identificação e especificação de requisitos) a possibilidade de enxergar de outra maneira o conteúdo estudado (identificar os requisitos em um software existente, cuja documentação não é satisfatória), com uma visão próxima da realidade. Um dos estudantes do estudo EC1 destacou que a visão do código existente possibilita que o estudante tenha a percepção do que poderá acontecer com um software durante seu ciclo de vida, de maneira que poderá visualizar o decaimento e envelhecimento do código concretamente.

Outra característica que aproxima a utilização do OSP do que acontece nas empresas é o fato de o estudante ter que lidar com 'código desenvolvido por terceiros'. Em uma empresa, nem sempre o profissional irá trabalhar apenas com código que ele próprio tenha desenvolvido. Assim, com a abordagem, o estudante experimenta esta situação ainda na academia, vivencia as dificuldades advindas e aprende a enfrentá-las. Essa situação foi reconhecida nos estudos EC2 e EC3.

Como o código é desenvolvido por terceiros, os estudantes podem visualizar 'diferentes estilos de programação'. Nos três estudos, os estudantes admitiram que se defrontaram com esta situação. Cria-se, portanto, uma oportunidade para que ocorra a aprendizagem social (descrita no Capítulo 2), explicada pela zona de desenvolvimento proximal de Vygotsky ou pela aprendizagem por observação proposta por Bandura (LEFRANÇOIS, 2012). Vendo o código, estudantes apontaram a possibilidade de descobrir soluções interessantes ('boas soluções') que também se aplicariam às suas necessidades (EC1). Ou o contrário, os estudantes podem discernir como não escrever os próprios programas (ELLIS et al., 2007). Após o emprego da abordagem, estudantes no estudo EC3 reconheceram que projetos de código aberto representam 'ambientes de aprendizagem disponíveis', seja para visualizar os erros e acertos do código disponibilizado e assim contribuir para a construção do próprio código, seja para colocar em prática o conteúdo estudado e assim ganhar mais experiência. Marmorstein (2011) também reportou que muitos dos estudan-

Tabela 9.9 Características do OSP que favorecem a aprendizagem.

EC1	**EC2**	**EC3**
Representar a realidade.	Software real.	Software real.
-	-	Proximidade com a realidade.
-	-	Permite a aprendizagem contextualizada.
Disponibilidade de código.	-	Disponibilidade de código.
Projeto existente.	Projeto existente.	Projeto existente.
Possibilita uma abordagem diferente.	Diferente da academia.	Possibilita uma abordagem diferente.
-	Código desenvolvido por terceiros.	Código desenvolvido por terceiros.
Diferentes estilos de programação.	Estilos diferentes de programação.	Estilos diferentes de programação.
-	Exibe padrões de organização.	-
-	Desenvolver a habilidade de entender e adequar-se ao padrão utilizado pela comunidade.	-
Boas soluções.	-	Ambiente de aprendizagem disponível.

tes que participaram de seu estudo perceberam o projeto como instrutivo, possibilitando que aprendessem sobre *design*, implementação e manutenção.

Por último, colaborando para uma visão mais profissional, estudantes no estudo EC2 relataram que, através do OSP, puderam enxergar que é preciso estabelecer e seguir algum padrão de organização para que seja possível trabalhar em equipe, principalmente considerando equipes realmente grandes.

A Tabela 9.9 resume todas as características discutidas, detectadas a partir do relato dos estudantes em cada estudo. Resultados da síntese de evidências (RS) confirmam os benefícios da utilização de projetos de código aberto por prover uma experiência de mundo real, permitir que o código seja manipulado livremente e possibilitar que os estudantes trabalhem com um projeto existente (ver Tabela 5.12).

Tudo o que foi discutido colabora para a visão da abordagem de uso de projetos de código aberto como um ambiente autêntico de aprendizagem, relevante para a formação do estudante, sentimento compartilhado por outros pesquisadores. Buchta et al. (2006) realçaram que aplicar o processo de mudança incremental em um projeto de código aberto permitiu que os estudantes trabalhassem em programas realísticos, dando-lhes uma importante experiência com projetos. Já Xing (2010) destacou que usar projetos de código aberto é um meio efetivo de engajar e expor os estudantes a uma experiência de desenvolvimento de software do mundo real. Tanto este autor quanto os anteriores ainda enfatizaram a motivação dos estudantes com a abordagem.

Tabela 9.10 Conscientizações geradas pela aplicação da abordagem em cada estudo.

EC1	**EC2**	**EC3**
-	Importância da teoria.	-
-	Importância da Engenharia de Software.	Importância das práticas propostas pela Engenharia de Software.
-	-	Importância da documentação.
-	-	Importância da modelagem.
-	Importância da organização do projeto.	-
Consciência sobre o ciclo de vida do software.	-	-
Importância de utilizar processo sistemático para manutenção e evolução.	-	-
-	Importância dos testes.	-
-	Importância de testes automatizados.	-
-	É preciso testar todas as possibilidades.	-
-	-	Importância do rastreamento entre requisitos.
-	-	Valorizar o trabalho do desenvolvedor.

Mudanças de atitudes

Embora a conscientização de um indivíduo sobre determinado aspecto não garanta uma mudança de comportamento, ela representa o primeiro e importante passo nessa direção. Por isso, perscrutamos se o uso do OSP, através da conexão da teoria com a prática em um projeto real, possibilitou alguma conscientização sobre a importância do conteúdo estudado. Como realizamos os estudos EC2 e EC3 após a definição dos objetivos finais para a pesquisa, investigamos quantativamente a conscientização dos estudantes sobre aspectos relacionados às áreas de conhecimento tratadas. Contudo, tanto nestes estudos quanto no estudo EC1 e na síntese de evidências (RS), a percepção dos estudantes sobre a relevância dos princípios, métodos e técnicas propostos pela engenharia de software também aflorou dos dados coletados qualitativamente.

A Tabela 9.10 mostra a conscientização detectada em cada estudo de caso, enquanto que a Tabela 9.11 reporta a concordância dos estudantes sobre a contribuição da abordagem para compreensão da importância dos aspectos estudados nos estudos EC2 e EC3, investigada diretamente. Como podemos verificar nas tabelas, existem poucas interseções entre os estudos sobre este aspecto.

Os resultados presentes na Tabela 9.10 demonstram a conscientização dos estudantes

Tabela 9.11 Concordância dos estudantes sobre a contribuição da abordagem para compreensão da importância dos aspectos estudados.

Aspecto investigado	**EC2**	**EC3**
Compreender a importância da especificação de requisitos.	86,67%	100%
Compreender a importância de um software bem documentado (modelos, documentação do código, etc.)	100%	100%
Compreender a importância da construção de testes automatizados dentro do processo de desenvolvimento e evolução do software.	86,67%	-
Compreender a importância da aplicação de técnicas apropriadas para testar o software.	86,67%	-
Compreender a importância da criação de casos de testes de maneira sistemática.	86,67%	-
Compreender a importância da engenharia de requisitos.	-	100%

sobre diversos aspectos. De modo geral, podemos resumir que, a partir do emprego de um software real, especificamente um OSP, os estudantes conscientizaram-se sobre a importância da teoria, sobretudo dos princípios, métodos e técnicas propostos pela engenharia de software para lidar com projetos grandes e complexos. Obviamente, como cada estudo tratava de determinada área de conhecimento, a conscientização ocorreu nestas áreas.

Outros estudos também relataram que, com a utilização de projetos de código aberto, os estudantes aperceberam-se da relevância dos conceitos e princípios propostos pela engenharia de software (ELLIS et al., 2007) (MORELLI et al., 2009) (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008). Ellis et al. (2007) ainda ressaltaram que esta visão dificilmente é obtida em um curso usual. Já Nandigam, Gudivada e Hamou-Lhadj (2008) salientaram que tal conscientização proveio da experiência prática, e não simplesmente do discurso do professor.

Em nossos estudos, os estudantes confirmaram a prática com um projeto real como razão para essa conscientização (EC3), principalmente em virtude das dificuldades advindas dessa utilização (EC2).

As dificuldades são ainda maiores quando o software encontra-se mal documentado, conforme discutimos em 9.1. Portanto, reconhecer a importância da documentação acontece naturalmente. 100% dos estudantes nos estudos EC2 e EC3 concordaram com a contribuição do projeto para compreender a importância de um software bem documentado (Tabela 9.11). Na síntese (RS), também encontramos várias evidências que comprovam a conscientização dos estudantes com relação à documentação, inclusive sobre a autodocumentação do código (Tabela 5.9). Estudantes reconheceram a sua relevância até mesmo para o próprio desenvolvedor do software, quando este precisar retornar ao projeto para fazer alguma modificação (EC3) (ELLIS et al., 2007).

Aliada à questão da documentação, houve a conscientização sobre a necessidade de organização do projeto para facilitar o trabalho de futuros desenvolvedores (EC2). Em McCartney, Gokhale e Smith (2012), após a aplicação da abordagem, os estudantes salientaram que é preciso trabalhar de maneira metódica para que futuramente outras pessoas

possam entender seus trabalhos.

No que diz respeito à modelagem, um estudante no estudo EC3, acostumado a elaborar somente o diagrama entidade-relacionamento para o software em construção, admitiu que, para ter mais chance de sucesso em projetos grandes e complexos, outros modelos também devem ser elaborados. Por outro lado, na síntese das evidências (RS), capturamos que os estudantes reconheceram a importância da modelagem para a compreensão do código e para futuras implementações (Tabela 5.9).

Especificamente para as áreas de conhecimento trabalhadas nos estudos de caso, destacamos a conscientização ocorrida no estudo EC1 de que as decisões tomadas ao longo do desenvolvimento do software irão impactar no seu ciclo de vida, de modo que é preciso pensar na evolução do software, ainda durante o seu desenvolvimento. Esta evidência confirma o argumento, dentre outros, utilizado por Ellis, Morelli e Hislop (2008a) para a utilização de projetos de código aberto voltados para necessidades humanitárias na educação em engenharia de software. Servindo como uma evidência de mudança de atitude, em McCartney, Gokhale e Smith (2012), os autores relataram, justamente, a preocupação dos estudantes em evitar a desestruturação do código ao ter que realizar uma modificação.

Enquanto que na síntese das evidências (RS) detectamos a conscientização dos estudantes quanto à importância da manutenção e evolução de software (Tabela 5.9), estudantes com alguma experiência profissional no estudo EC1 reconheceram que a manutenção de software não deve ser intuitiva, como geralmente eles executavam, mas deve seguir um processo sistemático a fim de não causar outros problemas. Neste estudo, também encontramos evidências da mudança de atitude, uma vez que, estudantes relataram já estarem utilizando práticas do processo estudado em seus estágios.

De modo particular, obtivemos a conscientização no que concerne a requisitos e testes de software em cada um dos estudos onde estas áreas foram tratadas, como podemos visualizar nas Tabelas 9.10 e 9.11 e discutidos nos Capítulos 7 e 8. Enfatizamos apenas a mudança de atitude relatada por um dos estudantes participante do estudo EC2, que posteriormente afirmou, que atuando em seu trabalho, estava procurando construir o software de modo a facilitar a criação de testes automatizados.

Ao realizarmos a síntese dos estudos relacionados (RS), acrescentamos ainda outras categorias de evidências relacionadas à conscientização dos estudantes (Tabela 5.9), entre as quais: (i) que existem melhores práticas a serem seguidas para a programação; (ii) de que seus trabalhos encaixam-se em um contexto maior, de maneira que é preciso preocupar-se em como integrá-lo ao trabalho dos demais desenvolvedores; (iii) sobre a importância dos projetos de código aberto para a sociedade e, consequentemente, (iv) da relevância em aprender sobre projetos de código aberto. Cabe ressaltar que a consciência de que o próprio trabalho encaixa-se em um contexto maior representa uma visão mais profissional e importante para que o estudante aprenda a trabalhar em equipe.

Estes resultados mostram o potencial de conscientização dos estudantes sobre aspectos importantes da engenharia de software, através da abordagem de uso de projetos reais, especificamente considerando projetos de código aberto. Como mencionamos, a conscientização representa o primeiro passo para a mudança de atitude. Em nossa investigação, identificamos três exemplos onde realmente esta mudança de comportamento

ocorreu. Todavia, para avaliar de fato a ocorrência de mudanças de comportamento, seria necessário realizar um acompanhamento em longo prazo dos estudantes, o que não correspondia ao propósito desta pesquisa. Por conseguinte, outra questão que poderia ser investigada seria se tais mudanças de atitude gerariam produtos de melhor qualidade, por exemplo, melhor documentados, mais fáceis de manter, mais testáveis ou mais corretos.

Outros conhecimentos adquiridos e habilidades desenvolvidas

Ainda como consequência da aplicação da abordagem, identificamos outros conhecimentos adquiridos e habilidades desenvolvidas para os quais não existia nenhum objetivo de aprendizagem estabelecido que pudesse ser associado. São exemplos dessa aprendizagem, a familiaridade com conceitos como decaimento de código e recuperação arquitetural (EC1), e o desenvolvimento de habilidades como identificar casos de uso e fazer a rastreabilidade de requisitos (EC3).

A própria aprendizagem ocorrida sobre projetos de código aberto tem seu valor (EC2, EC3 e RS). Além de despertar a curiosidade do estudante sobre como é participar de um OSP, identificada em nossos estudos, o assunto tem ganhado destaque, tendo em vista a preparação para o mercado, dado o interesse crescente sobre este tipo de projeto tanto no ambiente público quanto privado (MEGIAS et al., 2009) (LUNDELL; PERSSON; LINGS, 2007). Mesmo os estudantes reconhecem a sua importância para a sociedade e, por conseguinte, a necessidade de aprendizado a seu respeito, conforme recuperado na síntese de evidências (RS) e mencionado no tópico anterior.

Ademais, os estudantes aperceberam-se que não é tão difícil contribuir para um OSP (EC2 e EC3), que podem participar através de ações simples. Segundo Gokhale, Smith e McCartney (2013), projetos de código aberto permitem uma série de aprimoramentos de diferentes níveis de dificuldade. Para Lundell, Persson e Lings (2007), é preciso mudar a concepção incorreta de que para participar de um OSP, o indivíduo tem que ser um técnico ou programador habilidoso.

O fato de não existirem correspondências entre os objetivos de aprendizagem estabelecidos e os conhecimentos e habilidades que mencionamos nesta seção é um indício que o conjunto de objetivos estabelecido no Capítulo 3 precisa ser ajustado. Assim, após a sua utilização como base para análise da aprendizagem, identificamos que é preciso acrescentar objetivos ausentes, subdividir objetivos que compreendem mais de um propósito e eliminar redundâncias.

Visão geral sobre o processo de aprendizagem

Diante de tudo que foi exposto, resumimos a interpretação sobre o que extraímos das evidências encontradas na Figura 9.4. Esta visão não descreve a aprendizagem obtida[4], mas complementa a Figura 6.1. Sem a preocupação com questões como motivação e satisfação, detalha o processo de aprendizagem proporcionado pela abordagem. Vale recordar que representamos a influência da associação teoria e prática para a aprendizagem nas Figuras 9.1, 9.2 e 9.3.

Tendo em vista toda a discussão apresentada neste capítulo, representada na Figura

[4]Representada nas Figuras 6.8, 7.9 e 8.9.

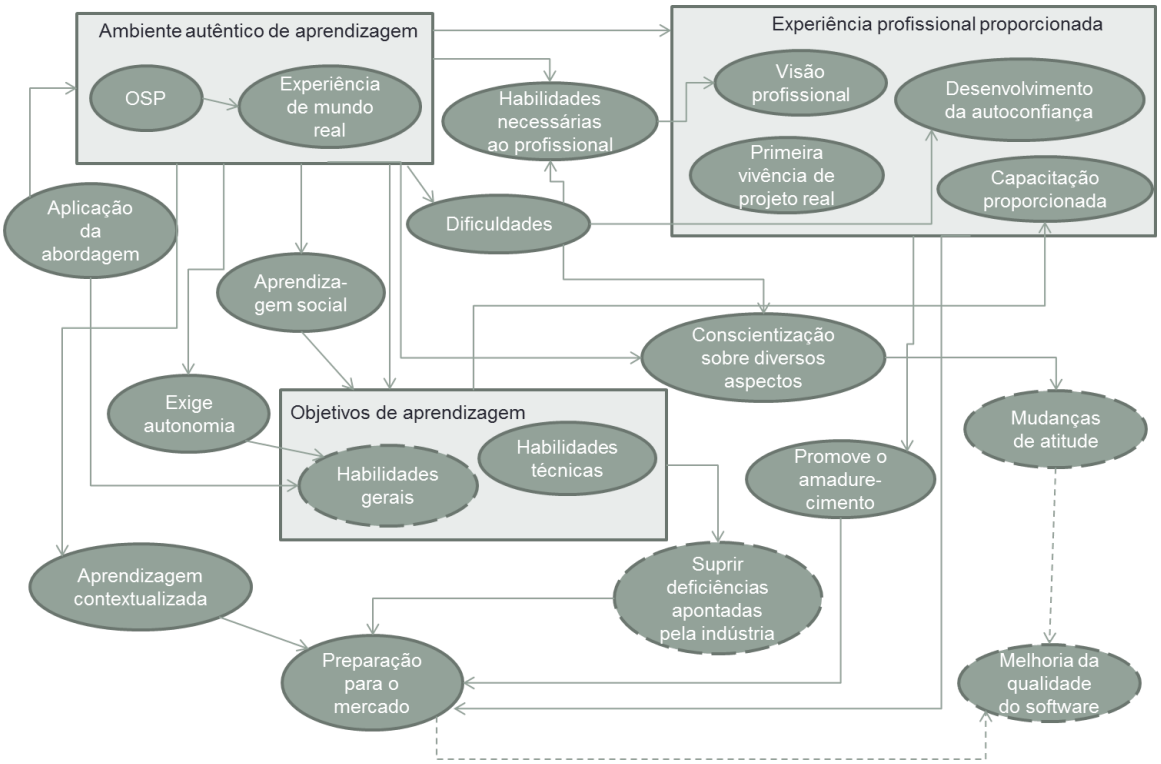

Figura 9.4 Visão sobre o processo de aprendizagem com OSP.

9.4, vemos que a aplicação da abordagem cria um ambiente autêntico de aprendizagem que, por sua vez, permite alcançar alguns objetivos de aprendizagem e, com isso, ajuda a suprir algumas deficiências de formação apontadas pela indústria de software. Paralelamente, o estudante ganha alguma experiência profissional, sendo que, para muitos, corresponde a sua primeira experiência. Esta experiência promove o amadurecimento do estudante. O ambiente autêntico também exige do estudante mais autonomia, o que possibilita o desenvolvimento de algumas habilidades gerais. O desenvolvimento de algumas habilidades gerais também é influenciado pela forma que a abordagem é aplicada. As dificuldades advindas do ambiente autêntico e a própria experiência de mundo real permitem a visualização das habilidades necessárias ao profissional que trabalha com software e a conscientização sobre diversos aspectos propostos pela teoria da engenharia de software. Visualizar as dificuldades e conseguir transpô-las aumenta a autoconfiança do estudante, assim como o alcance dos objetivos de aprendizagem coopera para a sua capacitação. O ambiente autêntico ainda possibilita a aprendizagem social, em virtude dos exemplos concretos disponíveis, produzidos por outros profissionais, e a aprendizagem contextualizada, dada a proximidade com a realidade. A aprendizagem contextualizada, o suprimento de deficiências, a experiência profissional e o amadurecimento proporcionados colaboram para a preparação do estudante para o mercado de trabalho. Por outro lado, a conscientização sobre os princípios, métodos e técnicas propostos pela engenharia de software pode acarretar em mudanças de atitudes. Finalmente, conjecturamos que a preparação para o mercado, aliada às possíveis mudanças de atitude, podem levar a

produção de softwares de melhor qualidade.

A Figura 9.4 representa o mapa mental do que pode ser conseguido, a depender do contexto e de como a abordagem for aplicada, já que não era objetivo dessa pesquisa fazer generalizações, mas entender as possibilidades. Nos estudos de caso que realizamos, o responsável pela abordagem selecionou um determinado OSP e definiu as atividades que deveriam ser realizadas pelos estudantes sem, contudo, exigir a interação com a comunidade do projeto ou mesmo que os produtos gerados fossem disponibilizados como contribuições para a comunidade. Representaram, portanto, exemplos da abordagem com 'controle interno' que descrevemos em 5.1.

Ao realizarmos a síntese das evidências reportadas pelos estudos relacionados, não distinguimos entre o nível de controle do professor com relação ao projeto e as atividades que foram realizadas pelos estudantes. Os resultados apresentados sumarizaram todos os tipos de controle: para quatro estudos não havia 'nenhum controle'; quatro eram do tipo 'definição interna e aprovação externa'; em seis, o controle era interno e, por fim, um com controle total (ver 5.1). Apesar da diversidade neste aspecto, como vimos no decorrer da discussão deste capítulo, as evidências da síntese confirmam as nossas evidências e, por conseguinte, a nossa interpretação retratada no mapa apresentado. A principal diferença provém dos estudos onde houve a interação com a comunidade do projeto (especificamente na abordagem 'nenhum controle'), onde nem sempre aconteceu uma boa acolhida por parte desta, prejudicando, assim, o alcance dos objetivos de aprendizagem relacionados à interação com desenvolvedores externos.

A Figura 9.4 corresponde a uma visão geral do processo de aprendizagem que pode ser desencadeado com OSP. Porém, em muitos casos, além do próprio aprimoramento da aplicação da abordagem, investigações mais adequadas e detalhadas devem ser realizadas, conforme mencionamos ao longo deste capítulo. Principalmente, se considerarmos o desenvolvimento de habilidades gerais, o atendimento das deficiências apontadas pela indústria, a real mudança de atitude por parte dos estudantes e a melhoria da qualidade dos softwares produzidos. Contornos e linhas tracejadas indicam estes elementos na figura.

Outros estudos ressaltaram ainda o profissionalismo (ELLIS; MORELLI; HISLOP, 2008a), a ampliação da visão e o desenvolvimento da autoconfiança dos estudantes (REKHA et al., 2009) como consequência da aplicação da abordagem, dentre outras. Entretanto, nenhum deles descreveu com tantos detalhes como isso acontece.

Finalmente, Ellis et al. (2007), depois de relatarem as lições aprendidas com a utilização de um projeto de código aberto com fins humanitários para o ensino de engenharia de software, comentaram que as respostas que obtiveram representavam tudo que eles, como educadores do curso de ciência da computação, esperavam alcançar, contudo, raramente conseguido em sala de aula. Mesmo utilizando um projeto de código aberto cujo propósito não era a ajuda humanitária e aplicando-o em três situações distintas, podemos dizer que o nosso sentimento é o mesmo.

Entretanto, conjuntamente com os benefícios apresentados, precisamos mostrar que o uso de projetos de código aberto, além das dificuldades a serem enfrentadas pelos estudantes por se tratar de um software real (discutidas em 9.1), impõem também dificuldades aos professores, provenientes da aplicação da abordagem, conforme relatamos a seguir.

Tabela 9.12 Dificuldades enfrentadas pelo professor ao aplicar a abordagem.

EC1	**EC2**	**EC3**
Planejamento das atividades.	Planejamento das atividades.	Planejamento das atividades.
Cobertura parcial dos tópicos teóricos na execução das atividades práticas.	Dificuldades na execução dos testes.	Restrições quanto aos objetivos de aprendizagem que poderiam ser trabalhados.
Escolha do projeto	Escolha do projeto.	Escolha do projeto.
Suporte às atividades dos estudantes.	Não há um suporte dos desenvolvedores já estabelecido.	Dificuldade de informação.
Problemas com o sistema de controle de versões distribuído.	Uso de ferramentas: gerenciamento de versões, automação de *build* e testes.	Dificuldades com programação.
Não houve tempo suficiente para outros projetos práticos.	Pouco tempo disponível.	Tempo curto.
-	-	Sobrecarga do final do período.
-	-	Entendimento das atividades solicitadas.

9.7 DIFICULDADES PROVENIENTES DA APLICAÇÃO DA ABORDAGEM

Ao discutirmos a proximidade de projetos de código aberto com os softwares encontrados nas empresas, apresentamos as dificuldades percebidas pelos estudantes ao terem que executar as atividades solicitadas em projetos reais. Todavia, os professores interessados em aplicar a abordagem também precisam enfrentar obstáculos. A Tabela 9.12 expõe as dificuldades encontradas ao aplicarmos a abordagem para a aprendizagem de evolução e manutenção de software (EC1), testes de software (EC2) e requisitos de software (EC3). Como discorremos a seguir, constatamos que as dificuldades são recorrentes na aplicação da abordagem. Cabe salientar que discutimos os problemas de modo geral, uma vez que os detalhes de cada dificuldade já foram descritos em cada estudo.

Planejamento das atividades

Encontramos dificuldades no planejamento das atividades a serem realizadas pelos estudantes nos três estudos de caso. Como mencionamos, o principal problema era dimensionar o esforço e a complexidade das atividades a serem realizadas, sem o conhecimento aprofundado sobre o software a ser manipulado.

Adequação do software à proposta didática

Outra dificuldade também relacionada ao planejamento das atividades é a dificuldade de adequação do software à proposta didática. No estudo EC1, obtivemos uma cobertura parcial dos tópicos teóricos na execução das atividades práticas, enquanto que, no estudo

EC3, não pudemos trabalhar todos os objetivos de aprendizagem propostos para a área de conhecimento devido às condições apresentadas pelo projeto. Os problemas foram as ausências de testes implementados para as funcionalidades sendo modificadas, no estudo EC1, e de modelos e documentação de especificação de requisitos, no estudo EC3, como detalhamos nos respectivos capítulos. No estudo EC2, a situação do código não impediu que os testes fossem implementados, mas exigiu que os estudantes ajustassem o código antes de implementá-los. No estudo EC3, a característica de autoconfiguração das funcionalidades do software ainda prejudicou o entendimento dos conceitos de requisitos funcionais e não funcionais.

Escolha do software

Tais situações indicam que cuidados devem ser tomados na seleção dos projetos de código aberto. Segundo (TOTH, 2006), um dos principais desafios em empregar projetos de código aberto na educação em engenharia de software está em identificar projetos que sejam apropriados. No mapeamento sistemático (NASCIMENTO; BITTENCOURT; CHAVEZ, 2015), identificamos vários estudos que proveem comentários, direções ou diretrizes para a escolha do software a ser utilizado (GEHRINGER, 2011; BUCHTA et al., 2006; ELLIS et al., 2011; TOTH, 2006; JACCHERI; OSTERLIE, 2007; PAPADOPOULOS; STAMELOS; MEISZNER, 2012; ELLIS et al., 2007; MENEELY; WILLIAMS; GEHRINGER, 2008; HEPTING et al., 2008; STROULIA et al., 2011; SOWE; STAMELOS; DELIGIANNIS, 2006; GERMAN, 2005; KAMTHAN, 2007; SOWE; STAMELOS, 2007; GOKHALE; SMITH; MCCARTNEY, 2012; ELLIS; PURCELL; HISLOP, 2012). Como contribuição para os critérios que devem ser empregados, evidências de nossos estudos de caso apontam condições que impactaram nas dificuldades, motivação e satisfação dos estudantes na realização das atividades e que, por isso, também devem ser consideradas. São elas: (i) situação do código, (ii) situação da documentação, (iii) domínio do software e (iv) tecnologia empregada.

Através da síntese das evidências (RS) dos estudos relacionados, observamos dificuldades na seleção dos projetos quando os próprios estudantes eram responsáveis por essa seleção (Tabela 5.14). Os problemas reportados foram: (i) dificuldade em encontrar um OSP que fosse adequado, (ii) dificuldade em encontrar um OSP que fosse do interesse do estudante e (iii) necessidade de mais suporte para achar um OSP apropriado. A primeira dificuldade reforça o problema encontrado em nossos estudos de achar um OSP apropriado à realização das atividades. A segunda remete à questão do domínio do software ser interessante para o estudante, conforme também mencionamos. Como já existe o desafio para o estudante de lidar com um software de porte razoável e desconhecido, se o domínio for desinteressante, os fatores terminam por somar-se e colaborar para a desmotivação do estudante, o que prejudica o processo de aprendizagem. Por fim, a última evidência demonstra que, independente de quem for o responsável pela seleção do software a ser trabalhado, professor ou estudante, a dificuldade sempre existirá e o professor precisa atuar no processo.

Ainda sobre a escolha do OSP a ser utilizado, convém ressaltar que Meneely, Williams e Gehringer (2008), embora tenham apresentado uma lista de critérios para a seleção de projetos, concluíram que não é viável encontrar um único projeto que atenda a todas as

necessidades didáticas pretendidas. Acrescentamos que este obstáculo existirá independentemente de o projeto real utilizado, uma vez que softwares reais não são criados para fins didáticos.

Suporte às atividades dos estudantes

Outra dificuldade está relacionada ao suporte necessário para que os estudantes consigam realizar as atividades solicitadas. Este suporte pode ser interno, dado pelo professor ou por algum assistente, ou externo, dado por desenvolvedores participantes da comunidade do projeto. Detectamos dificuldades ocorridas nas duas situações.

No caso interno, o conhecimento superficial sobre o software, além de dificultar o planejamento das atividades como dito, prejudicou o suporte dado aos estudantes para que estes realizassem suas tarefas. O suporte estava diretamente ligado ao auxílio no entendimento do código. No estudo EC3, onde apenas uma atividade exigia que os estudantes compreendessem o código, o suporte não foi tão necessário. Entretanto, no estudo EC2, contamos inclusive com auxílio de um assistente, experiente com a linguagem do projeto, para tentar esclarecer todas as dúvidas dos estudantes. Ele foi importante principalmente na semana próxima à data de entrega da tarefa solicitada, quando um grande volume de dúvidas era encaminhado.

A síntese de evidências (RS) também detectou esta dificuldade de suporte dado internamente. Os estudos reportaram que existe menos conhecimento sobre o OSP internamente e, consequentemente, menos orientação por parte do professor, e que há a demanda dos estudantes por respostas rápidas do professor ou assistente.

No que diz respeito ao suporte externo, estudantes nos estudos EC2 e EC3 relataram que, distintamente do que ocorre nas empresas, em projetos de código aberto, não há um suporte de desenvolvedores já estabelecido e que é difícil conseguir informações, pois não se sabe exatamente onde é possível consegui-las. Contudo, em nenhum dos estudos houve qualquer tentativa de interação com a comunidade. Quando questionamos aos estudantes porque isso não aconteceu, um estudante do estudo EC2 respondeu que não sabia se eles teriam paciência para explicá-lo. Esses fatos demonstram algum receio em interagir com a comunidade e a preferência por um suporte com o qual já tenha sido estabelecido um canal de comunicação.

Por outro lado, em alguns trabalhos relacionados, houve a interação ou tentativa de interação entre os estudantes e a comunidade. Nestes casos, ao realizarmos a síntese (RS), detectamos evidências com resultados contrários. Alguns estudos confirmaram que o suporte dado pelos desenvolvedores da comunidade foi suficiente e outros afirmaram que foi insuficiente (Tabela 5.13). Este fato atesta que podem existir comunidades mais receptivas do que outras.

Conhecimento prévio necessário

Algumas dificuldades estavam relacionadas a conhecimentos prévios que independem do uso de projetos de código aberto. Envolviam desde a habilidade de programação, ao uso de ferramentas de gerenciamento de versões, automação de *builds* e testes. Na verdade, antes de enfrentar o desafio de lidar com um projeto real, é desejável que os estudantes possuam ao menos um conhecimento razoável sobre a tecnologia que será utilizada.

Horstmann (2009) ressalta a inexperiência dos estudantes com a tecnologia empregada no projeto como um dos principais desafios a serem transpostos. Evidências provenientes da síntese (RS) também confirmam as dificuldades associadas à tecnologia utilizada, inclusive com a necessidade de configuração do ambiente[5] (Tabela 5.13). A forma de amenizar tais dificuldades esbarra novamente na seleção de projetos, mencionada anteriormente. É interessante selecionar softwares que utilizem tecnologias com as quais os estudantes já tiveram alguma experiência ao longo do curso. Quanto à falta de habilidade de programação, algum conhecimento prévio é interessante. Entretanto, como descrevemos anteriormente, o OSP pode contribuir com o seu aperfeiçoamento, em virtude dos exemplos positivos e negativos que são visualizados.

Limitação de tempo

Outra dificuldade relevante para a aplicação da abordagem corresponde à limitação do tempo. Qualquer atividade que seja incorporada ao longo de uma disciplina fica limitada ao período letivo no qual a mesma foi ofertada. Contudo, a utilização de projetos reais exige mais tempo e esforço do que um projeto simplificado apresentado pelo professor. Afinal, antes de realizar as atividades solicitadas, é preciso estudar e entender um pouco sobre o projeto. Portanto, a limitação do tempo representa um dos desafios a serem tratados pelo professor ao aplicar projetos de código aberto (ELLIS; MORELLI; HISLOP, 2008b) (MEISZNER; MOUSTAKA; STAMELOS, 2009).

A limitação de tempo prejudicou a realização dos três estudos de caso. No estudo EC1, não houve tempo suficiente para realizar modificações no projeto que exigissem a interação entre as equipes e que outras etapas do processo de mudança incremental fossem exercitadas. Nos estudos EC2 e EC3, os estudantes relataram como dificuldade o tempo curto para entender o OSP. A curva de aprendizagem das ferramentas (EC2), problemas no uso do sistema de gerenciamento de versões (EC1), a sobrecarga de atividades da própria disciplina (EC2) e de outras disciplinas (EC2 e EC3) reduziram ainda mais o tempo disponível para executar as tarefas solicitadas. Ademais, a falta de tempo prejudicou as discussões sobre as soluções apresentadas[6], nos estudos EC2 e EC3.

Satisfeitos com a abordagem, os estudantes nos estudos EC2 e EC3 sugeriram algumas melhorias, dentre as quais: que o projeto seja iniciado desde o princípio do período letivo, que o processo seja mais gradativo, enfatizando exemplos mais simples, inicialmente, e por fim, que as atividades sejam quebradas em tarefas com escopos menores e repassadas sequencialmente, estabelecendo-se prazos de entrega também reduzidos.

A síntese das evidências (RS) corrobora com o desafio da limitação do tempo, reconhecendo que adotar práticas da indústria exige mais tempo e esforço. Todavia, acrescenta como causa para tal limitação, a procrastinação do início das atividades no projeto por parte dos estudantes (Tabela 5.13). Lembrando que a procrastinação também foi percebida no estudo EC2 e reconhecida por um de seus estudantes, que inclusive propôs que as tarefas compreendessem escopos menores com prazos de entrega reduzidos.

[5]Em nossos resultados, esta necessidade foi associada pelos estudantes (EC2) a uma das características que aproximam o OSP dos softwares existentes nas empresas (Tabela 9.1).

[6]Relevantes para a aprendizagem experiencial (Capítulo 2).

Considerações sobre as dificuldades encontradas

Steinmacher et al. (2014) investigaram as barreiras que dificultam a participação de novos desenvolvedores em os projetos de código aberto. Comparando tais barreiras com as dificuldades que detectamos, discutidas ao longo dos estudos de caso e resumidas nas Tabelas 9.3 e 9.12, vemos que muitas delas se repetem, ou seja, são intrínsecas aos projetos de código aberto. As barreiras identificadas foram: (i) questões relacionadas ao código (e.g., tamanho, dificuldade de compreensão, entre outras); (ii) configuração do ambiente de trabalho (e.g., questões de configuração, descobrir a versão correta, dependência de plataformas, dependência de bibliotecas); (iii) problemas com a documentação (e.g., falta de documentação, falta de documentação do design do projeto, falta de comentários no código, documentação dispersa, documentação desatualizada, entre outras); (iv) conhecimento técnico dos novatos (e.g., falta de conhecimento da tecnologia utilizada, falta de conhecimento da linguagem de programação, falta de conhecimento prévio das ferramentas utilizadas no projeto, falta de conhecimento prévio do sistema de controle de versões, curva de aprendizagem das ferramentas, entre outras); (v) descobrir por onde começar; (vi) comportamento dos novatos (i.e., não compreendem o desafio, falta de comprometimento/coragem)e , por último, (vi) questões de interação social (e.g., questões de comunicação, demora para obter resposta, respostas grosseiras, uso de termos intimidadores, descobrir alguém para ajudar, descobrir um mentor).

Enquanto que Carrington (2003) evidenciam que as dificuldades podem variar de projeto para projeto, nós acrescentamos que as dificuldades também ocorrem mesmo com projetos proprietários, desde que representem projetos reais que não foram criados para fins didáticos. Isto porque, se um desses softwares for utilizado na educação em engenharia de software, existe a mesma possibilidade de ocorrência de: (i) questões relacionadas ao código; (ii) necessidade de configuração do ambiente de trabalho; (iii) problemas com a documentação; (iv) falta de conhecimento técnico dos estudantes; (v) impacto inicial para os estudantes; (vi) os estudantes não saberem por onde começar; (vii) os estudantes terem dificuldade na execução das atividades; (viii) não atendimento de toda a proposta didática; (ix) limitação do tempo, e finalmente (x) dificuldade no planejamento e suporte às atividades por parte do professor . Particularmente sobre as dificuldades no planejamento e suporte interno às atividades solicitadas, estas sempre irão ocorrer desde que o professor não seja um dos desenvolvedores do software.

No que diz respeito às dificuldades mais especificamente ligadas ao uso de projetos de código aberto, ressaltamos a falta ou insuficiência de suporte externo e a dificuldade de seleção do software. Quanto ao suporte dos desenvolvedores envolvidos no projeto, as evidências encontradas apontam que a dificuldade pode ocorrer com alguma frequência. Por outro lado, é mais raro de acontecer quando o projeto representa um software proprietário e de interesse de uma empresa. Provavelmente haverá um desenvolvedor da empresa designado para dar suporte aos estudantes. Por último, a dificuldade de seleção de projetos acontece devido à grande quantidade de projetos cujo código está disponível na internet, o que, em virtude das outras dificuldades e das mesmas variarem de projeto para projeto, faz com que o professor precise balancear entre os benefícios e dificuldades de cada projeto, dentro da proposta didática pretendida.

Voltamos a lembrar, que quando comparamos as dificuldades provenientes de utilizar-

mos um OSP ou outro projeto de software real, não pretendemos dizer que um ou outro é melhor. Apenas queremos justificar que, qualquer que seja a situação, onde o professor busque empregar um ambiente de aprendizagem mais autêntico, ele precisará lidar com obstáculos.

Capítulo 10

CONCLUSÕES E TRABALHOS FUTUROS

A aprendizagem é um processo próprio de cada indivíduo. Contudo, o professor pode atuar como um facilitador deste processo, desde que promova experiências externas significativas para os aprendizes. Para a formação de profissionais de computação, os currículos de referência recomendam que os estudantes vivenciem situações próximas à prática profissional e, particularmente, que participem de projetos reais. Tendo em vista a disponibilidade do código que pode ser estudado, modificado, testado, sem que seja necessário nenhum processo burocrático e sem que tentativas frustradas gerem prejuízos financeiros aos envolvidos, decidimos investigar o uso de projetos de código aberto como solução para apoiar o processo de ensino-aprendizagem de engenharia de software. Por conseguinte, ao longo deste estudo, perscrutamos como projetos de código aberto são incorporados na educação formal em engenharia de software e a eficácia desta abordagem.

Através do mapeamento sistemático realizado, mostramos de modo detalhado como a abordagem tem sido empregada para a aprendizagem de engenharia de software, bem como o interesse da comunidade científica a respeito do tema. Haja visto que a maioria dos estudos recuperados tinha como objetivo propor uma forma de aplicar a abordagem ou relatar as lições aprendidas com a sua aplicação, onde nem sempre resultados empíricos eram apresentados, direcionamos a continuidade de nossa pesquisa na busca por evidências, isto é, em averiguar a eficácia da abordagem.

Para a análise da eficácia da abordagem, apuramos se ela propicia o alcance de um subconjunto de objetivos de aprendizagem, se ajuda a suprir um subconjunto de deficiências de formação apontadas pela indústria e quais outras consequências contribuem para a preparação do estudante para o mercado de trabalho.

No que concerne aos objetivos de aprendizagem, antes de iniciarmos a investigação, estabelecemos um conjunto de objetivos que poderiam ser trabalhados por meio da abordagem. Este conjunto tinha como propósito direcionar a perquirição a partir de uma base de objetivos da área de engenharia de software que fosse independente da disciplina ou disciplinas onde a abordagem estivesse sendo aplicada. Com a construção do conjunto, chegamos a um escopo de objetivos que poderiam ser tratados. A visão demonstrou a

possibilidade de abrangência da abordagem em nível de subáreas de conhecimento do SWEBOK. Para apenas três subáreas, não encontramos a possibilidade de cobertura por meio da abordagem.

Estabelecidos os objetivos de aprendizagem, realizamos três estudos de caso, onde trabalhamos as áreas de conhecimento de evolução e manutenção de software, testes de software e requisitos de software. Para não ficarmos restritos unicamente aos nossos resultados, executamos ainda uma síntese das evidências reportadas pelos trabalhos relacionados. O estabelecimento do conjunto de objetivos foi essencial para a comparação entre as investigações realizadas. Os resultados obtidos permitiram-nos chegar a conclusões sobre as três perspectivas descritas para análise da eficácia da abordagem, conforme mencionado anteriormente.

A abordagem de uso de projetos de código aberto propiciou o alcance de vários objetivos de aprendizagem, tanto relacionados à área técnica, quanto aos associados às competências da prática profissional. Especificamente na área técnica, a adoção mostrou-se interessante para algumas áreas de conhecimento como testes de software, evolução e manutenção de software e complementar para outras, como requisitos de software. Apesar de não ter propiciado o alcance de todos os objetivos de aprendizagem investigados para determinada área de conhecimento, possibilitou o alcance de objetivos de outras áreas de conhecimento, favorecendo assim o desenvolvimento integrado das diversas áreas da engenharia de software.

Quanto aos objetivos associados às práticas profissionais, verificamos que o desenvolvimento de algumas competências é consequência da própria aplicação da abordagem. Outras, a depender das condições que porventura ocorram por se tratar de um projeto real, podem ser desenvolvidas. Por fim, algumas dependem também das atividades solicitadas e a forma que serão exigidas por parte do professor.

Para os objetivos não alcançados ou parcialmente alcançados, lembramos que todos os estudos de caso representaram a nossa primeira aplicação da abordagem para a aprendizagem da área de conhecimento sendo trabalhada, de maneira que ajustes ainda podem e devem ser realizados. Todavia, constatamos que a abordagem não descarta a necessidade de exercícios em nível crescente de dificuldade e sim, deve ser considerada como complementar a um conjunto de exercícios práticos.

No que se refere ao potencial da abordagem para suprir as deficiências de formação apontadas pela indústria, vimos que existe contribuição não somente para as áreas de conhecimento trabalhadas nos estudos de caso, mas também para as áreas inter-relacionadas. Observamos contribuições relevantes para deficiências associadas às práticas profissionais. É importante ressaltar que nem todas as deficiências foram alvo da aplicação da abordagem ou investigadas diretamente nos estudos de caso, e ainda, que analisamos apenas as deficiências apontadas com maior frequência pela literatura, de forma que outras deficiências existem e poderiam ser supridas pelos demais objetivos de aprendizagem alcançados.

Apesar de termos simplesmente usado o alcance dos objetivos de aprendizagem associados para julgar a abordagem no suprimento das deficiências apontadas pela indústria, pudemos visualizar como a adoção de projetos de código aberto ajuda a atender tais necessidades. Todavia, investigações que confirmem realmente esta contribuição, consi-

derando novamente a opinião de representantes da indústria, devem ser realizadas.

Por último, capturamos várias consequências positivas para a aprendizagem dos estudantes, principalmente considerando a sua preparação para o mercado de trabalho. Em virtude da sua proximidade com a realidade, o uso do OSP como exemplo de software a ser trabalhado pelos estudantes cria um ambiente autêntico de aprendizagem. Para muitos estudantes representa a primeira experiência com um projeto real. Este ambiente autêntico, diferentemente dos projetos simplificados usualmente utilizados ao longo das disciplinas, permite uma aprendizagem contextualizada, uma vez que o estudante não precisará transferir o conteúdo estudado para o contexto de um projeto real, pois já está vivenciando o conteúdo estudado em um contexto real. A partir de exemplos concretos, visualizados (e não imaginados), o estudante pode entender melhor os conceitos, que ganham significado e, com isso, tornam-se mais fáceis de serem assimilados e consolidados em sua estrutura cognitiva. Além dos relatos dos estudantes, o alcance de objetivos de aprendizagem em nível de 'entendimento' (taxonomia de Bloom) e 'familiaridade' (CS2013) comprovam esta percepção. O código disponível desenvolvido por outros profissionais também propicia a aprendizagem social. A prática no projeto possibilita aprender o como fazer e, assim, enxergar os problemas em sua complexidade. Porque, conforme relatado pelos estudantes, na teoria tudo parece muito simples. Somente quando precisam aplicar o conteúdo estudado em projetos reais, eles constatam as dificuldades e desafios que precisam ser transpostos. Exatamente por sentirem estas dificuldades, os estudantes podem perceber as habilidades que o profissional que trabalha com software precisa desenvolver, a aplicabilidade das práticas estudadas e se conscientizarem sobre a importância dessas práticas, i.e., princípios, métodos e técnicas propostos pela engenharia de software. Transpor estas dificuldades colabora ainda para o desenvolvimento da autoconfiança do estudante. O ambiente autêntico criado e a percepção sobre as habilidades necessárias ao profissional de software possibilitam uma visão mais profissional. A percepção sobre a aplicabilidade das práticas em conjunto com enxergar o problema em sua complexidade propiciam a discussão e com isso o desenvolvimento de algumas habilidades gerais. Algumas habilidades gerais também podem ser desenvolvidas, em virtude do ambiente autêntico exigir mais autonomia por parte do estudante. Os objetivos de aprendizagem alcançados colaboram para a capacitação do estudante. A vivência de um projeto real de software, a visão profissional obtida, o desenvolvimento da autoconfiança e a capacitação associada proveem experiência profissional ao estudante, que, por conseguinte, promove o seu amadurecimento. Finalmente, a aprendizagem contextualizada, o suprimento de deficiências apontadas pela indústria, experiência profissional e o amadurecimento proporcionados colaboram para a preparação do estudante para o mercado de trabalho.

Entretanto, apesar de reconhecerem a contribuição da abordagem, os estudantes não se consideraram completamente preparados após o uso do OSP. Além de admitirmos a necessidade de aperfeiçoamento da aplicação da abordagem nos três estudos que realizamos, supomos que esta sensação de preparação parcial deve-se à aplicação da abordagem ter representado a primeira experiência com um projeto real para muitos estudantes. Deste modo, é normal que os mesmos sintam-se apenas parcialmente preparados. Todavia, se ao longo das disciplinas de engenharia de software, ambientes autênticos forem proporci-

onados, é provável que, aos poucos, o estudante amadureça e sinta-se mais preparado.

Retornando ao enunciado de nossas hipóteses, verificamos que as evidências obtidas com a pesquisa confirmam que a adoção de um OSP propiciou o alcance de um subconjunto de objetivos de aprendizagem, ajudou a suprir um subconjunto das deficiências apresentadas pelos profissionais recém-formados atuantes na indústria de software e contribuiu para a preparação do estudante para o mercado de trabalho. Portanto, concluímos que projetos de código aberto são eficazes para a aprendizagem formal de engenharia de software. Contudo, não podemos generalizar estes resultados, em virtude da grande quantidade de variáveis de contexto que podem influenciá-los, a exemplo do contexto social, capacidade cognitiva e estado emocional dos estudantes, método didático utilizado, conhecimento prévio do professor sobre o software, conhecimento técnico do professor, características do OSP selecionado, receptividade da comunidade do projeto, entre outros. Inclusive, reconhecemos que o método que incorporamos à abordagem em nossos estudos pode ser aperfeiçoado, de forma que resultados mais interessantes possam ser gerados.

Se, por um lado, a adoção de projetos de código aberto traz vários benefícios para a aprendizagem dos estudantes, por outro, acarreta em algumas dificuldades, como: (i) no planejamento das atividades a serem executadas pelos estudantes; (ii) na adequação do software à proposta didática; (iii) com a escolha do software; (iv) no suporte às atividades dos estudantes; (v) pela necessidade de uma base de conhecimento prévio por parte dos estudantes e (vi) pela limitação de tempo. Contudo, do nosso ponto de vista, a maioria das dificuldades apontadas pode ocorrer qualquer que seja o projeto real utilizado. Apenas os problemas de escolha do software e de insuficiência de suporte dos desenvolvedores participantes do projeto são especialmente ligados aos projetos de código aberto. Portanto, o professor disposto a prover um ambiente de aprendizagem mais autêntico, diferente do usual, mas relevante para a formação profissional dos estudantes, deverá estar ciente dos obstáculos que precisará enfrentar.

Reconhecendo as oportunidades de aprendizagem providas pela adoção de projetos de código aberto na educação em engenharia de software, visualizamos a possibilidade de realização de vários trabalhos futuros.

Para a investigação de um recurso didático empregado, a análise de aspectos como a motivação e a satisfação do estudante ao lidar com este recurso é essencial, principalmente pela sua influência sobre o processo de ensino-aprendizagem. Todavia, em virtude dos vários objetivos traçados nesta pesquisa, da extensão que a análise destes objetivos estava tomando e da restrição de tempo para o término da pesquisa, vimos por bem não executarmos a análise desses aspectos. Portanto, um próximo e urgente trabalho será acrescentar a análise desses aspectos, principalmente porque os dados já foram coletados.

Um outro trabalho, não especificamente ligado à adoção de projetos de código aberto, mas a pesquisas na área de educação em engenharia de software, representa a realização de ajustes no conjunto de objetivos de aprendizagem estabelecido. É preciso acrescentar objetivos de aprendizagem relevantes ausentes (dentre os quais, alguns detectados no decorrer desta pesquisa); subdividir objetivos que compreendem mais de um foco, de maneira que o julgamento do seu alcance seja mais adequado; eliminar objetivos redundantes em áreas de conhecimento distintas, finalmente, validá-lo com outros professores de engenharia de software.

Durante a comparação e discussão dos resultados encontrados, algumas vezes ressaltamos que investigações mais detalhadas deveriam ser realizadas. Um desses exemplos seria a investigação mais detalhada sobre o desenvolvimento de habilidades relacionadas à prática profissional, dado os resultados positivos já conseguidos quanto aos objetivos de aprendizagem associados às competências da prática profissional. Por exemplo, através de uma análise mais detalhada sobre as soluções apresentadas para as atividades solicitadas, pode-se perscrutar o desenvolvimento da criatividade, proatividade, capacidade de solucionar problemas ou propor soluções alternativas. Por meio do registro das observações das discussões realizadas em sala de aula, pode-se avaliar a capacidade dos estudantes de expressar opiniões, demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.

A fim de realmente validarmos a contribuição da abordagem no suprimento das deficiências de formação apontadas pela indústria de software seria necessário uma investigação de longo prazo, com o acompanhamento da atuação profissional dos estudantes que participaram de alguma aplicação da abordagem e da opinião de seus contratantes. Outro trabalho a ser realizado, muito próximo deste, seria acompanhar o comportamento do estudante ao longo de sua atuação profissional, avaliando se a conscientização sobre os princípios, métodos e técnicas propostos pela engenharia de software realmente se refletiram em mudanças de comportamento. Finalmente, também em longo prazo, investigar qual o reflexo da aplicação da abordagem nos produtos de software gerados por estes estudantes. Entretanto, admitimos que tais pesquisas não são muito fáceis de serem implementadas.

Por último, um trabalho que já iniciamos é a construção de um projeto de código aberto internamente que, além de atender as necessidades de usuários interessados e procurar capturar o interesse de participação de outros desenvolvedores, serviria de exemplo de projeto de software a ser trabalhado no decorrer das disciplinas. Neste estudo pretendemos investigar todos esses achados, porém na perspectiva de 'controle total', onde o projeto é originado dentro da instituição de ensino e o professor faz parte do núcleo da comunidade.

Como pode ser constatado, a eficácia obtida com a adoção de projetos de código aberto nas investigações realizadas ao longo desta pesquisa só aumentou a nossa motivação e interesse pelo tema. A contextualização, os exemplos concretos disponibilizados, a possibilidade de integração entre as áreas de conhecimento que compreendem a engenharia de software, a experiência de mundo real proporcionada são apenas alguns exemplos que demonstram a potencialidade da abordagem para a educação em engenharia de software. Mesmo que a sua adoção imponha desafios, analisando a relação entre benefícios e dificuldades, concluímos que vale a pena transpor estes desafios. A utilização de projetos de código aberto na educação formal em engenharia de software é uma solução factível, viável e, em nossa opinião, apropriada como cenário de projeto de software real a ser trabalhado pelos estudantes.

REFERÊNCIAS BIBLIOGRÁFICAS

ACM AND IEEE. Joint Task Force on Computing Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. [S.l.], 2013. 514 p.

ACM AND IEEE. Joint Task Force on Computing Curricula. *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. [S.l.], 2015. 134 p.

AEE. Association for Experiential Education. *What is Experiential Education?* 2011. Disponível em: ⟨http://www.aee.org/what-is-ee⟩.

ALLEN, E.; CARTWRIGHT, R.; REIS, C. Production Programming in the Classroom. *ACM SIGCSE Bulletin*, v. 35, n. 1, p. 89–93, jan 2003. ISSN 00978418.

ANZAI, Y.; SIMON, H. A. The Theory of Learning by Doing. *Psychological Review*, v. 86, n. 2, p. 124–140, 1979.

ATKINSON, R. C.; SHIFFRIN, R. M. Human Memory: A Proposed System and its Control Processes. *Psychology of Learning and Motivation*, v. 2, p. 89–195, 1968.

AUER, L.; JUNTUNEN, J.; OJALA, P. Open Source Project as a Pedagogical Tool in Higher Education. In: *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments (MindTrek '11)*. New York, New York, USA: ACM Press, 2011. p. 207–213. ISBN 9781450308168.

BARRA, A. S. B. Uma Análise do Conceito de Zona de Desenvolvimento Proximal. *Revista da Universidade Vale do Rio Verde*, Três Corações, v. 12, n. 1, p. 765–774, 2014.

BILLIET, J. B.; MCCLENDON, M. J. On the Identification of Acquiescence in Balanced Sets of Items Using Structural Models. *Advances in Methodology, Data Analysis, and Statistics. Metodolo{\v{s}}ki zvezki*, v. 14, p. 129–150, 1998.

BISHOP, J. et al. How to use open source software in education. In: *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. Memphis: Association for Computing Machinery, Inc, 2016. p. 321–322. ISBN 9781450338561.

BONACI, C. G.; MUSTATA, R. V.; IENCIU, A. Revisiting Bloom's Taxonomy of Educational Objectives. *The Macrotheme Review: A multidisciplinary journal of global macro trends*, v. 2, n. 2, p. 1–9, 2013.

BORREGO, M.; DOUGLAS, E. P.; AMELINK, C. T. Quantitative, Qualitative, and Mixed Research Methods in Engineering Education. *Journal of Engineering Education*, v. 98, n. 1, p. 53–66, jan 2009. ISSN 10694730.

BOUD, D.; COHEN, R.; WALKER, D. Understanding learning from experience. In: BOUD, D.; COHEN, R.; WALKER, D. (Ed.). *Using Experience for Learning,*. [S.l.]: Open University Press, 1993. p. 1–18.

BOURQUE, P.; FAIRLEY, R. E. D. (Ed.). *Guide to the Software Engineering Body of Knowledge - SWEBOK v.3*. [S.l.], 2014. 335 p. Disponível em: ⟨www.swebok.org/⟩.

BUCHTA, J. et al. Teaching Evolution of Open-Source Projects in Software Engineering Courses. In: *22nd IEEE International Conference on Software Maintenance*. Philadelphia: IEEE, 2006. p. 136–144. ISBN 0-7695-2354-4. ISSN 1063-6773.

BUDD, T. A. A Course in Open Source Development. In: *Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium*. Chattanooga: [s.n.], 2009. Disponível em: ⟨http://www.cs.trincoll.edu/{~}ram/hfoss/Budd-FOSS-Course.p⟩.

BUDGEN, D. et al. Using Mapping Studies in Software Engineering. In: *Proceedings of PPIG 2008*. [S.l.]: Lancaster University, 2008. v. 2, p. 195–204.

CARRINGTON, D. Teaching Software Design with Open Source Software. In: *33rd Annual Frontiers in Education Conference (FIE)*. Westminster, CO, USA: IEEE, 2003. v. 3, p. S1C–9–S1C–14. ISBN 0-7803-7961-6.

CHEN, Y. et al. 200 Students Can't Be Wrong! GamesCrafters , a Computational Game Theory Undergraduate Research and Development Group. In: *AAAI Spring Symposium - Technical Report*. [S.l.: s.n.], 2008.

CMU. Carnegie Mellon University. *Design & Teach a Course: Articulate your Learning Objectives*. 2015. Disponível em: ⟨https://www.cmu.edu/teaching/designteach/design/learningobjectives.html⟩.

COHEN, L.; MANION, L.; MORRISON, K. *Research Methods in Education*. 6. ed. [S.l.]: Taylor & Francis e-Library, 2007. 638 p.

CONLON, M. P.; HULICK, F. W. Is There a Role for Open Source Software in Systems Analysis ? In: *Proceedings of ISECON*. [S.l.: s.n.], 2005. v. 22, p. 1–7.

COSTA-SORIA, C.; PÉREZ, J. Teaching Software Architectures and Aspect-Oriented Software Development Using Open-Source Projects. *ACM SIGCSE Bulletin*, v. 41, n. 3, p. 385–385, aug 2009. ISSN 00978418.

CRESWELL, J. W. *Projeto de Pesquisa: Métodos Qualitativo, Quantitativo e Misto*. 3. ed. Porto Alegre: Artmed, 2010. 296 p.

CRESWELL, J. W. *Investigação Qualitativa & Projeto de Pesquisa: escolhendo entre cinco abordagens*. 3 ed.. ed. Porto Alegre: Penso, 2014. 341 p.

CRUZES, D. S.; DYBÅ, T. Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, Elsevier, v. 53, n. 5, p. 440–455, 2011.

CRUZES, D. S. et al. Case studies synthesis: a thematic, cross-case, and narrative synthesis worked example. *Empirical Software Engineering*, v. 20, n. 6, p. 1634–1665, 2015. ISSN 15737616.

DAVIS, H. A.; SUMMERS, J. J.; MILLER, L. M. *An Interpersonal Approach to Classroom Management: Strategies for Improving Student Engagement*. [S.l.]: Corwin, 2012. 256 p.

DEWEY, J. *Experience and Education*. [s.n.], 1938. Disponível em: ⟨http://ruby.fgcu.edu/courses/ndemers/colloquium/experienceducationdewey.pdf⟩.

DIXON-WOODS, M. et al. Synthesising qualitative and quantitative evidence: a review of possiblemethods. *Journal of Health Services Research & Policy*, v. 10, n. 1, p. 45–53, 2005.

DRISCOLL, M. P. *Psychology of Learning for Instruction*. Boston: Allyn and Bacon, 1993.

ELLIS, H. J.; HISLOP, G. W.; MORELLI, R. A. A Comparison of Software Engineering Knowledge Gained from Student Participation in Humanitarian FOSS Projects. In: *Proceedings of the 16th Annual joint Conference on Innovation and Technology in Computer Science Education (ITiCSE'11)*. New York, New York, USA: ACM Press, 2011. p. 360–360. ISBN 9781450306973.

ELLIS, H. J. et al. Holistic Software Engineering Education Based on a Humanitarian Open Source Project. In: *20th Conference on Software Engineering Education & Training (CSEET'07)*. Dublin: IEEE, 2007. p. 327–335. ISBN 0-7695-2893-7. ISSN 1093-0175.

ELLIS, H. J.; PURCELL, M.; HISLOP, G. W. An Approach for Evaluating FOSS Projects for Student Participation. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*. New York, New York, USA: ACM Press, 2012. p. 415–420. ISBN 9781450310987.

ELLIS, H. J. C. et al. How to Involve Students in FOSS Projects. In: *41st Annual Frontiers in Education Conference (FIE)*. Rapid City, SD: IEEE, 2011. p. T1H–1–T1H–6. ISBN 978-1-61284-469-5.

ELLIS, H. J. C. et al. Student Software Engineering Learning via Participation in Humanitarian FOSS Projects. In: *ASEE Annual Conference and Exposition*. San Antonio: [s.n.], 2012.

ELLIS, H. J. C.; MORELLI, R. A.; HISLOP, G. W. Support for Educating Software Engineers Through Humanitarian Open Source Projects. In: *21st IEEE-CS Conference on Software Engineering Education and Training Workshop*. Charleston: IEEE, 2008. p. 1–4. ISBN 978-0-7695-3387-2.

ELLIS, H. J. C.; MORELLI, R. A.; HISLOP, G. W. Work in Progress: Challenges to Educating Students within the Community of Open Source Software for Humanity. In: *38th Annual Frontiers in Education Conference (FIE)*. Saratoga Springs, NY: IEEE, 2008. p. S3H–7–S3H–8. ISBN 978-1-4244-1969-2.

ELLIS, H. J. C. et al. Can Humanitarian Open-Source Software Development Draw New Students to CS? *ACM SIGCSE Bulletin*, v. 39, n. 1, p. 551–555, mar 2007. ISSN 00978418.

FERRAZ, A. P. d. C. M.; BELHOT, R. V. Bloom's taxonomy and its adequacy to define instructional objective in order to obtain excellence in teaching. *Gestão & Produção*, São Carlos, v. 17, n. 2, p. 421–431, 2010.

GASPARIN, J. L. *Objetivos do Ensino e da Aprendizagem: Análise Crítica*. PORTAL DO FÓRUM SUL E ANPED SUL, 2010. 13 p. Disponível em: ⟨http://www.portalanpedsul.com.br/admin/uploads/2010/Didatica/Trabalho/02_31_56_OBJETIVOS_DO_ENSINO_E_DA_APRENDIZAGEM_ANALISE_CRITICA.PDF⟩.

GEE, N. A study of student completion strategies in a Likert-type course evaluation survey. *Journal of Further and Higher Education*, v. 9486, n. February, p. 1–11, 2015. ISSN 0309-877X. Disponível em: ⟨http://www.tandfonline.com/doi/full/10.1080/0309877X.2015.1100717⟩.

GEHRINGER, E. F. From the Manager's Perspective: Classroom Contributions to Open-source Projects. In: *41st Annual Frontiers in Education Conference (FIE)*. Rapid City, SD: IEEE, 2011. p. F1E–1–F1E–5. ISBN 978-1-61284-469-5.

GENTRY, J. W. WHAT IS EXPERIENTIAL LEARNING? In: *Guide to Business Gaming and Experiential Learning*. [S.l.]: Nichols Pub Co, 1990. cap. 2, p. 370.

GERMAN, D. M. Experiences Teaching a Graduate Course in Open Source Software Engineering. In: *the First International Conference on Open Source Systems*. Genova, Italy: [s.n.], 2005. p. 326–328.

GHOSH, R.; GLOTT, R. *FLOSSPOLS Skills Survey Interim Report*. Maastricht, Netherland, 2005. 1–32 p. Disponível em: ⟨http://flosspols.org/deliverables/FLOSSPOLS-D10-skillssurvey{_}interim{_}report-revision-FIN⟩.

GHOSH, R. A. et al. *Free/Libre and Open Source Software: Survey and Study - Part IV: Survey of Developers*. Maastricht, Netherland, 2002. 1–69 p.

GOKHALE, S.; SMITH, T.; MCCARTNEY, R. Teaching Software Maintenance with Open Source Software: Experiences and Lessons. In: *Frontiers in Education Conference, 2013 IEEE*. [S.l.: s.n.], 2013. p. 1664 – 1670.

GOKHALE, S. S.; SMITH, T.; MCCARTNEY, R. Integrating Open Source Software into Software Engineering Curriculum: Challenges in Selecting Projects. In: *First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex)*. Zurich: IEEE, 2012. p. 9–12. ISBN 978-1-4673-1805-1.

HANNAFIN, M. J. Instructional strategies and emerging instructional technologies: Psychological perspectives. *Canadian Journal of Educational Communication*, v. 18, p. 167–179, 1989.

HEPTING, D. H. et al. Creating Synergy Between Usability Courses and Open Source Software Projects. *ACM SIGCSE Bulletin*, v. 40, n. 2, p. 120–123, jun 2008. ISSN 00978418.

HISLOP, G. W.; ELLIS, H. J.; MORELLI, R. A. Evaluating Student Experiences in Developing Software for Humanity. *ACM SIGCSE Bulletin*, v. 41, n. 3, p. 263–267, aug 2009. ISSN 00978418.

HOLMBOE, C.; MCIVER, L.; GEORGE, C. Research Agenda for Computer Science Education. In: *13th Workshop of the Psychology of Programming Interest Group*. Bournemouth UK: [s.n.], 2001. p. 207–223.

HORSTMANN, C. S. Challenges and Opportunities in an Open Source Software Development Course. In: *Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium*. Chattanooga: [s.n.], 2009. Disponível em: ⟨http://www.hfoss.org/symposium09/documentdl/papers/HFOSS{_}Symp-paper24.⟩

I-TECH. International Training and Education Center for Health . *Writing Good Learning Objectives: A Technical Implementation Guide*. 2010. 8 p. Disponível em: ⟨http://www.go2itech.org/resources/technical-implementation-guides/TIG4.WritingLrngObj.pdf/view⟩.

JACCHERI, L.; OSTERLIE, T. Open Source Software: A Source of Possibilities for Software Engineering Education and Empirical Software Engineering. In: *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)*. Minneapolis, MN: IEEE, 2007. p. 1–5. ISBN 0-7695-2961-5.

JARVIS, P. *The Paradoxes of Learning*. San Francisco:: Jossey-Bass, 1992.

KAMTHAN, P. On the Prospects and Concerns of Integrating Open Source Software Environment in Software Engineering Education. *Journal of Information Technology Education*, v. 6, p. 45–64, 2007. Disponível em: ⟨http://www.jite.org/documents/Vol6/JITEv6p045-064Kamt214.pdf⟩.

KILAMO, T. The Community Game: Learning Open Source Development through Participatory Exercise. In: *Proceedings of the 14th International Academic MindTrek Conference on Envisioning Future Media Environments (MindTrek '10)*. New York, New York, USA: ACM Press, 2010. p. 55–60. ISBN 9781450300117.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Disponível em: ⟨http://www.citeulike.org/user/akom/article/3955888⟩.

KNUDSON, D.; RADERMACHER, A. Software Engineering and Project Management in CS Projects vs. "Real-world" Projects: A Case Study. In: *IASTED Internation Conference on Software Engineering and Applications, SEA '09*. [S.l.: s.n.], 2009.

KOLB, D. *Experiential Learning as the Science ofLearning and Development*. [S.l.]: Prentice Hall, 1984.

KRATHWOHL, D. R. A Revision of Bloom's Taxonomy: An Overview. *Theory into Practice*, v. 41, n. 4, p. 212–218, 2002.

KROGSTIE, B. R. Power Through Brokering: Open Source Community Participation in Software Engineering Student Projects. In: *Proceedings of the 13th International Conference on Software Engineering (ICSE'08)*. Leipzig: ACM Press, 2008. p. 791–800. ISBN 9781605580791.

KUME, I.; NITTA, N.; TAKEMURA, Y. A Method for Creating Teaching Materials of Practical Object-Oriented Methods Education. In: *Learning by Effective Utilization of Technologies: Facilitating Intercultural Understanding, Proceeding of the 14th International Conference on Computers in Education, ICCE 2006*. Beijing, China: [s.n.], 2006.

KUSSMAUL, C. Software Projects Using Free and Open Source Software: Opportunities, Challenges, & Lessons Learned. In: *ASEE Annual Conference and Exposition*. austin: [s.n.], 2009.

LANEROLLE, T. de et al. Creating an Academic Community to build Humanitarian FOSS: A Progress Report. In: *the 5th International ISCRAM Conference*. Washington: [s.n.], 2008. p. 337–341.

LAVE, J.; WENGER, E. *Situated Learning: Legitimate Peripheral Participation*. [S.l.]: Cambridge University Press, 1991. 138 p.

LEFRANÇOIS, G. R. *Teorias da Aprendizagem*. 5 ed.. ed. São Paulo: Cengage Learning, 2012. 479 p.

LETHBRIDGE, T. C. et al. Improving software practice through education: Challenges and future trends. In: *Future of Software Engineering (FOSE '07)*. Minneapolis, MN, USA: IEEE, 2007. p. 12–28. ISBN 0-7695-2829-5.

LI, W.; ZHANG, S.; LI, Z. Open Source Movement and Computer Science Education Innovation. In: *International Conference on Information Engineering and Computer Science*. [S.l.]: IEEE, 2009. p. 1–4. ISBN 978-1-4244-4994-1.

LIU, C. Enriching Software Engineering Courses with Service-Learning Projects and the Open-Source Approach. In: *Proceedings of the 27th International Conference on Software Engineering (ICSE)*. [S.l.]: IEEe, 2005. p. 613–614. ISBN 1-59593-963-2.

LUNDELL, B.; PERSSON, A.; LINGS, B. Learning Through Practical Involvement in the OSS Ecosystem: Experiences from a Masters Assignment. In: FELLER, J. et al. (Ed.). *Open Source Development, Adoption and Innovation*. [S.l.]: Springer, 2007. p. 289–294.

LUTZ, M. J.; NAVEDA, J. F.; VALLINO, J. R. Undergraduate Software Engineering: Addressing the Needs of Professional Software Development. *Queue*, ACM, v. 12, n. 6, p. 30, jun 2014. ISSN 1542-7730.

MARMORSTEIN, R. Open Source Contribution as an Effective Software Engineering Class Project. In: *Proceedings of the 16th Annual joint Conference on Innovation and Technology in Computer Science Education (ITiCSE'11)*. New York, New York, USA: ACM Press, 2011. p. 268–272. ISBN 9781450306973.

MAXWELL, J. A. *Qualitative Research Design: An Interactive Approach*. 2nd. ed. Thousand Oaks, CA: Sage, 2005.

MCCARTNEY, R.; GOKHALE, S. S.; SMITH, T. M. Evaluating an Early Software Engineering Course with Projects and Tools from Open Source Software. In: *Proceedings of the 9th Annual International Conference on International Computing Education Research (ICER '12)*. New York, New York, USA: ACM Press, 2012. p. 5–10. ISBN 9781450316040.

MEAD, N. R. Software engineering education: How far we've come and how far we have to go. *Journal of Systems and Software*, Elsevier Science Inc., v. 82, n. 4, p. 571–575, apr 2009. ISSN 01641212.

MEC. Ministério da Educação. *Parecer CNE/CES Nº:136/2012 - Diretrizes Curriculares Nacionais para os cursos de graduação em Computação*. 2012. Disponível em: ⟨http://portal.mec.gov.br/index.php?option=com_content&view=article&id=12991⟩.

MEGIAS, D. et al. Free Technology Academy: A European Initiative for Distance Education about Free Software and Open Standards. *ACM SIGCSE Bulletin*, v. 41, n. 3, p. 70–74, aug 2009. ISSN 00978418.

MEISZNER, A.; MOUSTAKA, K.; STAMELOS, I. A Hybrid Approach to Computer Science Education a Case Study: Software Engineering at Aristotle University. In: *Proceedings of the 1st International Conference on Computer Supported Education (CSEDU)*. [s.n.], 2009. p. 39–46. Disponível em: ⟨http://pt.scribd.com/doc/10933440/A-HYBRID-APPROACH-TO-COMPUTER-SCIENCE-EDUCATION⟩.

MENEELY, A.; WILLIAMS, L.; GEHRINGER, E. F. ROSE: A Repository of Education-Friendly Open-Source Projects. *ACM SIGCSE Bulletin*, v. 40, n. 3, p. 7–11, aug 2008. ISSN 00978418.

MERRIAM, S. B. *Qualitative Research: A Guide to Design and Implementation*. San Francisco: Jossey-Bass, 2009. 304 p.

MEYER, B. Software engineering in the academy. *Computer*, IEEE Computer Society Press, v. 34, n. 5, p. 28–35, may 2001. ISSN 00189162.

MOODY, D.; SINDRE, G. Evaluating the Effectiveness of Learning Interventions: An Information Systems Case Study. In: *ECIS 2003 Proceedings*. [s.n.], 2003. p. 80. Disponível em: ⟨http://aisel.aisnet.org/ecis2003/80⟩.

MOON, J. *A Handbook of Reflective and Experiential Learning: Theory and Practice*. London: RoutledgeFalmer, 2004. 252 p.

MORELLI, R.; LANEROLLE, T. de. Foss 101: Engaging Introductory Students in the Open Source Movement. In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. New York, New York, USA: ACM Press, 2009. v. 41, n. 1, p. 311. ISBN 9781605581835. ISSN 0097-8418.

MORELLI, R. et al. Revitalizing Computing Education through Free and Open Source Software for Humanity. *Communications of the ACM*, v. 52, n. 8, p. 67–75, aug 2009. ISSN 00010782.

MORENO, A. M. et al. Balancing software engineering education and industrial needs. *The Journal of Systems and Software*, v. 85, p. 1607–1620, 2012.

NANDIGAM, J.; GUDIVADA, V. N.; HAMOU-LHADJ, A. Learning Software Engineering Principles Using Open Source Software. In: *38th Annual Frontiers in Education Conference (FIE)*. [S.l.]: IEEE, 2008. p. S3H–18–S3H–23. ISBN 978-1-4244-1969-2.

NASCIMENTO, D. M. C.; BITTENCOURT, R. A.; CHAVEZ, C. V. F. G. Open Source Projects in Software Engineering Education: A Mapping Study. *Computer Science Education*, v. 25, p. 67–114, 2015.

NAUMAN, M.; UZAIR, M. SE and CS Collaboration: Training Students for Engineering Large, Complex Systems. In: *20th Conference on Software Engineering Education & Training (CSEET'07)*. [S.l.]: IEEE, 2007. p. 167–174. ISBN 0-7695-2893-7. ISSN 1093-0175.

NAVARRO, E. O.; HOEK, A. van der. Comprehensive Evaluation of an Educational Software Engineering Simulation Environment. In: *CSEET 2007 - Proceedings of the 20th Conference on Software Engineering Education & Training*. Dublin: [s.n.], 2007. p. 195–202. ISBN 0-7695-2893-7. ISSN 1093-0175.

NEWBY, T. J. et al. *Instructional technology for teaching and learning: Designing instruction, integrating computers, and using media*. [S.l.]: Merrill/Prentice-Hall, 1996.

NIU. Northern Illinois University Faculty Development and Instructional Design Center. Experiential Learning. In: *Instructional Guide for University Faculty and Teaching Assistants*. 2012. p. 1–9. Disponível em: ⟨http://www.niu.edu/facdev/resources/guide⟩.

OREG, S.; NOV, O. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, Elsevier Science Publishers B. V., v. 24, n. 5, p. 2055–2073, sep 2008. ISSN 07475632.

PAPADOPOULOS, P. M.; STAMELOS, I. G.; MEISZNER, A. Students' Perspectives on Learning Software Engineering with Open Source Projects: Lessons Learnt After Three Years of Program Operation. In: *4th International Conference on Computer Supported Education – CSEDU*. [S.l.: s.n.], 2012. p. 313–322.

PARNAS, D. Software engineering programs are not computer science programs. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 16, n. 6, p. 19–30, 1999. ISSN 07407459.

PARNAS, D. *Software Engineering, Why And What*. Maceió, Br: [s.n.], 2014.

PETERSEN, K. et al. Systematic Mapping Studies in Software Engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*. [S.l.: s.n.], 2008. p. 68–77.

PETRENKO, M. et al. Teaching Software Evolution in Open Source. *Computer*, IEEE Computer Society, v. 40, n. 11, p. 25–31, nov 2007. ISSN 0018-9162.

QIAN, K.; FU, X. Teaching Component-Based Software Development. In: *2008 21st IEEE-CS Conference on Software Engineering Education and Training Workshop*. [S.l.]: IEEE, 2008. p. 13–15. ISBN 978-0-7695-3387-2.

RADERMACHER, A.; WALIA, G. Gaps between industry expectations and the abilities of graduates. In: *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. Denver, Colorado, USA: ACM Press, 2013. p. 525. ISBN 9781450318686.

RADERMACHER, A.; WALIA, G.; KNUDSON, D. Investigating the skill gap between graduating students and industry expectations. In: *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*. Hyderabad, India: ACM Press, 2014. p. 291–300. ISBN 9781450327688.

RAJ, R.; KAZEMIAN, F. Using Open Source Software in Computer Science Courses. In: *36th Annual Frontiers in Education Conference (FIE)*. [S.l.]: IEEE, 2006. p. 21–26. ISBN 1-4244-0256-5.

RAJLICH, V. *Software Engineering: The Current Practice*. [S.l.]: CRC Press, 2012. 291 p.

RAJLICH, V. Teaching developer skills in the first software engineering course. In: ... *(ICSE), 2013 35th International Conference on*. [s.n.], 2013. p. 1109–1116. Disponível em: ⟨http://www.cs.wayne.edu/rajlich/publications/Rajlich2013TeachingDeveloperSkills.pdf⟩.

RAY, J. J. Reviving the Problem of Acquiescent Response Bias. *The Journal of Social Psychology*, v. 121, n. 1, p. 81–96, 1983. ISSN 0022-4545.

REICHLMAYR, T. J. Collaborating with Industry: Strategies for an Undergraduate Software Engineering Program. In: *International Workshop on Summit on Software Engineering Education (SSEE'06)*. New York, NY, USA: ACM, 2006. p. 13–16.

REKHA, V. S. et al. Bridging the Computer Science Skill Gap with Free and Open Source Software. In: *International Conference on Engineering Education (ICEED)*. [S.l.: s.n.], 2009. p. 77–82.

ROBLES, G.; CABALLÉ, S.; GONZÁLEZ-BARAHONA, J. Teaching Software Development in Community-Driven Software Projects- A Practical Experience. In: *Free Knowledge Free Technology. The SELF Conference*. Barcelona: [s.n.], 2008. Disponível em: ⟨http://calistay.inet-tr.org.tr/SELF/FKFT/robles.pdf⟩.

SALOMON, G.; PERKINS, D. N. Individual and Social Aspects of Learning. *Review of Research in Education*, American Educational Research Association, v. 23, p. 1–24, 1998.

SAMUELSON, P. IBM's Pragmatic Embrace of Open Source. *Communications of the ACM*, New York, NY, USA, p. 21–25, 2006.

SANTORE, J. et al. The Software Engineering Class Builds a GUI for Subversion. *ACM SIGCSE Bulletin*, v. 41, n. 4, p. 82—-84, jan 2010. ISSN 00978418.

SAVI, R. *Avaliação de Jogos Voltados para a Disseminação do Conhecimento*. 238 p. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2011.

SBC. Sociedade Brasileira da Computação. *Currículo de Referência para Cursos de Graduação em Bacharelado em Ciência da Computação e Engenharia da Computação*. 2005. Disponível em: ⟨http://www.sbc.org.br/index.php?option=com_jdownloads&Itemid=195&task=view.download&catid=36&cid=183⟩.

SCHUNK, D. H. *Learning Theories: An education perspective*. [S.l.]: Merrill/Prentice-Hall, 1991. 402 p.

SEITER, L. M. Computer Science and Service Learning: Empowering Nonprofit Organizations through Open Source Content Management Systems. In: *Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium*. Chattanooga: [s.n.], 2009. Disponível em: ⟨http://www.hfoss.org/symposium09/documentdl/papers/HFOSS{_}Symp-final28.⟩

SHAW, M. Software engineering education: a roadmap. In: *Proceedings of the conference on The future of Software engineering - ICSE '00*. New York, New York, USA: ACM Press, 2000. p. 371–380. ISBN 1581132530.

SILBERMAN, M. *Active Learning: 101 strategies to teach any subject*. [S.l.]: Allyn and Bacon, 1996. 189 p.

SIMON, B.; PARRIS, J.; SPACCO, J. How we teach impacts student learning. In: *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, 2013. p. 41. ISBN 9781450318686.

SMITH, T. M. et al. Selecting open source software projects to teach software engineering. In: *SIGCSE '14 Proceedings of the 45th ACM technical symposium on Computer science education*. [S.l.: s.n.], 2014. p. 397–402.

SOWE, S.; STAMELOS, I.; DELIGIANNIS, I. A Framework for Teaching Software Testing using F/OSS Methodology. In: *Open Source Systems*. [S.l.: s.n.], 2006. v. 203, p. 261–266.

SOWE, S. K.; STAMELOS, I. Involving Software Engineering Students in Open Source Software Projects: Experiences from a Pilot Study. *Journal of Information Systems Education (JISE)*, v. 18, n. 4, p. 425–435, 2007.

SPINELLIS, D. Open Source and Professional Advancement. *IEEE Software*, v. 23, n. 5, p. 70–71, sep 2006. ISSN 0740-7459.

STAMELOS, I. Teaching Software Engineering with Free/Libre Open Source Projects. *International Journal of Open Source Software and Processes*, IGI Global, v. 1, n. 1, p. 72–90, jan 2009. ISSN 1942-3926. Disponível em: ⟨http://www.igi-global.com/article/teaching-software-engineering-free-libre/2772⟩.

STEINMACHER, I. et al. The hard life of open source software project newcomers. In: *Proceedings of the 7th international workshop on cooperative and human aspects of software engineering*. [S.l.]: ACM, 2014. p. 72–78.

STROULIA, E. et al. Teaching Distributed Software Engineering with UCOSP: the Undergraduate Capstone Open-Source Project. In: *Proceeding of The Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD '11)*. New York, New York, USA: ACM Press, 2011. p. 20–25. ISBN 9781450305907.

SVINICKI, M.; MCKEACHIE, W. J. *McKeachie's Teaching Tips: Strategies, Research, and Theory for College and University Teachers*. 13 ed.. ed. [S.l.]: Wadsworth Cengage Learning, 2011. 388 p.

SZABO, C. Student Projects Are Not Throwaways: Teaching Practical Software Maintenance in a Software Engineering Course. In: *Proceedings of the 45th ACM technical symposium on Computer science education*. [S.l.: s.n.], 2014. p. 55–60.

TAO, Y.; NANDIGAM, J. Work in Progress: Open Source Software as the Basis of Developing Software Design Case Studies. In: *36th Annual Conference Frontiers in Education (FIE)*. [S.l.]: IEEE, 2006. p. 27–28. ISBN 1-4244-0256-5.

TOTH, K. Experiences with Open Source Software Engineering Tools. *IEEE Software*, v. 23, n. 6, p. 44–52, nov 2006. ISSN 0740-7459.

TUCKER, A. B. Teaching client-driven software development. *Journal of Computing Sciences in Colleges*, v. 24, n. 4, p. 29–39, apr 2009. ISSN 1937-4771.

TVEDT, J. D.; TESORIERO, R.; GARY, K. A. The Software Factory: An Undergraduate Computer Science Curriculum. *Computer Science Education*, v. 2, n. 1-2, p. 91–117, 2002.

UDEN, L.; BEAUMONT, C. *Technology and Problem-Based Learning*. London: Information Science Publishing, 2006. 344 p.

UNM. University of New Mexico School of Medicine. *Effective Use of Performance Objectives for Learning and Assessment*. 2005. 6 p. Disponível em: ⟨http://ccoe.rbhs.rutgers.edu/forms/EffectiveUseofLearningObjectives.pdf⟩.

WHITEHURST, J. Open Source: Narrowing the Divide between Education Business and Community. *EDUCAUSE Review*, p. 70–71, 2009. Disponível em: ⟨http://www.educause.edu/ero/article/open-source-narrowing-divide-between-education-business-and-community⟩.

WILLIAMS, L.; SHIN, Y. Work in Progress: Exploring Security and Privacy Concepts through the Development and Testing of the iTrust Medical Records System. In: *36th Annual Frontiers in Education Conference (FIE)*. [S.l.]: IEEE, 2006. p. 30–31. ISBN 1-4244-0256-5.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer Berlin Heidelberg, 2012.

XING, G. Teaching Software Engineering Using Open Source Software. In: *Proceedings of the 48th Annual Southeast Regional Conference on (ACM SE '10)*. New York, New York, USA: ACM Press, 2010. ISBN 9781450300643.

YAMAKAMI, T. Re-engineering Software Education: OSS-aware Software Education in the Era of Utilizing External Resources. *International Conference on Advanced Communication Technology, ICACT*, 2012.

YIN, R. K. *Estudo de Caso: Planejamento e Métodos*. 4 ed.. ed. Porto Alegre: Bookman, 2010. 248 p.

ZHANG, H.; SU, H. A Collaborative System for Software Engineering Education. In: *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*. [S.l.]: IEEE, 2007. p. 313–318. ISBN 0-7695-2870-8. ISSN 0730-3157.

APÊNDICE A

CONJUNTO DE OBJETIVOS DE APRENDIZAGEM EM ENGENHARIA DE SOFTWARE

A Tabela A.1 lista os 320 objetivos de aprendizagem em engenharia de software estabelecidos a partir das perspectivas: acadêmica, indústria, literatura relacionada e proposta pessoal. Para cada um dos objetivos, apresentamos sua descrição, as referências utilizadas para a sua formulação, o nível de aprendizagem pretendido, o grau de impacto da adoção da abordagem para o alcance do objetivo de aprendizagem e algumas atividades possíveis de serem realizadas com o emprego de OSP, sugeridas no intuito de justificar o grau de impacto atribuído.

Para especificar 'a' ou 'as' referências utilizadas para a formulação do objetivo, utilizamos quatro colunas: 'CS2013', 'Indústria', 'Literatura' e 'P', esta última quando representa uma proposta pessoal.

A coluna 'CS2013' significa que o objetivo de aprendizagem provém do currículo de referência, sendo que pode estar preenchida com as seguintes informações:

(CSLO) *Computer Science Learning Outcome* quando provém de um resultado de aprendizagem de determinada unidade de conhecimento;

(CSCT) *Computer Science Curriculum Topic* quando provém de um tópico de determinada unidade de conhecimento;

(CSCG) *Computer Science Characteristics of Graduates* quando provém das características esperadas para os alunos recém-formados.

Para os objetivos provenientes de CSLO e CSCT, a unidade de conhecimento também encontra-se assinalada pela respectiva sigla, bem como, a exigência especificada pelo currículo para a sua cobertura.

Siglas das unidades de conhecimento:

(RE) *Requirements Engineering*;

(SD) *Software Design*;

(SC) *Software Construction*;

(V&V) *Software Verification and Validation*;

(SE) *Software Evolution*;

(SP) *Software Process*;

(SPM) *Software Project Management*;

(T&E) *Software Tools and Environments*;

(SR) *Software Reliability*;

(FM) *Formal Methods*;

(SDF) *Software Development Fundamentals*;

(SSE) *Secure Software Engineering*;

(PC) *professional communication*;

(S) *Sustainability*;

(UCDT) *User-Centered Design & Testing*;

(PE) *Professional Ethics*;

(IP) *Intellectual Property*;

(EC) *Economies of Computing*;

(HCIF) *Human-Computer Interaction Foundations*;

(DI) *Designing Interaction*;

(PIS) *Programming Interactive Systems.*

Exigência de cobertura:

(c1) *Core Tier-1* corresponde a tópicos ou resultados de aprendizagem que devem ser cobertos durante o curso, tipicamente em disciplinas introdutórias;

(c2) *Core Tier-2* representa tópicos ou resultados de aprendizagem geralmente essenciais, que devem ser cobertos em sua maioria. O curso deve cobrir de 90 a 100% desse material, sendo o mínimo aceitável de 80%;

(e) *Elective* abrange tópicos ou resultados de aprendizagem avançados, de aprofundamento das várias áreas de especialização. Muitos cursos não cobrem todo esse material.

A coluna 'Indústria' sinaliza que o objetivo de aprendizagem representa uma deficiência apontada pela indústria, enquanto que, a coluna 'Literatura' expressa que algum ou alguns estudos indicaram a adoção da abordagem OSP (*Open Source Project*) para o atendimento do mesmo. Nestas duas situações, as colunas estão preenchidas com as referências encontradas.

A coluna 'Nível' denota o nível de aprendizagem pretendido com relação ao objetivo e segue a taxonomia definida pelo currículo de referência: 'familiaridade', 'uso' e 'avaliação'.

A coluna 'G' designa o grau de favorecimento de utilização da abordagem para o alcance do objetivo em questão. Os valores atribuídos variam de zero à três (do não favorecimento, ao máximo favorecimento).

Enfim, a coluna 'Justificativas' representa possíveis atividades a serem realizadas com a adoção de OSP, visando atender o objetivo de aprendizagem em questão. O intuito não é explorar todas as possibilidades de utilização, mas apenas apresentar alguma ou algumas oportunidades de emprego da abordagem que possam justificar o grau de impacto atribuído. Nesta coluna, todas as vezes que palavra 'projeto' for usada, estamos nos referindo a um projeto OSP.

A descrição dos níveis de aprendizagem e dos possíveis graus de favorecimento (impacto), bem como, o processo de estabelecimento dos objetivos de aprendizagem, encontram-se detalhados no Capítulo 3 e no Capítulo 4.

Os objetivos também encontram-se classificados por área de conhecimento do SWEBOK.

Tabela A.1: Objetivos de Aprendizagem em ES

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
Requisitos de Software								
LO1	Identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.	CSLO [RE-c2]				Uso	3	Identificar quais são os RF e os RNF do projeto a partir do uso do software. Identificar quais são os RF e os RNF do projeto a partir da documentação existente. Identificar RF e RNF a partir do código-fonte do projeto. Identificar RF e RNF a partir das solicitações existentes no *issue tracker* do projeto.
LO2	Descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.	CSLO [RE-c1]				Familiaridade	2	Entender como é o processo de engenharia de requisitos em um OSP. Identificar exemplos do processo de elicitação/ especificação dos requisitos no *issue tracker* do projeto. Descrever qual seria o processo de eng. de requisitos, para uma nova funcionalidade proposta.
LO3	Aplicar os principais métodos e técnicas de elicitação e análise de requisitos para na especificação de um software de médio porte	CSLO [RE-e]	(RADERMACHER; WALIA, 2013)	(LI; ZHANG; LI, 2009) (ELLIS et al., 2007) (AUER; JUNTUNEN; OJALA, 2011)		Uso	1	Muitas técnicas de elicitação propõem a comunicação face-a-face, o que usualmente não ocorre em OSP. Dada as características do OSP, descrever quais técnicas de elicitação de requisitos podem ser aplicadas. Identificar no *issue tracker* alguma nova funcionalidade e interagir com a comunidade para descrever essa nova funcionalidade incorporando-a as funcionalidade já existentes. Analisar o software localmente, propor e especificar uma nova funcionalidade para a comunidade, integrada com as funcionalidades existentes.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO4	Comparar as abordagens dirigida por planos (tradicional, prescritiva) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.	CSLO [RE-e]				Familiaridade	1	Comparar a engenharia de requisitos no método tradicional, ágil e OSP.
LO89	Aplicar a técnica de entrevistas para elicitar requisitos junto aos usuários finais.		(RADERMACHER; WALIA, 2013)			Uso	0	
LO5	Listar os principais componentes de um caso de uso ou descrição similar para algum comportamento requerido de um software.	CSLO [RE-c1]				Uso	3	Criar ou atualizar o diagrama de casos de uso para o projeto. Descrever um caso de uso para uma funcionalidade importante do projeto. Propor novos casos de uso para o projeto (sendo baseados nos comentários da comunidade ou não).
LO6	Usar método comum (não formal) para modelar e especificar os requisitos de software de médio porte.	CSLO [RE-e]	(RADERMACHER; WALIA, 2013)	(STAMELOS, 2009) (PAPADOPOULOS; STAMELOS; MEISZNER, 2012) (XING, 2010) (MARMORSTEIN, 2011) (LI; ZHANG; LI, 2009)		Uso	3	Produzir, complementar ou atualizar a documentação dos requisitos do projeto (usar o software, criar ou atualizar especificações, usar *templates* porventura existentes, incluir diagramas).
LO7	Diferenciar o rastreamento de 'construção' (*forward*) e 'recuperação (*backward*) e explicar seus papeis no processo de validação dos requisitos.	CSLO [RE-e]				Familiaridade	3	Propor atividades para rastrear se os requisitos projetados na documentação existente foram implementados. Propor atividades de recuperar informações do projeto e comparar com aquilo que foi inicialmente projetado.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO8	Descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos				P	Familiaridade	3	Identificar o processo de controle de mudanças de requisitos existente através do *issue tracker*. Para alguma *issue* de mudança existente, verificar no código todas as mudanças realizadas e problemas ocorridos durante o processo. Discutir os problemas porventura existentes.
LO9	Conduzir uma revisão de um conjunto de requisitos de software para determinar a qualidade dos mesmos, com respeito à consistência, validade, completude e viabilidade.	CSLO [RE-c2]		(STAMELOS, 2009)		Uso	3	Passar checklists de revisão para que possam ser verificados no projeto. Propor o registro dos problemas identificados. Realizar a inspeção dos requisitos sobre alguma funcionalidade especificada localmente.
LO10	Demonstrar a capacidade de usar ferramentas de análise de requisitos em suporte ao desenvolvimento de um software de médio porte.	CSLO [T&E-c2]				Uso	3	Usar ferramentas para documentar o projeto.
LO274	Descrever os desafios fundamentais de técnicas comuns usadas para a elicitação de requisitos.	CSLO [RE-c2]				Familiaridade	0	
LO275	Criar um protótipo de software para atenuar riscos relacionados aos requisitos.	CSLO [RE-e]				Uso	0	
LO276	Usar técnicas de prototipação de baixa fidelidade para reunir e reportar respostas do usuário.	CSLO [UCDT-e]				Uso	0	
LO277	Para um grupo identificado de usuários, obter e documentar uma análise de suas necessidades de interface.	CSLO [DI-c2]				Avaliação	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO278	Traduzir a especificação de requisitos de software escrita em linguagem formal para a linguagem natural.	CSLO [RE-e]				Uso	0	
Design de Software								
LO52	Enfatizar a importância do design do software.			(XING, 2010)		Familiaridade	3	Apresentar exemplos de interfaces de usuários mal estruturadas, de alto acoplamento e baixa coesão entre os componentes do projeto, situações de intolerância a falhas, etc. Discutir as consequências decorrentes desses problemas. Indicar melhorias possíveis. Identificar no *issue tracker* discussões que podem ter sido causadas por problemas no design do software. Analisar o código que motivou a discussão e solicitar aos alunos que sugiram soluções para melhoria.
LO12	Articular princípios de design incluindo separação de interesses (*concerns*), ocultação de informações, acoplamento, coesão, encapsulamento.	CSLO [SD-c1]		(CARRINGTON, 2003)		Familiaridade	3	Apresentar exemplos extraídos do código. Solicitar que os alunos busquem no código exemplos de cada um desses princípios.
LO13	Explicar a relação entre os requisitos do software e seu design, usando os modelos apropriados.	CSLO [SD-c2]				Avaliação	3	Identificar no projeto como os requisitos foram projetados e implementados no código. Criar ou atualizar os modelos, de modo que estes reflitam o que foi implementado. Discutir a importância do design.
LO14	Usar um determinado paradigma de design, para projetar um software simples, e explicar como os princípios de design foram aplicados neste projeto.	CSLO [SD-c1]	(RADERMACHER; WALIA, 2013)	(ELLIS et al., 2007)		Uso	3	Identificar o paradigma de design utilizado no projeto e como os princípios de design foram aplicados nesse caso. Projetar uma nova funcionalidade para o projeto.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO15	Projetar código módular.			(GHOSH; GLOTT, 2005)		Uso	3	Usar ferramentas que coletem métricas para identificar possíveis problemas de modularidade no código. Realizar refatorações no código para melhorar a modularidade. Usar novamente as ferramentas de métricas e comparar com os resultados obtidos na primeira execução. Projetar uma nova funcionalidade para o projeto, com código modularizado.
LO16	Projetar uma solução de persistência para uma aplicação de médio porte.				P	Uso	3	Verificar a solução de persistência adotada no projeto. Aplicar a solução de persistência em uma nova funcionalidade. Comparar e discutir a solução de persistência adotada em projetos OS distintos.
LO17	Explicar e usar conceitos de programação de interfaces gráficas de usuários (GUI): tratamento de eventos, gerenciamento de layout baseado em restrições.	CSLO [PIS-e]				Uso	3	Verificar como os conceitos de programação de GUI foram implementados no projeto. Aplicar a solução para projetar uma nova funcionalidade. Comparar e discutir soluções adotadas em diferentes projetos OS.
LO18	Projetar uma solução de tratamento de erros e exceções para uma aplicação de médio porte .				P	Uso	3	Verificar como o tratamento de erros e exceções foram implementados no projeto. Verificar a abrangência dos erros tratados. Caso algum erro não esteja sendo tratado, projetar a incorporação desse tratamento no projeto. Aplicar a solução para projetar uma nova funcionalidade. Comparar e discutir a solução adotada em diferentes projetos OS. Ou, projetar uma solução para um OSP que não possua tratamento de erros e exceções.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO19	Identificar modos de aplicar redundância para alcançar tolerância a falhas em um software de médio porte.	CSLO [SR-e]				Uso	3	Verificar se o projeto utiliza alguma forma de redundância para alcançar tolerância a falhas. Se existir, discutir as vantagens e desvantagens dessa solução. Se não existir, projetar uma solução de redundância apropriada ao projeto. Ou, identificar em alguns projetos OS como são aplicadas redundância. Projetar e implementar uma solução de redundância para um projeto que não possui.
LO20	Aplicar os conceitos de: Princípios de Design para Mecanismos de Proteção; Princípios para Segurança de Software; e Princípios para Design Seguro no desenvolvimento de um projeto de software.	CSLO [SSE-e]				Uso	3	Verificar se o OSP atende a estes princípios. No caso afirmativo, verificar como a solução foi projetada. No caso negativo, projetar uma solução. Ou, avaliar projetos OS existentes e sugerir como os conceitos sejam aplicados.
LO21	No contexto de um único paradigma de design, descrever um ou mais padrões de projeto que poderiam ser aplicados para projetar um software simples.	CSLO [SD-c1]		(CARRINGTON, 2003) (KAMTHAN, 2007) (TAO; NANDIGAM, 2006)		Familiaridade	3	Identificar padrões de projeto aplicados no OSP. Identificar onde padrões de projeto poderiam ser aplicados.
LO22	Descrever a arquitetura de um software, no contexto de um único paradigma de design.	CSLO [SD-c2]		(COSTA-SORIA; PÉREZ, 2009) (NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008)		Familiaridade	3	Identificar o paradigma de design aplicado no projeto. Modelar e Descrever a arquitetura do projeto no contexto do paradigma aplicado. Ou, modelar e descrever a arquitetura de alguns projetos OS selecionados.
LO23	Para um software adequado a um dado cenário, discutir e selecionar o paradigma de design apropriado.	CSLO [SD-c2]				Uso	3	Identificar o paradigma de design aplicado no projeto. Discutir as vantagens e desvantagens do paradigma utilizado. Verificar a aplicação de um outro paradigma.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO24	Aplicar exemplos simples de padrões em um projeto de software.	CSLO [SD-c2]				Uso	3	Identificar onde padrões de projeto poderiam ser aplicados. Refatorar o código para aplicar os padrões.
LO25	Discutir e selecionar a arquitetura apropriada para um software simples, adequada para um dado cenário.	CSLO [SD-e]		(AUER; JUNTUNEN; OJALA, 2011)		Uso	2	Descrever a arquitetura empregada no projeto. Discutir as vantagens e desvantagens da arquitetura utilizada. Verificar a aplicação de uma outra arquitetura. (A atividade é prejudicada porque o software já está pronto. Começar uma arquitetura do zero é sempre um grande desafio.)
LO26	Dado um projeto de alto nível, identificar a arquitetura diferenciando entre arquiteturas comuns: três camadas, cliente-servidor e *pipe-and-filter*.	CSLO [SD-c2]				Familiaridade	3	Apresentar exemplos de projetos OS com os estilos arquiteturais: em camadas, cliente-servidor e *pipe-and-filter*. O aluno deverá identificar o estilo arquitetural empregado em cada OSP e descrever como o estilo foi implementado. Comparar e discutir os objetivos, vantagens e desvantagens de cada estilo.
LO27	Investigar o impacto da seleção da arquitetura para o design do projeto.	CSLO [SD-c2]				Avaliação	3	Descrever a arquitetura empregada no projeto. Discutir o impacto dessa arquitetura para o design do projeto. Ou, discutir como arquiteturas de projetos OS distintos evoluíram, em versões diferentes do software.
LO28	Identificar métodos que conduzem a uma arquitetura de software que alcança um nível específico de confiabilidade.	CSLO [SR-e]				Uso	2	Discutir o nível de confiabilidade proporcionado pela arquitetura do projeto. Tentar identificar os mecanismos de confiabilidade criados nas arquiteturas de outros projetos OS. Discutir e implementar melhorias no projeto em questão.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO29	Selecionar componentes adequados para uso no design de um software.	CSLO [SD-c2]				Uso	3	Identificar os componentes utilizados em um ou mais projetos OS. Para determinado OSP, buscar alternativas de outros componentes que poderiam ser utilizados em substituição aos usados atualmente.
LO30	Explicar como componentes adequados poderiam ser adaptados para uso no design de um software.	CSLO [SD-c2]		(SEITER, 2009)		Familiaridade	3	Identificar componentes que poderiam ser utilizados no projeto. Descrever como esses componentes poderiam ser adaptados ao projeto. Ou para o nível de 'uso', solicitar que o componente seja substituído na arquitetura. E que todas as adaptações sejam documentadas para posterior discussão.
LO31	Projetar um contrato para o uso de um componente típico pequeno em um dado software.	CSLO [SD-c2]				Uso	3	Identificar os componentes utilizados no projeto. Descrever o contrato de uso para um desses componentes. Ou, identificar componentes que poderiam ser utilizados no projeto. Descrever um contrato para um desses componentes.
LO32	Descrever como *frameworks* de aplicação são empregados no projeto de software.	CSCT [SD-e]				Familiaridade	3	Descrever como o *framework* foi empregado no software.
LO75	Projetar e implementar um software para a plataforma *mobile*.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Descrever a arquitetura e o design detalhado do projeto. Projetar e construir uma nova funcionalidade para o software.
LO279	Discutir porque o desenvolvimento de software centrado na interface humano-computador é importante.	CSLO [HCIF-c1]				Familiaridade	2	Discutir problemas de usabilidade da interface do projeto. Discutir as possíveis causas para estes problemas. Identificar entradas no *issue tracker* que revelam solicitações de mudança em virtude de características do usuário do software não terem sido consideradas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO280	Sumarizar os preceitos psicológicos e de interação social básicos.	CSLO [HCIF-c1]				Familiaridade	0	
LO281	Definir o processo de design de interface centrado no usuário, que considere explicitamente o fato de que o usuário não é como os desenvolvedores, considerando a facilidade de aprendizagem de uso de uma nova interface destes últimos.	CSLO [HCIF-c1]				Uso	0	
LO53	Desenvolver e usar vocabulários conceituais para analisar a interação humana com o software: necessidades, modelos conceituais, *feedback*, etc.	CSLO [HCIF-c1]				Uso	3	Analisar o vocabulário empregado na interface do projeto. Identificar pontos da interface no qual o vocabulário não está adequado. Propor mudanças apropriadas para o vocabulário.
LO56	Explicar a importância do padrão *Model-View-Controller* (MVC) para a programação de interface.	CSLO [PIS-e]				Familiaridade	3	Verificar se o OSP utiliza o padrão MVC para a programação de interface. Solicitar que uma nova interface seja projetada para o software. Descrever o que é necessário realizar para integrar essa nova interface com o restante do código. Discutir as vantagens e desvantagens da utilização do MVC.
LO57	Identificar semelhanças e diferenças entre interfaces de usuários de diferentes plataformas.	CSLO [PIS-e]				Familiaridade	3	Comparar interfaces de diferentes plataformas em projetos OS distintos. Ex: projeto Eclipse e projeto Android.
LO33	Criar uma aplicação simples com interface gráfica de usuário, incluindo ajuda (*help*) e documentação.	CSLO [DI-c2]		(AUER; JUNTUNEN; OJALA, 2011) (SANTORE et al., 2010) (CHEN et al., 2008) (HEPTING et al., 2008)		Uso	3	Identificar como a interface do projeto está implementada. Caso não existam, adicionar a 'ajuda' e a documentação. Caso não estejam atualizados, atualizar a 'ajuda' e a documentação. Projetar e implementar uma nova interface, incluindo a 'ajuda' e a documentação.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO320	Criar uma aplicação com uma interface de usuário moderna.	CSLO [PIS-e]				Uso	3	Semelhante ao LO33
LO35	Projetar a localização e internacionalização de uma aplicação.			(AUER; JUNTUNEN; OJALA, 2011)		Uso	3	Apresentar um projeto que exista localização e internacionalização. Projetar a localização e internacionalização para um projeto OSP que não possua. Ou, identificar e comparar como a solução de localização e internacionalização foi implementada em diferentes projetos OS. Projetar uma solução para um projeto OSP que não possua.
LO36	Listar os componentes principais de um modelo de dados.	CSLO [RE-c2]				Familiaridade	3	Entender o modelo de dados do projeto. Discutir o modelo de dados do projeto.
LO37	Construir os modelos de design adequados ao paradigma utilizado para projetar um software simples.	CSLO [SD-c1]	(RADERMACHER; WALIA, 2013)	(STAMELOS, 2009)		Uso	3	Construir ou atualizar os modelos de design do projeto. Modelar uma nova funcionalidades, incorporando-a aos modelos de design existentes.
LO38	Construir os modelos apropriados para modelar a estrutura e o comportamento que atendam as especificações dos requisitos do software.	CSLO [SD-c2]	(RADERMACHER; WALIA, 2013)	(STAMELOS, 2009) (MCCARTNEY; GOKHALE; SMITH, 2012)		Uso	3	Construir ou atualizar os modelos de estrutura e comportamento do projeto. Modelar uma nova funcionalidades, incorporando-a aos modelos de design existentes. (muito próximo ao LO37)
LO40	Analisar o design do software da perspectiva de atributos internos importantes de qualidade.	CSLO [SD-e]		(STAMELOS, 2009) (KAMTHAN, 2007) (MARMORSTEIN, 2011)		Avaliação	3	Analisar o design do projeto da perspectiva de atributos internos importantes de qualidade. Usar ferramentas para análise dos atributos internos de qualidade. Discutir os resultados. Descrever possíveis melhorias.
LO41	Analisar o design do software da perspectiva de atributos externos importantes de qualidade.	CSLO [SD-e]		(MARMORSTEIN, 2011)		Avaliação	3	Analisar o design do projeto da perspectiva de atributos externos importantes de qualidade. Usar ferramentas para análise dos atributos externos de qualidade. Discutir os resultados. Descrever possíveis melhorias.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO42	Aplicar modelos para o design de atributos de qualidade internos e externos em um componente de software, a fim de alcançar negociações aceitáveis entre aspectos de qualidade conflitantes.	CSLO [SD-e]				Uso	3	Verificar como os atributos de qualidade (internos e externos) conflitantes foram atendidos no projeto. Discutir o que certas modificações que poderiam ser feitas para melhorar um atributo, causariam no outro atributo.
LO43	Aplicar o desenvolvimento baseado em componentes para projetar uma parte do software, como por exemplo: concorrência e transações; serviços de comunicação confiáveis; interações com o banco de dados, incluindo consultas remotas, gerenciamento, ou acesso e comunicação segura.	CSLO [SD-e]		(QIAN; FU, 2008)		Uso	2	Verificar se componentes estão sendo utilizados no projeto. Verificar quais componentes poderiam ser utilizados.
LO44	Aplicar o paradigma OO na análise e projeto de um software de médio porte.	CSCT [SD-c1]	(RADERMACHER; WALIA, 2013)	(KUME; NITTA; TAKEMURA, 2006) (CHEN et al., 2008)		Uso	3	Descrever o design do projeto. Projetar uma nova funcionalidade para a aplicação.
LO46	Ter noção sobre o desenvolvimento orientado a aspectos.	CSCT [SD-c1]		(COSTA-SORIA; PÉREZ, 2009)		Familiaridade	3	Apresentar um projeto onde a orientação a aspectos seja utilizada. Ou, Descrever como a orientação a aspectos poderia ser aplicada no projeto.
LO47	Demonstrar a capacidade de usar ferramentas de modelagem de design em suporte ao desenvolvimento de um projeto de médio porte.	CSLO [T&E-c2]				Uso	3	Usar a ferramenta para criar ou atualizar os modelos de design do projeto. Usar a ferramenta para modelar uma nova funcionalidade para o projeto.
LO282	Explicar o papel dos objetos em sistemas de *middleware* e a relação com componentes.	CSLO [SD-e]				Familiaridade	1	Analisar sistemas de *middleware* para entender as relações entre objetos e componentes.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO283	Estabelecer e aplicar os princípios de menos privilégio e padrões de falha segura.	CSLO [SD-e]				Familiaridade	3	Analisar os privilégios do usuário em vários componentes da arquitetura. Por exemplo, o privilégio de acesso ao banco de dados e as configurações existentes para o caso de falhas nas operações.
LO284	Desenvolver especificações para o esforço de desenvolvimento de software que especifique completamente os requisitos funcionais e identifique os caminhos de execução esperados.	CSLO [SSE-e]				Uso	3	Modelar os caminhos de execução esperados para alguma funcionalidade do projeto.
Construção do Software								
LO48	Aplicar estilos de programação padrões e documentação consistente que contribuam para a confiabilidade e manutenibilidade do software.	CSLO [SDF-c1]		(NANDIGAM; GUDIVADA; HAMOU-LHADJ, 2008) (GHOSH; GLOTT, 2005)		Uso	3	Implementar alguma nova funcionalidade. Explicar exemplos de código retirados do projeto onde nenhum estilo de programação ou documentação foi aplicado. Avaliar a dificuldade de entender o código. Explicar exemplos de código retirados do projeto onde estilos de programação ou documentações tenham sido aplicados. Avaliar a dificuldade de entender o código. Comparar o código implementado com os exemplos dados. Caso necessário, melhorar as próprias implementações. Avaliar estilos de programação e documentação existente em diferentes projetos OS. Discutir vantagens e desvantagens em relação à confiabilidade e manutenibilidade.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO49	Selecionar e usar um dos padrões de codificação existentes em um pequeno projeto de software.	CSLO [SC-c2]	(RADERMACHER; WALIA; KNUDSON, 2014)	(REKHA et al., 2009) (AUER; JUNTUNEN; OJALA, 2011)		Uso	3	Avaliar os estilos de programação empregados no projeto. Comparar com alguns estilos padrões existentes. Configurar os padrões de codificação do projeto na IDE de desenvolvimento. Implementar uma nova funcionalidade no projeto usando os padrões de codificação adotados pelo mesmo.
LO50	Discutir as vantagens e desvantagens de diferentes tipos de reuso de software.	CSLO [SE-c2]		(REKHA et al., 2009)		Familiaridade	3	Apresentar exemplos de diferentes tipos de reuso de software retirados de diversos projetos OS. A partir dos exemplos extraídos, discutir as vantagens e desvantagens de cada tipo.
LO60	Analisar a extensão que o código do outro programador atende aos padrões de documentação e estilos de programação.	CSLO [SDF-c1]				Avaliação	3	Avaliar o quanto do código do projeto atende aos padrões de documentação e estilos de programação.
LO6	Vivenciar um processo de verificação e aprovação do código construído.			(PAPADOPOULOS; STAMELOS; MEISZNER, 2012) (ELLIS et al., 2007)		Familiaridade	3	Identificar nos instrumentos de comunicação e no controle de versões, contribuições de código realizadas por outros desenvolvedores e as respostas dos desenvolvedores *core* da comunidade. Fazer alguma contribuição de código para a comunidade do projeto e analisar a resposta. Modelar o processo de verificação e aprovação do código.
LO62	Comentar e documentar o código apropriadamente.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Solicitar que os alunos avaliem os comentários e a documentação existente no código do projeto. Comentar e documentar algum código do projeto, onde for necessário. Avaliar estilos de comentários em diferentes projetos OS.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO63	Demonstrar habilidade em programação.		(RADERMACHER; WALIA, 2013)	(GHOSH; GLOTT, 2005) (HISLOP; ELLIS; MORELLI, 2009) (AUER; JUNTUNEN; OJALA, 2011) (MARMORSTEIN, 2011) (XING, 2010) (CARRINGTON, 2003)		Uso	3	Solicitar que os alunos expliquem partes do código do projeto. Solicitar que os alunos codifiquem novas funcionalidades para o projeto.
LO64	Identificar erros de codificação comuns que levam a construção de programas inseguros (por exemplo *buffer overflows*, *memory leaks*, código malicioso) e aplicar estratégias para evitar esses erros.	CSLO [SDF-c1]				Uso	3	Identificar erros comuns de codificação no projeto. Usar ferramentas automáticas que identifiquem esses problemas no código. Identificar e discutir soluções que poderiam ser aplicadas no código para evitar o problema. Ou, localmente gerar esses erros em exemplos de código do projeto e solicitar que os alunos corrijam esses erros.
LO72	Reescrever um programa simples para remover vulnerabilidades comuns, como *buffer overflows*, *integer overflows* e *race conditions*.	CSLO [SC-e]				Uso	3	Identificar erros comuns de codificação no projeto. Usar ferramentas automáticas que identifiquem esses problemas no código. Solicitar que esses erros sejam corrigidos. Identificar e discutir soluções que poderiam ser aplicadas no código para evitar o problema. Implementar uma solução. Ou, localmente gerar esses erros em exemplos de código do projeto e solicitar que os alunos corrijam esses erros. (continuidade do LO64)

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO73	Escrever um componente de software que execute uma tarefa não trivial e que seja flexível quanto a erros de entrada de dados e execução do programa.	CSLO [SC-e]				Uso	3	Identificar no projeto componentes importantes que sejam vulneráveis a erros de entrada de dados e execução. Propor e implementar soluções para que estes componentes tornem-se mais flexíveis à ocorrência desses erros. (Reescrever componente que não trate erros de entrada e execução de um projeto OSP).
LO74	Descrever as melhores práticas de desenvolvimento de software para minimizar as vulnerabilidades do código.	CSLO [SSE-e]				Familiaridade	3	Identificar as vulnerabilidades existentes no código do projeto. Implementar correções para eliminá-las. Discutir quais práticas no processo de desenvolvimento pode minimizar essas vulnerabilidades.
LO65	Compreender diferentes linguagens de programação.			(GHOSH; GLOTT, 2005)		Familiridade	3	Solicitar que os alunos identifiquem as diversas tecnologias e linguagens empregadas no projeto. Ou, analisar o modelo de arquitetura de projetos OS distintos, implementados em diferentes linguagens de programação (ex: Java, C#, Php).
LO66	Aplicar várias estratégias de testes e depuração (*debug*) de programas simples.	CSLO [SDF-c1]	(RADERMACHER; WALIA; KNUDSON, 2014) (RADERMACHER; WALIA, 2013)	(GHOSH; GLOTT, 2005) (REKHA et al., 2009) (HISLOP; ELLIS; MORELLI, 2009) (SOWE; STAMELOS; DELIGIANNIS, 2006) (AUER; JUNTUNEN; OJALA, 2011) (PAPADOPOULOS; STAMELOS; MEISZNER, 2012) (STAMELOS, 2009)		Uso	3	Construir testes para o projeto. Aplicar as diversas estratégias de testes. Localizar os erros encontrados.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO67	Construir e depurar (*debug*) programas usando bibliotecas padrões existentes para determinada linguagem de programação.	CSLO [SDF-c1]				Uso	3	Identificar partes do código que utilizem as bibliotecas disponibilizadas pela linguagem Construir testes. Localizar os erros encontrados. Ou, criar novas funcionalidades usando as bibliotecas disponibilizadas pela linguagem (ex: Colections em Java)
LO70	Reusar código escrito por outros.			(GHOSH; GLOTT, 2005) (REKHA et al., 2009)		Uso	3	Identificar no projeto partes do código que foram reusados. Implementar uma nova funcionalidade para o projeto, de modo que seja necessário utilizar componentes desenvolvidos por outros.
LO71	Escrever código de modo que este código possa ser reusado.			(GHOSH; GLOTT, 2005) (REKHA et al., 2009)		Uso	3	Com o uso de ferramentas, identificar códigos duplicados e avaliar possibilidade de refatoração para torná-los reusáveis.
LO76	Saber opinar sobre diferentes estilos de programação baseado em experiência prévia.			(CARRINGTON, 2003) (LI; ZHANG; LI, 2009)		Avaliação	3	Visualizar os diferentes estilos de programação aplicados no projeto. Comparar os estilos encontrados. Ou, avaliar estilos de programação em diferentes projetos OS. Comparar os estilos encontrados.
LO77	Descrever como um contrato pode ser usado para especificar o comportamento de um componente do programa.	CSLO [SDF-c1]				Uso	3	Escrever contratos para especificar o comportamento dos diversos componentes do projeto. Escrever contratos para novas funcionalidades propostas para o projeto.
LO78	Descrever técnicas e mecanismos que podem ser utilizados na implementação do projeto, de maneira que propriedades como confiabilidade, eficiência e robustez possam ser alcançadas.	CSLO [SC-c2]				Familiaridade	3	Verificar a confiabilidade, eficiência e robustez do código do projeto. Identificar as técnicas e mecanismos utilizados ou que poderiam ser utilizados no projeto. Ou para o nível de 'uso', realizar codificações em grupo para novas funcionalidades e discutir como que essas propriedades podem ser alcançadas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO79	Construir código robusto usando mecanismos de tratamento de exceções.	CSLO [SC-c2]				Uso	3	Verificar como o tratamento de exceções foi implementado no projeto. Analisar as vantagens e desvantagens da solução. Verificar a abrangência das exceções tratadas. Caso alguma exceção não esteja sendo tratada, incorpora-la no projeto. Ou, avaliar arquiteturas de tratamento de exceção em projetos OS distintos. Refatorar um projeto para incorporar uma arquitetura de exceção robusta, após a análise dos projetos.
LO80	Descrever o que é um código seguro e quais as práticas defensivas de codificação podem ser aplicadas.	CSLO [SC-c2]				Familiaridade	3	Identificar e justificar se o código do projeto é seguro. Identificar as práticas defensivas de codificação utilizadas ou que poderiam ser utilizadas. Ou, identificar em projetos OS exemplos onde as práticas defensivas não foram utilizadas. Ou para o nível de 'uso', refatorar o código para incorporação dessas práticas.
LO81	Identificar os princípios fundamentais do método de desenvolvimento dirigido por testes e explicar o papel da automatização dos testes nesse método.	CSLO [V&V-e]				Familiaridade	2	Exemplificar como um determinado componente do projeto poderia ser desenvolvido a partir do método de desenvolvimento dirigido por testes. Demonstrar a construção do componente a partir da implementação e execução dos testes.
LO82	Construir, executar e depurar (*debug*) programas usando uma IDE moderna, que possua ferramentas de teste de unidade e depuração (*debug*) visual.	CSLO [SDF-c1]	(RADERMACHER; WALIA; KNUDSON, 2014)	(XING, 2010) (GHOSH; GLOTT, 2005)		Uso	3	Usar na manipulação do projeto uma IDE com ferramentas de testes e depuração (*debug*) integradas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO83	Ter noção sobre ambientes que automatizam parte do processo de construção do software (*build automation*); integração contínua.	CSCT [T&E-c2]		(HORSTMANN, 2009)		Familiaridade	3	Utilizar as ferramentas utilizadas pela comunidade. Utilizar localmente o projeto para demonstrar ferramentas de automatização da construção do software. Criar um *script* para realização automática do *build* e integração dos componentes.
LO84	Saber usar e configurar ferramentas necessárias ao ambiente de produção do software na indústria.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	1	Criar um servidor, instalar e configurar todas as ferramentas necessárias o funcionamento do software. Implantar e usar o software.
LO85	Saber usar ferramentas de análise de código.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Aplicar ferramentas de análise de código no projeto. Avaliar resultados em relação a padrões de qualidade discutidos pela comunidade.
LO86	Ter noção de ferramentas de desenvolvimento distribuído.			(ELLIS et al., 2007) (HORSTMANN, 2009)		Familiaridade	3	Utilizar as ferramentas utilizadas pela comunidade do projeto.
LO285	Comparar e contrastar estratégias de integração de código, incluindo *top-down*, *bottom-up* e integração sanduíche.	CSLO [SC-c2]				Familiaridade	2	Demonstrar usando uma funcionalidade do projeto e seus componentes implementados, como seria a sua integração, usando cada uma das estratégias. Localmente fazer com que os alunos façam a integração de componentes fornecidos, utilizando cada uma das estratégias. Criar *scripts* para integração automática de código que implemente as diferentes estratégias.
LO286	Acompanhar o passo a passo da execução de segmentos de código e descrever resumos do processamento ocorrido.	CSLO [SDF-c1]				Uso	3	Solicitar que os alunos acompanhem a execução passo a passo de segmentos de código e descrevam resumos do processamento ocorrido. Discutir os dados em memória que podem ser visualizados a cada execução de linha do código.
Teste de Software								

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO90	Descrever os conceitos e práticas relacionadas ao teste de software.			(CARRINGTON, 2003)		Familiaridade	3	O professor demonstra os conceitos e as práticas de testes usando o OSP.
LO91	Entender a diferença entre testar pequenos programas e testar software de grande porte.			(SOWE; STAMELOS, 2007)		Familiaridade	3	Criar testes para um pequeno programa isolado depois criar testes para todo o software. (relacionado ao LO95, LO101, LO104)
LO92	Discutir as limitações de testar softwares em domínios particulares.	CSLO [V&V-c2]				Familiaridade	3	Selecionar projetos OS de diferentes domínios (*Web* e *Mobile*) para entender as diferenças entre os testes.
LO93	Avaliar o conjunto de testes para um segmento de código de médio porte.	CSLO [V&V-e]				Uso	3	Avaliar o conjunto de testes existentes para um projeto OSP de médio porte.
LO94	Discutir as questões envolvidas em testar um software orientado a objetos.	CSLO [V&V-e]				Uso	3	O prof. demonstra exemplos das questões que envolvem testar software OO. Alunos identificam outros exemplos e elaboram testes para esses casos. Avaliar testes em diferentes projetos OS implementados segundo o paradigma OO e discutir características.
LO95	Descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).	CSLO [V&V-c2]		(SOWE; STAMELOS, 2007)		Uso	3	Analisar exemplos de testes existentes no projeto. Elaborar testes de unidade, integração, sistema e aceitação. (relacionado ao LO101 e LO104)
LO96	Descrever e distinguir entre tipos diferentes de testes, incluindo interface humano computador, usabilidade, confiabilidade, segurança, conformidade com a especificação.	CSCT [V&V-c2]		(SOWE; STAMELOS, 2007) (WILLIAMS; SHIN, 2006)		Uso	3	Analisar exemplos de testes existente no projeto. Elaborar testes de interface, usabilidade, confiabilidade, segurança, aderência às especificações (relacionado ao LO101, LO104)

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO97	Conduzir a verificação e avaliação (estática e dinâmica) da segurança de uma aplicação de software.	CSLO [SSE-e]				Uso	3	Realizar a verificação e avaliação (estática e dinâmica) da segurança do software. Usar ferramentas que possam auxiliar nessas verificações. Implementar correções para os erros apresentados.
LO34	Criar e conduzir testes de usabilidade simples para um software existente.	CSLO [HCIF-c1]				Avaliação	3	Elaborar e realizar testes de usabilidade Criar testes de usabilidade para diferentes projetos OS de diferentes domínios (ex: *Web*, *Mobile* e *Desktop*).
LO98	Descrever e distinguir entre as diferentes técnicas de testes caixa-preta e caixa-branca.	CSCT [V&V-c2]		(WILLIAMS; SHIN, 2006)		Uso	3	Elaborar casos de testes caixa-preta e caixa-branca. (relacionado ao LO99)
LO99	Criar testes de unidade que não sejam redundantes.		(RADERMACHER; WALIA; KNUDSON, 2014)	(MARMORSTEIN, 2011)		Uso	3	(Objetivo das técnicas de testes caixa-preta e caixa-branca é reduzir a redundância, está relacionado ao LO98).
LO100	Realizar o processo de testes e correção de erros sem que haja a reintrodução de erros antigos.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Construir e executar testes. Localizar e corrigir erros. Executar testes de regressão.
LO101	Criar e documentar um conjunto de testes para um segmento de código de médio porte.	CSLO [V&V-c2]		(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)		Uso	3	Elaborar e documentar um conjunto de testes para o projeto (encontrar um pequeno número de erros (5-10) não submetidos anteriormente). (Relacionado ao LO66, LO95, LO96, LO104, LO91, LO99)
LO102	Descrever técnicas para identificar casos de testes significantes para integração, regressão e testes de sistema.	CSLO [V&V-c2]		(STAMELOS, 2009)		Familiaridade	3	Demonstrar as técnicas usando o projeto, como exemplo. (Relacionado ao LO103, LO95, LO100)

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO103	Descrever como selecionar bons testes de regressão e automatizalos.	CSLO [V&V-c2]		(STAMELOS, 2009)		Familiaridade	3	Demonstrar o uso e a importância de testes de regressão usando o projeto, inclusive automatizando estes testes. (O LO102 inclui parte deste objetivo, também está relacionado ao LO100)
LO104	Planejar e gerar casos de testes para softwares de médio porte.	CSCT [V&V-c2]		(SOWE; STAMELOS, 2007)		Uso	3	Muito próximos de LO101, LO66, LO95, LO96, LO99, LO91
LO105	Descrever um processo de gerenciamento de defeitos: como reportar o erro, monitoramento das ações de remoção de erros, submissão de correções.	CSCT[V&V-c2]		(STAMELOS, 2009) (PAPADOPOULOS; STAMELOS; MEISZNER, 2012) (LI; ZHANG; LI, 2009) (MARMORSTEIN, 2011)		Familiaridade	3	Identifcar no *bug tracker* do projeto como ocorre o gerenciamento de defeitos do projeto. Apresentar um exemplo (monitorar um exemplo de ações de remoção de erros). Participar do processo reportando erros e auxiliando desenvolvedores na sua correção. Participar do processo propondo soluções para os erros já reportados. Submeter correções.
LO87	Avaliar o programa sendo testado por meio de métricas.				P	Uso	3	Avaliar as funcionalidades testadas do OSP usando métricas. Usar ferramentas para coleta de métricas. Analisar os resultados em grupo.
LO88	Avaliar os testes sendo executados por meio de métricas.				P	Uso	3	Avaliar os testes executados no projeto usando métricas de cobertura Usar ferramentas para análise de cobertura.
LO106	Descrever como ferramentas de testes estáticos e dinâmicos disponíveis podem ser integrados no ambiente de desenvolvimento do software.	CSLO [T&E-c2]				Familiaridade	3	Identificar quais e como as ferramentas de testes (dinâmicas ou estáticas – análise estática) são usadas no projeto. Usar uma cópia local do projeto e demonstrar como as ferramentas de testes estáticas e dinâmicas podem ser utilizadas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO107	Demonstrar a capacidade de usar ferramentas de testes incluindo ferramentas de análise estáticas e dinâmicas, em suporte ao desenvolvimento de um software de médio porte.	CSLO [T&E-c2]	(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Elaborar e executar testes usando as ferramentas apropriadas. (Em conjunto com LO95, LO96, LO104, LO101, LO106, LO66)
LO108	Demonstrar a capacidade de usar ferramentas de depuração (*debug*), em suporte ao desenvolvimento de um software de médio porte.		(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Detectar os erros no código usando ferramentas apropriadas Relacionado com LO66 (*debug*), LO105 (submissão de correções) LO82 (ferramentas de *debug* integradas à IDE)
LO109	Demonstrar a capacidade de usar ferramentas de teste de cobertura eficientemente.		(RADERMACHER; WALIA, 2013)			Uso	3	Usar ferramentas de cobertura para avaliar os testes existentes no projeto. Usar ferramentas de cobertura para avaliar os testes elaborados pelo próprio aluno. (associado ao LO88) Discutir resultados.
LO110	Ter noção sobre ferramentas de integração continua e testes de regressão.		(RADERMACHER; WALIA; KNUDSON, 2014)			Familiaridade	3	Verificar se ferramentas de integração contínua e testes de regressão são empregados no projeto ou como poderiam ser utilizadas. Fazer uma cópia local do projeto e demonstrar o uso das ferramentas de integração contínua e testes de regressão. Criar ou atualizar *scripts* para realização de integração contínua e testes de regressão para novos componentes desenvolvidos.
Manutenção de software								
LO111	Reconhecer os tipos de manutenção de software (categorias: corretivas, adaptativas, aperfeiçoamento/perfectivas, preventiva).				P	Familiaridade	3	Identificar as solicitações de mudanças no *issue tracker* do projeto. Classificar essas solicitações de acordo com a manutenção que irão provocar. Identificar no projeto as necessidades de manutenção preventiva.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO112	Discutir os desafios de evoluir sistemas em ambientes de mudanças constantes.	CSLO [SE-c2]		(ELLIS et al., 2007)		Familiaridade	1	Realizar algumas atividades de manutenção no projeto. Analisar se o domínio do projeto corresponde a um ambiente de mudanças constantes. Comparar com outros projetos. Discutir os desafios de evoluir o projeto.
LO113	Identificar as questões principais associadas com a evolução de software e explicar seu impacto no ciclo de vida do software.	CSLO [SE-c2]		(HISLOP; ELLIS; MORELLI, 2009)		Familiaridade	1	Realizar algumas atividades de manutenção no projeto. Discutir as consequências da evolução para o projeto. Verificar o decaimento do código do projeto, analisando algumas versões anteriores do projeto.
LO114	Perceber a importância da documentação para a manutenção de software.			(MCCARTNEY; GOKHALE; SMITH, 2012) (MARMORSTEIN, 2011)		Familiaridade	3	Verificar se o projeto está documentado, se a documentação está atualizada. Fazer a manutenção para uma parte não documentada do projeto. Fazer a manutenção para uma parte documentada do projeto. Ou, fazer manutenção em um OSP com boa documentação e em outro OSP com pouca documentação e discutir as dificuldades.
LO115	Entender as nuances da programação e os desafios envolvidos em escrever código limpo.			(REKHA et al., 2009)		Familiaridade	3	Identificar se o código do projeto é claro e coeso. Identificar *code smells* no código. Reescrever (refatorar) código de maneira que o mesmo fique claro e coeso. Ou, comparar códigos em diferentes projetos OS verificando a adoção de boas práticas de programação.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO116	Identificar o impacto das decisões dos desenvolvedores feitas no início do projeto para o ciclo de vida do projeto.			(ELLIS; MORELLI; HISLOP, 2008a) (HISLOP; ELLIS; MORELLI, 2009) (MCCARTNEY; GOKHALE; SMITH, 2012)		Familiaridade	1	Realizar algumas atividades de manutenção no projeto. Analisar se a arquitetura do projeto facilita a realização de manutenções.
LO117	Compreender a importância da manutenção e evolução em um projeto de software, especificamente considerando o tamanho do esforço necessário nessas atividades.			(MCCARTNEY; GOKHALE; SMITH, 2012)		Familiaridade	3	Avaliar o ciclo de vida do projeto, número de *releases* liberadas, quantidade de modificações realizadas ao longo de cada *release*, o tempo entre *releases*. Procurar identificar a *release* que o software ficou mais estável. Verificar a quantidade de *realeases*, a quantidade de modificações após a estabilidade do software para cada *release* e o período de tempo entre as *releases*. Comparar os resultados de outros projetos OS, obtidos por outros alunos.
LO118	Realizar a manutenção de um software real de médio porte.		(RADERMACHER; WALIA, 2013)	(GHOSH; GLOTT, 2005) (CARRINGTON, 2003) (PETRENKO et al., 2007)		Uso	3	Realizar atividades de manutenção no projeto.
LO119	Descrever as características apresentadas por um software fácil de manter (Descrever as características que facilitam a manutenção do software).	CSCT [SE-c2]				Uso	3	Analisar se o projeto apresenta características que facilitam a sua manutenção. Solicitar modificações que exijam que os alunos interajam entre si, uma vez que as modificações sobrepõem-se. Ou, solicitar análise de pelo menos dois projetos OS. O primeiro com facilidade e o segundo com dificuldade de manutenção. Fazer comparativo e analisar as características que ajudaram ou dificultaram a manutenção.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO120	Estimar o impacto da mudança solicitada para um produto de médio porte existente.	CSLO [SE-c2]		(STAMELOS, 2009) (XING, 2010) (PETRENKO et al., 2007)		Uso	3	Analisar o impacto da mudança para o projeto. Usar métodos que possam estimar o impacto da mudança.
LO68	Descrever o processo de analisar e implementar mudanças no código de determinado projeto.	CSLO [SC- c2]		(MCCARTNEY; GOKHALE; SMITH, 2012) (BUCHTA et al., 2006) (PETRENKO et al., 2007)		Familiaridade	3	Solicitar uma manutenção para o projeto. Solicitar que os alunos relatem como executariam tal mudança. Discutir com os alunos o processo para executar essa mudança. Solicitar que os alunos executem a mudança. Modelar o processo para analisar e implementar mudanças.
LO69	Descrever o processo de analisar e implementar mudanças no código de um projeto grande.	CSLO [SC- c2]		(MCCARTNEY; GOKHALE; SMITH, 2012) (BUCHTA et al., 2006) (PETRENKO et al., 2007)		Familiaridade	3	Solicitar manutenções para o projeto, que envolvam a modificação de várias partes do projeto, de maneira que os alunos precisem interagir. Solicitar que os alunos relatem como executariam tais mudanças. Discutir com os alunos o processo para executar essas mudanças. Solicitar que os alunos executem as mudanças. Discutir como foi a integração das manutenções realizadas. Modelar o processo para analisar e implementar mudanças.
LO122	Descrever uma forma de refatoração e discutir quando ela seria aplicável.	CSLO [SD-c2]		(STAMELOS, 2009) (CARRINGTON, 2003) (BUCHTA et al., 2006)		Familiaridade	3	Identificar *code smells* no projeto. Usar ferramentas para identificar *code smells*. Analisar se refatorações seriam realmente necessárias. No caso afirmativo, discutir quais refatorações poderiam ser aplicadas. Aplicar as refatorações selecionadas.
LO123	Refatorar uma implementação de software existente a fim de melhorar algum aspecto do seu design.	CSLO [SD-e]		(STAMELOS, 2009) (CARRINGTON, 2003) (BUCHTA et al., 2006)		Uso	3	Realizar a refatoração de partes do projeto. Discutir o resultado.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO318	Refatorar um programa identificando oportunidades de aplicar abstração procedural	CSLO [SDF-c1]		(STAMELOS, 2009) (CARRINGTON, 2003) (BUCHTA et al., 2006)		Uso	3	Refatorar o código aplicando abstração procedural. Discutir o resultado.
LO319	Usar refatoração no processo de modificação de um componente de software.	CSLO [SE-c2]		(STAMELOS, 2009) (CARRINGTON, 2003) (BUCHTA et al., 2006)		Uso	3	Apresentar um exemplo de modificação no código que exija a refatoração prévia do mesmo. Solicitar uma modificação em um código que exija a refatoração prévia do mesmo. Realizar a refatoração. Realizar a modificação. Discutir o resultado.
LO124	Aplicar engenharia reversa em um software de médio porte.			(NANDIGAM; GUDIVADA; HAMOULHADJ, 2008) (KAMTHAN, 2007) (MCCARTNEY; GOKHALE; SMITH, 2012)		Uso	3	Recuperar o design do projeto. Discutir a importância da engenharia reversa para a compreensão do código com fins de adicionar novas funcionalidades ou melhorar o próprio código.
LO125	Aplicar reengenharia para um software existente de médio porte.	CSCT [SE-c2]		(KAMTHAN, 2007)		Uso	3	Reprojetar parte do projeto.
LO126	Demonstrar a capacidade de usar ferramentas que deem suporte as tarefas do processo de manutenção, como *issue tracker*, gerenciamento de versões.			(MCCARTNEY; GOKHALE; SMITH, 2012) (MARMORSTEIN, 2011) (PETRENKO et al., 2007)		Uso	3	Interagir com a comunidade a partir do *issue tracker* do projeto. Ao realizar as modificações solicitadas, usar ferramentas que propiciem o gerenciamento de configuração, a detecção de *code smells*, a refatoração, testes de regressão, automação de *builds*.
LO287	Descrever o processo de testes de regressão e seu papel no gerenciamento de releases.	CSLO [SE-c2]				Familiaridade	3	Aplicar testes de regressão em um projeto que já possua os testes implementados e automatizados. Discutir o histórico dos resultados da execução dos testes.
Gerência de Configuração do Software								

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO127	Demonstrar os conceitos e práticas do gerenciamento de configuração.			(CARRINGTON, 2003) (REKHA et al., 2009)		Uso	3	Criar uma cópia local do projeto. Demonstrar os conceitos e as práticas do gerenciamento de configuração usando a cópia local. Elaborar exercícios que os alunos exercitem as práticas do gerenciamento de configuração (*check in, check out, branch, merge*). Aplicar as técnicas implementando mudanças em um projeto OSP.
LO128	Descrever como o controle de versões pode ser usado para auxiliar no gerenciamento de releases do software.	CSLO [T&E-c2]		(ROBLES; CABALLÉ; GONZÁLEZ-BARAHONA, 2008)		Familiaridade	3	Verificar no sistema de controle de versões utilizado no projeto, como ocorre a liberação das *releases* do software. Identificar as atividades necessárias e o que fica registrado.
LO129	Descever as diferenças entre softwares de gerenciamento de configuração centralizados e distribuídos.	CSLO [T&E-c2]				Familiaridade	3	Realizar mudanças em um projeto com controle centralizado e em outro com controle distribuído. Identificar a abordagem utilizada em cada projeto. Discutir as diferenças entre as duas abordagens.
LO130	Identificar itens de configuração e usar o controle de código fonte em um projeto pequeno desenvolvido em equipe.	CSLO [T&E-c2]				Uso	3	Identificar os itens de configuração utilizados no projeto. Criar uma cópia local do projeto. Elaborar exercícios que os alunos exercitem as práticas do gerenciamento de configuração (*check in, check out, branch, merge*).
LO131	Ter noção sobre políticas de gerenciamento de configuração			(STAMELOS, 2009)		Familiaridade	3	Identificar qual a política de gerenciamento de configuração utilizada no projeto. Buscar documento que descreva a política de gerenciamento de configuração. Avaliar se o que está escrito está sendo executado pelos desenvolvedores.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO58	Saber quais são as informações necessárias para gerenciar eficientemente a configuração.				P	Familiaridade	3	Identificar as informações utilizadas no gerenciamento de configuração do projeto. Avaliar se alguma outra informação seria necessária.
LO59	Realizar a auditoria física em um software.				P	Uso	3	Realizar a auditoria física no projeto. Usar ferramentas de auditoria de repositórios.
LO132	Emular o lançamento de *releases*.			(STAMELOS, 2009)		Uso	3	Criar uma cópia local do projeto e emular o lançamento de uma *release*.
LO133	Demonstrar a capacidade de usar as ferramentas de gerenciamento de configuração, controle de versões e gerenciamento de *releases* em suporte ao desenvolvimento de um software de médio porte.	CSLO [T&E-c2]	(RADERMACHER; WALIA, 2013)	(HISLOP; ELLIS; MORELLI, 2009) (BUCHTA et al., 2006) (PETRENKO et al., 2007) (HORSTMANN, 2009)		Uso	3	Com uma cópia local do projeto usar as ferramentas de gerenciamento de configuração, controle de versões e gerenciamento de *releases*.
LO134	Ser capaz de fazer *merge* e ramificação de código.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Criar uma cópia local do projeto Elaborar exercícios que os alunos exercitem o *branch* e o *merge*.
Gerenciamento de Engenharia de Software								
LO135	Ter a capacidade de definir o escopo e os objetivos de um projeto.				P	Uso	3	Descrever os objetivos e escopo do projeto. Definir melhorias significativas para o projeto, seja em relação aos requisitos ou atualização tecnológica.
LO136	Analisar a viabilidade técnica para um projeto.				P	Avaliação	2	Definir melhorias significativas para o projeto, seja em relação aos requisitos ou atualização tecnológica. Avaliar a viabilidade dessas melhorias. Propor alternativas técnicas, apontando as vantagens e desvantagens de cada uma. Ou, encontrar alguma solicitação de mudança que necessite uma alteração técnica considerável. Discutir soluções e analisar a viabilidade de cada uma delas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO137	Analisar a viabilidade de negócio para um projeto.			(STAMELOS, 2009)		Avaliação	1	Avaliar a capacidade do projeto em atrair novos desenvolvedores e usuários. Avaliar estratégias de captar financiamentos para o projeto.
LO329	Planejar o desenvolvimento de um projeto.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Elaborar o planejamento de desenvolvimento de uma nova funcionalidade. Planejar a execução de uma melhoria para o projeto.
LO138	Usar um método *ad hoc* para a estimativa de esforço de desenvolvimento de software e comparar com o esforço real exigido.	CSLO [SPM-c2]	(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	2	Definir uma nova funcionalidade para o projeto. Aplicar um método *ad hoc* para a estimativa de esforço necessário a implementação da nova funcionalidade. Implementar a nova funcionalidade. Comparar com o esforço exigido.
LO139	Discutir comportamentos comuns que contribuem para o efetivo funcionamento da equipe.	CSLO [SPM-c2]				Familiaridade	1	Identificar nas ferramentas de comunicação empregadas no projeto, exemplos de comportamentos que contribuem e que não contribuem para o efetivo funcionamento da equipe. Realizar as diversas atividades sobre o projeto, em grupo, solicitar que cada aluno avalie o seu próprio comportamento e o comportamento dos colegas anonimamente. Discutir os comportamentos relatados sem que nenhum aluno seja identificado. Buscar artigos científicos na literatura que discutam esses comportamentos (muitos artigos tratam do comportamento dos desenvolvedores em projetos OS).

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO140	Criar e seguir uma agenda de reuniões da equipe.	CSLO [SPM-c2]				Uso	1	Realizar as diversas atividades sobre o projeto, em grupo. Estabelecer uma agenda de reuniões para os grupos. Acompanhar os conteúdos discutidos durante as reuniões. Criar cronograma de projeto que possua essas reuniões agendadas.
LO141	Identificar e justificar os papeis necessários em uma equipe de desenvolvimento de software.	CSLO [SPM-c2]				Uso	1	Identificar no *issue tracker* do projeto, os papéis existentes no projeto. Discutir a responsabilidade de cada papel. Comparar com os papéis existentes no desenvolvimento de software em empresas.
LO142	Criar uma equipe, identificando os papéis apropriados e atribuindo-os aos membros da equipe.	CSLO [SPM-e]		(PAPADOPOULOS; STAMELOS; MEISZNER, 2012)		Uso	1	Fazer com que os alunos assumam os papéis de desenvolvedores, testadores... junto à comunidade. Localmente, podem ser definidos papéis para os alunos no desenvolvimento das atividades sobre o projeto.
LO143	Demonstrar através do envolvimento em um trabalho em equipe, os elementos centrais para a definição e gerenciamento da equipe.	CSLO [SPM-e]				Uso	1	Localmente definir a equipe para o desenvolvimento das atividades no projeto. Descrever características que devem ser consideradas ao montar uma equipe. Descrever como a equipe pode ser gerenciada.
LO144	Usando um processo particular de software, descrever os aspectos do projeto que precisam ser planejados e monitorados (por exemplo, estimativas de tamanho e esforço, cronograma, alocação de recursos, controle de configuração, gerência de mudança, identificação e gerenciamento de riscos).	CSLO [SPM-e]				Familiaridade	1	Usando o processo utilizado no desenvolvimento de OSP, descrever os aspectos do projeto que precisam ser planejados e monitorados. Localmente, propor como projeto, uma nova funcionalidade para o software. Selecionar um processo particular de software e descrever os aspectos do projeto que precisam ser planejados e monitorados, aplicar esse processo para desenvolver essa nova funcionalidade.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO145	Descrever as questões que são importantes na seleção de um conjunto de ferramentas para o desenvolvimento de um software particular, incluindo ferramentas para o rastreamento de requisitos, modelagem do design, implementação, automação de construção (*build*) e teste.	CSLO [T&E-c2]				Familiaridade	1	Identificar as ferramentas usadas no projeto. Identificar outros tipos de ferramentas não explicitados no projeto que poderiam ser utilizados. Descrever o objetivo de cada ferramenta. Descrever as características que cada ferramenta deve apresentar para atender a sua função. Fazer o comparativo das ferramentas usadas em diferentes projetos.
LO146	Compreender as fontes, os perigos e os potênciais benefícios de conflitos na equipe.	CSLO [SPM-c2]				Uso	1	Discutir as possíveis fontes, perigos e benefícios da existência de conflitos para o projeto. Identificar conflitos porventura existentes na comunidade do projeto (ferramentas de comunicação, existência e razões para *branches, forks*).
LO147	Aplicar uma estratégia de resolução de conflitos em uma equipe.	CSLO [SPM-c2]		(PETRENKO et al., 2007)		Uso	1	Caso tenham sido identificados conflitos no projeto, verificar como estes foram solucionados, quais foram as consequências. Localmente, propor como projeto, modificações em pontos distintos do software, mas que estejam interligados, de maneira que as equipes precisem interagir. Aplicar estratégias de resolução de conflitos localmente.
LO148	Cumprir o cronograma estabelecido.				P	Uso	1	Contribuir para a comunidade, atendendo ao planejamento de lançamento de novas *releases*. Cumprir o cronograma estabelecido para subprojetos locais.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO149	Ter a noção de como ocorre o monitoramento de um projeto.			(ROBLES; CABALLÉ; GONZÁLEZ-BARAHONA, 2008) (STAMELOS, 2009)		Familiaridade	1	Avaliar como as ferramentas de gerenciamento são usadas para monitoramento em diferentes projetos OS. Verificar como as atividades locais do projeto são monitoradas.
LO150	Ter a vivência em um projeto que tem o ciclo de vida maior que as disciplinas na universidade.			(STROULIA et al., 2011) (WHITEHURST, 2009)		Familiaridade	1	Participar ativamente com contribuições para o projeto.
LO151	Avaliar e prover respostas para a equipe e indivíduos sobre seu desempenho na equipe.	CSLO [SPM-e]				Uso	2	Receber as respostas da comunidade sobre suas contribuições. Realizar as diversas atividades sobre o projeto, em grupo, solicitar que cada aluno avalie o desempenho dos colegas anonimamente. Os alunos devem receber as avaliações anônimas de seus colegas, sobre o seu desempenho. Discutir os desempenhos relatados sem que nenhum aluno seja identificado.
LO152	Acompanhar o progresso de algum estágio de um projeto usando métricas apropriadas.	CSLO [SPM-e]				Familiaridade	1	Aplicar métricas apropriadas para acompanhar o progresso de subprojetos locais. Usar e configurar ferramentas para coleta de métricas.
LO153	Ter a vivência de um processo de revisão de código.			(ELLIS et al., 2007)		Familiaridade	3	Receber a resposta da comunidade sobre a contribuição de código submetida. Aplicar o processo de revisão de código nos subprojetos locais. Usar ferramentas automáticas de revisão de código.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO154	Comparar técnicas de estimativas de tamanho e custo de software.	CSLO [SPM-e]				Uso	1	Aplicar as técnicas para estimativa de tamanho do projeto a partir dos requisitos. Comparar com o tamanho real do projeto. Comparar os resultados obtidos com as técnicas de estimativa de tamanho. Aplicar as técnicas para estimar os subprojetos locais. Realizar os subprojetos locais. Comparar as estimativas com o tamanho e custo real obtido para o subprojeto. Comparar os resultados de cada técnica.
LO155	Usar uma ferramenta de gerenciamento de projetos para auxiliar a atribuição e o acompanhamento de tarefas no projeto de desenvolvimento de software.	CSLO [SPM-e]	(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Os alunos devem identificar como ocorre a atribuição e o acompanhamento de tarefas no projeto.
LO328	Usar ferramenta de rastreamento de necessidades (*issue tracker*).		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Os alunos podem selecionar e assumir as contribuições que irão fazer no *issue tracker* do projeto.
LO288	Descrever como a escolha do modelo de processo afeta a estrutura organizacional da equipe e o processo de tomada de decisão.	CSLO [SPM-e]				Familiaridade	1	Comparar o modelo do processo dos projetos de código aberto com os demais modelos, quanto à estrutura da equipe e tomada de decisão.
LO289	Examinar medidas apropriadas usadas para comunicar com os stakeholders do projeto.	CSLO [PC-c1]				Uso	0	
Processos de Engenharia de Software								
LO156	Descrever como desenvolver um software é diferente de um esforço individual de um programa, no que diz respeito a compreender uma base de código grande, o processo de *build*, e o contexto de mudanças constantes.	CSLO [SP-c1]				Familiaridade	3	Descrever a arquitetura de um OSP de tamanho razoável. Solicitar uma modificação em um OSP de tamanho razoável.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO157	Descrever os passos necessários para planejar e dar andamento em um projeto de software em um ambiente real.			(ELLIS et al., 2007) (HISLOP; ELLIS; MORELLI, 2009)		Familiaridade	3	Descrever o processo de desenvolvimento de projetos de código aberto. Comparar com processos usados para construir softwares proprietários. Comparar como seriam os passos em cada um desses processos.
LO158	Explicar o conceito de ciclo de vida de software e prover um exemplo, ilustrando suas fases, incluindo as entregas que são produzidas.	CSLO [SP-c2]		(LI; ZHANG; LI, 2009) (MARMORSTEIN, 2011)		Familiaridade	3	Visualizar todo o ciclo de vida do software, a evolução do projeto, reconhecer as fases porventura existentes, as diversas *releases* lançadas e analisar se o código já entrou em decaimento. Comparar o ciclo de vida de OSP e softwares proprietários.
LO159	Visualizar o processo e as técnicas de desenvolvimento para ambientes distribuídos.			(ELLIS; MORELLI; HISLOP, 2008a)		Familiaridade	3	O processo do desenvolvimento do projeto em si é distribuído, com vários desenvolvedores ao redor do mundo podendo participar. O aluno deverá descrever como é o processo e as ferramentas utilizadas.
LO160	Aplicar práticas ágeis de desenvolvimento de software.			(ALLEN; CARTWRIGHT; REIS, 2003)		Uso	3	Detectar as práticas ágeis empregadas no projeto. Efetuar contribuições para o projeto que exercitem estas práticas. Aplicar práticas ágeis nos subprojetos locais.
LO161	Vivenciar o processo de desenvolvimento de software baseado em componentes.			(QIAN; FU, 2008)		Familiaridade	3	Construir um projeto a partir do reuso de componentes de código aberto disponibilizados. Substituir um componente por outro em um projeto OSP.
LO162	Descrever as vantagens e desvantagens entre os principais modelos de processos (cascata, iterativo, ágil).	CSLO [SP-c1]		(LUNDELL; PERSSON; LINGS, 2007)		Familiaridade	2	Descrever o processo de desenvolvimento de projetos de código aberto. Comparar com processos usados para construir softwares proprietários. Discutir as vantagens e desvantagens de cada um desses processos.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO163	Comparar os modelos de processos comuns no que diz respeito à sua indicação para o desenvolvimento de classes particulares de software, considerando questões como estabilidade de requisitos, tamanho e atributos não funcionais.	CSLO [SP-c2]				Uso	1	Descrever o processo de desenvolvimento de projetos de código aberto. Verificar os tipos de software que são usualmente construídos usando essa abordagem. Comparar com processos usados para construir softwares que não são comuns para abordagem OSP. Discutir as vantagens e desvantagens de cada um desses processos.
LO164	Descrever os requisitos para integrar segurança ao ciclo de vida de desenvolvimento do software.	CSLO [SSE-e]				Familiaridade	1	Detectar quais mecanismos de segurança são incorporados ao ciclo de vida de desenvolvimento do projeto. Sugerir o que poderia ser melhorado.
LO165	Avaliar o esforço de desenvolvimento e recomendar mudanças potenciais pela participação em um processo de melhoria (usando um modelo como o PSP) ou engajando em uma retrospectiva do projeto.	CSLO [SP-e]				Uso	1	A contribuição é avaliada pela comunidade. A partir da resposta da comunidade o aluno poderá refletir em como melhorar seus resultados. Subprojetos locais também podem e devem passar por um processo de revisão.
LO166	Descrever várias métricas de processos para controle e avaliação de um projeto.	CSLO [SP-e]				Familiaridade	1	Reconhecer quais métricas de processo podem ser aplicadas ao processo OSP. Usar ferramentas que coletem métricas de processo em um projeto OSP em evolução.
LO167	Usar métricas de projeto para descrever a situação atual do projeto.	CSLO [SP-e]				Uso	1	Detectar a situação atual do projeto usando métricas de processo. Usar ferramentas para coletar as métricas.
LO168	Demonstrar a capacidade de usar ferramentas de modelagem de processo.			(STAMELOS, 2009)		Uso	3	Modelar o processo adotado no projeto (processo OSP).

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO290	Descrever as diferentes práticas que são principais para os vários modelos de processos.	CSLO [SP-c1]				Familiaridade	2	Descrever o processo de desenvolvimento de projetos de código aberto. Comparar com processos usados para construir softwares proprietários. Comparar as principais práticas em cada um desses processos.
LO291	Diferenciar entre as diferentes fases do desenvolvimento de software.	CSLO [SP-c1]				Familiaridade	1	Reconhecer as fases porventura existentes no modelo de processo usado em OSP. Comparar com as fases de outros modelos.
LO292	Descrever os objetivos e as similaridades fundamentais entre abordagens de melhorias do processo.	CSLO [SP-e]	(RADERMACHER; WALIA, 2013)			Familiaridade	1	A vivência de um projeto de software é importante para entender o processo de desenvolvimento de software, por conseguinte é importante para entender o que é necessário para melhora-lo, objetivo dos modelos de melhorias de processo. Descrever o processo de desenvolvimento do projeto. Comparar com outros processos. Descrever melhorias que podem ser realizadas.
LO293	Comparar os diversos modelos de melhorias do processo, como CMM, CMMI, CQI, *Plan-Do-Check-Act*, or ISO 9000.	CSLO [SP-e]				Avaliação	0	
LO294	Explicar o papel dos modelos de maturidade de processo no processo de melhoria.	CSLO [SP-e]				Familiaridade	0	
LO295	Explicar como o projeto centrado no usuário complementa outros modelos de processos de software.	CSLO [UCDT-e]				Familiaridade	0	
Modelos e Métodos de Engenharia de Software								
LO169	Demonstrar os conceitos e práticas da notação UML.			(CARRINGTON, 2003)		Uso	3	Entender os modelos existentes. Atualizar modelos porventura desatualizados. Elaborar modelos inexistentes.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO171	Interpretar os modelos de especificação de requisitos de um software simples.	CSLO [RE-c1]				Familiaridade	3	Interpretar os modelos de especificação de requisitos disponibilizados pela comunidade. Comparar modelos de especificação em projetos OS distintos. Ou para o nível de 'uso', atualizar a especificação em pelo menos um dos projetos.
LO172	Interpretar os modelos de design de um software simples.				P	Familiaridade	3	Interpretar os modelos de design disponibilizados pela comunidade. Comparar modelos de design em diferentes projetos OS. Ou para o nível de 'uso', atualizar modelos em pelo menos um dos projetos.
LO296	Escolher os métodos apropriados para o desenvolvimento de uma interface do usuário específica.	CSLO [UCDT-e]				Avaliação	0	
LO321	Descrever o papel das técnicas de especificação e análise formais podem assumir no desenvolvimento de software complexo e comparar o seu uso com técnicas de validação e verificação por meio de testes.	CSLO [FM-E]				Familiaridade	3	Desenvolver a especificação formal para um módulo complexo do OSP e apresenta-la aos alunos. Descrever o objetivo e a importância dessa especificação. Elaborar testes para o módulo, caso não existam. Comparar as duas soluções.
LO322	Aplicar técnicas de especificação e análise formais no projeto e desenvolvimento de um software de baixa complexidade.	CSLO [FM-E]				Uso	3	Os alunos deverão desenvolver a especificação formal de um OSP simples.
LO323	Explicar as vantagens e desvantagens do uso de linguagens de especificação formal.	CSLO [FM-E]				Familiaridade	3	Visualizando um exemplo de especificação formal elaborado para um OSP, deve-se comparar as duas especificações, identificando vantagens e desvantagens.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO324	Criar e avaliar asserções de programas para uma variedade de comportamentos, desde os mais simples, até os mais complexos.	CSLO [FM-E]				Uso	3	Solicitar que sejam criadas asserções para determinada funcionalidade do OSP.
LO325	Usando uma linguagem de especificação formal, formular a especificação de software simples e derivar exemplos de casos de testes dessa especificação.	CSLO [FM-E]				Uso	3	Os alunos deverão desenvolver a especificação formal de um OSP simples e elaborar casos de testes derivados dessa especificação.
Qualidade de Software								
LO173	Explicar porque a criação dos componentes corretos dos programas é importante para a produção de software de alta qualidade.	CSLO [SDF-c1]				Familiaridade	3	Validar a corretude das funcionalidades do projeto. Usar ferramentas para identificação de componentes com problemas. Analisar os problemas causados se as respostas produzidas pelos componentes do software não são corretas. Analisar o impacto de erros em componentes nas primeiras versões do software. Discutir a relação entre corretude e qualidade do software.
LO174	Definir qualidade de software e descrever o papel das atividades de garantia da qualidade no processo do software.	CSLO [SP-e]		(STAMELOS, 2009)		Familiaridade	3	Analisar a qualidade do projeto. Identificar as atividades de garantia de qualidade empregadas no processo de desenvolvimento / evolução do software. Analisar a efetividade de procedimentos de avaliação da qualidade existentes. Discutir o que poderia ser melhorado.
LO175	Explicar os problemas que existem em alcançar altos níveis de confiabilidade.	CSLO [SR-c2]				Familiaridade	3	Verificar a confiabilidade do projeto. Discutir considerando o projeto, o que seria necessário, quais seriam os desafios para atingir altos níveis de confiabilidade.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO177	Reconhecer atributos de qualidade de software.				P	Uso	3	Analisar a qualidade do projeto, considerando os atributos de qualidade definidos pelas normas de qualidade. Avaliar como cada atributo de qualidade é refletido em projetos OS diferentes.
LO178	Combinar técnicas apropriadas de testes para o desenvolvimento de um software confiável e seguro.			(WILLIAMS; SHIN, 2006)		Uso	3	Analisar se os testes existentes no projeto garantem a confiabilidade e segurança do mesmo. Criar novos casos de testes que cubram as falhas porventura existentes.
LO179	Descrever os conceitos relacionados à usabilidade do software.			(HEPTING et al., 2008)		Uso	3	Identificar como conceitos de usabilidade estão implementados no projeto. Avaliar a usabilidade do software segundo os conceitos defindos. Avaliar conceitos de usabilidade em diferentes projetos OSP.
LO54	Usar uma variedade de técnicas para avaliar uma interface de usuário fornecida.	CSLO [UCDT-e]				Avaliação	3	Aplicar as técnicas definidas para avaliar a interface de usuário do software. Usar ferramentas de acordo com a interface avaliada.
LO298	Conduzir uma avaliação quantitativa da interface do usuário e discutir/reportar os resultados.	CSLO [DI-c2]				Uso	3	Avaliar a interface do usuário. Usar ferramentas para a realização da avaliação e coleta de informações.
LO55	Comparar as restrições e benefícios de métodos de avaliação de interface do usuário diferentes	CSLO [UCDT-e]				Avaliação	3	Aplicar cada método de avaliação de interface do usuário no projeto e comparar o processo de cada método e os resultados obtidos.
LO180	Conduzir uma revisão de código pessoal (focada em erros de código comuns) em um componente do programa, usando um checklist fornecido.	CSLO [SDF-c1]				Uso	3	Elaborar em equipe um checklist para inspeção de código. Revisar um código existente do projeto. Comparar o resultado com os demais alunos. Revisar o próprio código desenvolvido para o projeto. Compreender a revisão do grupo de desenvolvedores core da comunidade para alguma contribuição de código disponibilizada.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO181	Contribuir para a revisão de código efetuada por uma equipe pequena, focada principalmente na corretude do componente.	CSLO [SDF-c1]				Uso	3	Usar checklist de corretude para um mesmo código do projeto, a ser aplicado por vários alunos. Comparar resultados.
LO182	Participar da inspeção de um segmento de código de médio porte.	CSLO [V&V-c2]				Uso	3	Fazer inspeção com vários alunos de vários componentes do projeto. Comparar resultados. Enviar o relatório final para a comunidade.
LO183	Conduzir uma inspeção ou revisão do código fonte para um projeto de pequeno e médio porte.	CSLO [V&V-e]				Uso	3	Fazer inspeção com vários alunos de vários componentes do projeto. Comparar resultados. Enviar o relatório final para a comunidade.
LO184	Distinguir entre validação e verificação de programas.	CSLO [V&V-c2]				Familiaridade	3	Usar o código do projeto para elaborar exemplos de validação e verificação de programas. Solicitar que os alunos sugiram operações de validação e verificação para determinado código do projeto.
LO185	Comparar abordagens estáticas e dinâmicas para verificação.	CSLO [V&V-e]				Familiaridade	3	Usar o código do projeto para elaborar exemplos de verificação estáticas e dinâmicas de programas. Solicitar que os alunos elaborem operações de verificação estáticas e dinâmicas para determinado código do projeto. Discutir os objetivos, vantagens e desvantagens de cada abordagem. Discutir ferramentas que poderiam ser usadas para auxiliar na verificação estática e dinâmica. Usar essas ferramentas no projeto.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO186	Descrever técnicas para verificação e validação dos demais artefatos gerados pelo projeto além do código.	CSLO [V&V-e]				Familiaridade	3	Usar os artefatos do projeto para elaborar exemplos de técnicas para verificação e validação que podem ser aplicadas. Solicitar que os alunos sugiram como verificariam e validariam determinado artefato do projeto.
LO187	Listar as abordagens que minimizam falhas, que podem ser aplicadas em cada estágio do ciclo de vida do software.	CSLO [SR-c2]				Familiaridade	3	Identificar quais abordagens são empregadas para minimizar as falhas no decorrer do projeto. Comparar com outros processos de desenvolvimento. Identificar quais outras abordagens poderiam ser aplicadas. Discutir como essas abordagens poderiam ser inseridas no projeto.
LO188	Avaliar a documentação técnica escrita para determinar problemas de vários tipos (comunicação).	CSLO [PC-c1]				Avaliação	3	Avaliar a documentação escrita existente quanto à clareza e utilidade. Comparar a documentação técnica existente em vários projetos OS.
LO189	Descrever abordagens para a estimativas de falhas.	CSLO [V&V-e]				Familiaridade	3	Apresentar exemplos para as abordagens de estimativas de falhas usando o código do projeto. Solicitar que os alunos sugiram a abordagem a ser empregada para determinado código do projeto.
LO190	Estimar o número de falhas de um software de pequeno porte baseados na densidade e disseminação de falhas.	CSLO [V&V-e]				Uso	3	Estimar o número de falhas para o projeto, baseando-se em técnicas de densidade e disseminação de falhas.
LO191	Conduzir o registro e o acompanhamento de falhas, provendo suporte para cada uma das atividades.	CSCT [V&V-e]				Uso	3	Verificar como as falhas são reportadas no projeto. Identificar uma falha já corrigida e reportar todo o processo envolvido na sua correção. Reportar uma falha e acompanhar a sua correção fornecendo todas as informações necessárias.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO193	Demonstrar a habilidade de aplicar métodos múltiplos para desenvolver estimativas de confiabilidade para um software.	CSLO [SR-e]				Uso	3	Aplicar vários métodos de estimativas de confiabilidade para o projeto.
LO194	Comparar as características de três abordagens de modelagem de confiabilidade diferentes.	CSLO [SR-e]				Familiaridade	3	Descrever os objetivos, vantagens e desvantagens de cada abordagem de modelagem de confiabilidade quando aplicadas ao projeto.
LO196	Aplicar métricas de qualidade.			(STAMELOS, 2009)		Uso	3	Aplicar métricas de qualidade no projeto. Discutir resultados obtidos. Aplicar as métricas em vários projetos diferentes, comparar e analisar os resultados.
LO197	Descrever o papel que ferramentas podem executar na validação de software.	CSLO [V&V-c2]				Familiaridade	3	Mostrar como ferramentas podem auxiliar na validação de software, usando o projeto como exemplo.
LO198	Usar uma ferramenta de acompanhamento de defeitos para gerenciar os defeitos em um software de pequeno porte.	CSLO [V&V-c2]				Uso	3	Identificar a ferramenta utilizada pela comunidade do projeto para reportar os erros e acompanhar a correção dos mesmos. Reportar ao menos um erro usando a ferramenta. Corrigir ao menos um erro e reportar a solução na ferramenta.
LO297	Discutir ao menos um padrão nacional ou internacional de projeto de interface de usuário.	CSLO [DI-c2]				Familiaridade	3	Usar ferramentas que validem se o padrão de interface está sendo utilizado em diferentes projetos OS.
Prática profissional em Engenharia de Software								
LO199	Distinguir comportamento profissional ético e não ético.		(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(STAMELOS, 2009) (KAMTHAN, 2007)		Familiaridade	2	Discutir comportamento profissional e ético usando o movimento *Open Source* como exemplo.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO200	Identificar questões éticas que surgem no desenvolvimento de software e que determinar como lidar tecnicamente e eticamente com elas.	CSLO [PE-c1]				Familiaridade	2	Listar quais os comportamentos não seriam éticos em um OSP. Comparar com o comportamento em um projeto proprietário.
LO201	Explicar a responsabilidade ética de garantir a correção, confiabilidade e segurança (confidencialidade) do software.	CSLO [PE-c1]				Familiaridade	2	Discutir as responsabilidades assumidas com relação à correção, confiabilidade, segurança no projeto. Entender contratos de uso de projetos de código aberto e proprietários.
LO202	Descrever os mecanismos que tipiciamente existem para que os profissionais mantenham-se atualizados.	CSLO [PE-c1]				Familiaridade	1	Visualizar as diversas tecnologias envolvidas no desenvolvimento de OSP.
LO326	Desenvolver o comprometimento com a aprendizagem contínua.	CSCG	(RADERMACHER; WALIA; KNUDSON, 2014)	(REKHA et al., 2009) (CARRINGTON, 2003)		Familiaridade	1	Ter noção da necessidade de sempre manter-se atualizado por meio da constatação das diversas tecnologias envolvidas no desenvolvimento de OSP, ou mesmo, das mudanças constantes na tecnologia.
LO299	Descrever os pontos positivos e negativos de códigos profissionais relevantes como expressões de profissionalismo e diretrizes para a tomada de decisão.	CSLO [PE-c1]				Familiaridade	0	
LO300	Avaliar códigos de ética profissional provenientes da ACM, IEEE e outras organizações.	CSLO [PE-c1]				Avaliação	0	
LO301	Identificar estágios progressivos em um *whistle-blowing incident*.	CSLO [PE-c2]				Familiaridade	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO302	Identifcar exemplos de como culturas regionais relacionam-se com dilemas éticos.	CSLO [PE-c2]				Familiaridade	0	
LO303	Investigar formas de assédio e discriminação e caminhos de assistência.	CSLO [PE-c2]				Uso	0	
LO304	Examinar várias formas de credenciamento profissional.	CSLO [PE-c2]				Uso	0	
LO305	Explicar a relação entre ergonomia no ambiente computacional e saúde das pessoas.	CSLO [PE-c2]				Familiaridade	0	
LO306	Desenvolver uma política de uso da computação aceitável e aplicável com medidas exigidas.	CSLO [PE-c2]				Avaliação	0	
LO203	Compreender as expectativas de trabalho (habilidades esperadas), questões legais, responsabilidade profissional.	CSCG	(RADERMACHER; WALIA; KNUDSON, 2014)	(GHOSH; GLOTT, 2005) (STAMELOS, 2009)		Familiaridade	1	Identificar no *issue tracker* quais são os perfis dos participantes da comunidade do projeto e quais são suas responsabilidades. Identificar quais são as atividades realizadas pelos desenvolvedores do software e consequentemente as habilidades que precisam ser desenvolvidas.
LO204	Analisar uma questão de computação global, obersvando o papel dos profissionais e governo em gerenciar esse problema.	CSLO [PE-c1]				Avaliação	0	
LO205	Descrever maneiras pelas quais profissionais podem contribuir com políticas públicas.	CSLO [PE-c2]				Familiaridade	3	Discutir possíveis projetos de código aberto que atendam às políticas públicas.
LO206	Descrever as consequências de comportamento profissional não apropriado.	CSLO [PE-c2]				Familiaridade	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO207	Descrever questões associadas com a pressão da indústria com relação a atender o tempo do mercado versus exigências de padrões de qualidade profissional.	CSLO [PE-c2]				Familiaridade	1	Analisar o histórico de lançamento de *releases* e discutir sobre a necessidade de sempre estar disponibilizando novas versões para o mercado, quais as razões e quais as consequências.
LO208	Discutir fundamentos filosóficos de propriedade intelectual.	CSLO [IP-c1]		(GERMAN, 2005)		Familiaridade	3	Aproveitar o contexto de OSP para discutir sobre propriedade intelectual.
LO209	Discutir as razões para proteções legais da propriedade intelectual.	CSLO [IP-c1]		(GERMAN, 2005)		Familiaridade	3	Aproveitar o contexto de OSP para discutir sobre as legislações de proteção à propriedade intelectual.
LO210	Descrever a legislação destinada às violações de direitos autorais. Compreender a legislação de direitos autorais.	CSLO [IP-c1]		(GHOSH; GLOTT, 2005)		Familiaridade	3	Aproveitar o contexto de OSP para discutir sobre a legislação sobre direitos autorais.
LO211	Criticar a legislação destinada às violações de direitos autorais.	CSLO [IP-c1]				Avaliação	3	Aproveitar o contexto de OSP para discutir sobre a legislação sobre direitos autorais.
LO212	Identificar exemplos contemporâneos de propriedade intelectual digital intangível.	CSLO [IP-c1]				Familiaridade	0	
LO213	Justificar o uso de material com direitos autorais.	CSLO [IP-c1]				Avaliação	3	Aproveitar o contexto de OSP para discutir sobre o uso de material com direitos autorais.
LO214	Avaliar questões éticas inerentes aos vários mecanismos de detecção de plágio.	CSLO [IP-c1]				Avaliação	0	
LO215	Interpretar os objetivos e implementações de licenças de software.	CSLO [IP-c1]		(GERMAN, 2005) (KAMTHAN, 2007) (GHOSH; GLOTT, 2005) (HORSTMANN, 2009)		Familiaridade	3	Identificar os tipos de licenças de OSP de sucesso, como Linux, Mozilla Firefox, Apache, etc. Descrever os tipos de licenças mais utilizados pela comunidade de projetos de código aberto. Discutir sobre os demais tipos de licenças de software existentes.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO216	Discutir as questões envolvidas nas leis sobre patentes de software.	CSLO [IP-c1]		(GHOSH; GLOTT, 2005)		Familiaridade	3	Aproveitar o contexto de OSP para discutir sobre a legislação de patentes.
LO217	Caracterizar e constratar os conceitos de copyright (direitos autorais), patentes e trademarks.	CSLO [IP-c1]		(GHOSH; GLOTT, 2005) (HORSTMANN, 2009)		Avaliação	3	Aproveitar o contexto de OSP para discutir conceitos como copyright, patente e trademark, o que significam, objetivos e diferenças.
LO218	Identificar os objetivos do movimento open source.	CSLO [IP-e]		(MORELLI; LANEROLLE, 2009) (GERMAN, 2005) (HORSTMANN, 2009) (HISLOP; ELLIS; MORELLI, 2009) (CARRINGTON, 2003) (LANEROLLE et al., 2008) (AUER; JUNTUNEN; OJALA, 2011)		Familiaridade	3	Identificar projetos de código aberto de sucesso. Discutir sobre o que é um OSP e quais as nomenclaturas utilizadas. Envolver os alunos com leituras e discussões sobre o movimento open source e seu impacto para a sociedade Explicar a diferença entre o desenvolvimento de projetos de código aberto e projetos proprietários. Discutir com os alunos o negócio envolvido em um OSP, qual a sua relação com a indústria e possibilidades futuras.
LO219	Identificar a natureza global da pirataria de software.	CSLO [IP-e]		(CARRINGTON, 2003)		Familiaridade	3	Discutir sobre pirataria de software no contexto do movimento *open source*. Discutir sobre pirataria de software, como ocorre, possíveis causas.
LO220	Identificar maneiras de ser um profissional com práticas sustentáveis.	CSLO [S-c1]				Familiaridade	1	Discutir com os alunos como os profissionais de TI podem ter um comportamento de sustentabilidade. Procurar projetos OS que promovam a sustentabilidade.
LO221	Descrever os impactos das escolhas de design no campo da computação para o meio ambiente, considerando o projeto de algoritmos, projeto de sistemas operacionais, redes, bancos de dados, etc.	CSLO [S-c2]				Familiaridade	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO222	Investigar os impactos sociais e ambientais de novos designs para sistemas.	CSLO [S-c2]				Uso	0	
LO223	Identificar diretrizes para o design e a entrega de tecnologia da informação sustentável.	CSLO [S-e]				Familiaridade	0	
LO224	Investigar a pervasividade da computação em áreas como sistemas energéticos inteligentes, redes sociais, transportes, agricultura, cadeia de suprimentos, monitoramento do meio ambiente e ativismo.	CSLO [S-e]				Uso	0	
LO225	Desenvolver aplicações de computação e avaliar através de pesquisas em relação às questões ambientais. (energia, poluição, uso de recursos, reciclagem e reuso, etc.)	CSLO [S-e]				Avaliação	0	
LO226	Ter consciência da extensa aplicabilidade da computação e o senso de responsabilidade social.	CSCG				Familiaridade	1	Discutir o impacto do uso do computador na sociedade e a sua responsabilidade social. Descrever como projetos OS ajudam a computação a suprir sua responsabilidade social.
LO307	Ilustrar o impacto social e ambiental global do uso e o descarte de computadores.	CSLO [S-c1]				Uso	0	
LO308	Listar os efeitos sustentáveis de trabalhar em casa e realizar compras pela web.	CSLO [S-e]				Familiaridade	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO227	Habilidade de trabalhar efetivamente como membro de equipes.	CSCG	(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(REKHA et al., 2009) (KUSSMAUL, 2009) (ELLIS et al., 2007)		Uso	3	Os alunos podem ser encorajados a participar da comunidade. As atividades podem ser realizadas localmente em grupo. Os alunos podem ser encorajados tanto a colaborar com a própria equipe, como com outras equipes.
LO228	Habilidade de trabalhar com membro de uma equipe multidisciplinar.		(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(XING, 2010) (MARMORSTEIN, 2011) (ELLIS et al., 2007) (STROULIA et al., 2011) (HISLOP; ELLIS; MORELLI, 2009)		Uso	3	Participação da comunidade do projeto (permite a interação com desenvolvedores experientes e usuários). Cada membro do grupo pode assumir perfil diferente (analista de requisitos, projetista, testador), assim o aluno pode compreender o papel de cada desenvolvedor na equipe.
LO229	Discutir maneiras de influenciar o desempenho e os resultados entre equipes de diversas culturas.	CSLO [PC-e]		(HISLOP; ELLIS; MORELLI, 2009) (MARMORSTEIN, 2011)		Familiaridade	3	Discutir fatores que influenciam o desempenho e os resultados em uma equipe multi-cultural a partir da experiência própria com contribuições para a comunidade do projeto.
LO230	Avaliar pontos positivos e negativos pessoais em trabalhar remotamente como parte de uma equipe multinacional.	CSLO [PC-e]		(HISLOP; ELLIS; MORELLI, 2009) (MARMORSTEIN, 2011)		Avaliação	3	O aluno contribui para a comunidade, realizando suas atividades remotamente, assim ele pode descrever quais são os seus pontos fortes e fracos nessa situação.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO231	Compreender como é o trabalho com pessoas de diferentes culturas.			(GHOSH; GLOTT, 2005)		Uso	3	A participação na comunidade faz com que o aluno trabalhe com pessoas de diferentes culturas, desse modo ele pode detectar possíveis diferenças culturais. Investigar a origem dos principais participantes da comunidade; Verificar como esses membros se comportam por meio da sua comunicação com a comunidade; Identificar questões culturais porventura existentes. Comparar artefatos produzidos por pessoas de países diferentes.
LO232	Expressar opiniões próprias.			(GHOSH; GLOTT, 2005)		Avaliação	3	O aluno deve discutir sua opinião localmente e / ou apresentar sua opinião para a comunidade.
LO233	Aceitar e responder críticas de terceiros (habilidade de enfrentar e aprender a partir de erros e estar aberto – receptivo a respostas).		(RADERMACHER; WALIA, 2013)	(GHOSH; GLOTT, 2005)		Uso	3	Comportamento com relação ao *feedback* dado pela comunidade a sua contribuição. Localmente pode haver avaliação por pares.
LO234	Coordenar o próprio trabalho com relação ao trabalho dos outros (habilidade de gerenciamento de tempo e auto-organização).		(RADERMACHER; WALIA; KNUDSON, 2014)	(GHOSH; GLOTT, 2005)		Uso	3	A contribuição do aluno precisa atender a data definida para o lançamento de uma nova *release* do projeto. Para um subprojeto local, que pode ser desenvolvido em equipe, então o aluno precisará atender as datas e padrões definidos para o mesmo. O professor pode fazer com que as equipes precisem interagir umas com as outras.
LO235	Avaliar o trabalhos desenvolvido por terceiros (revisão por pares).			(GHOSH; GLOTT, 2005)		Uso	3	Avaliar como determinada funcionalidade foi implementada ou um erro foi corrigido Localmente, revisar os trabalhos realizados pelos colegas.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO236	Habilidade de ser próativo.		(RADERMACHER; WALIA; KNUDSON, 2014)			Uso	3	Buscar soluções para questões levantadas no *issue tracker*. Propor soluções localmente ou para a comunidade. Buscar soluções para problemas levantados pelo professor, considerando o projeto. Como o aluno trabalha remotamente, ele precisa desenvolver sua independência.
LO237	Habilidade de ser criativo.			(WHITEHURST, 2009)		Uso	3	O professor pode solicitar que os alunos proponham novas soluções para o projeto e discuta-as localmente ou com a comunidade. A liberdade de manipulação do código pode desenvolver a criatividade.
LO241	Demonstrar pensamento crítico e capacidade de gerar e sintetizar novas ideias.		(RADERMACHER; WALIA, 2013)			Uso	3	O professor pode solicitar que os alunos proponham novas soluções para o projeto e discuta-as localmente ou com a comunidade.
LO238	Desenvolver a própria motivação – apresentar disposição (ser automotivado).		(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(REKHA et al., 2009)		Uso	1	O aluno pode sentir-se motivado em contribuir para um projeto real, que tem usuários ativos e que tem continuidade, ou mesmo, pela possibilidade de sua contribuição ter visibilidade externa. Por outro lado, o aluno pode ter dificuldade em se integrar a uma comunidade ou sentir-se sobrecarregado ao tentar entender o projeto.
LO239	Demonstrar paixão por tecnologia ou pelo campo no qual estão trabalhando.		(RADERMACHER; WALIA; KNUDSON, 2014)			-	1	Não vemos como algo que realmente pode ser considerado como objetivo de aprendizagem. Contudo, como os desenvolvedores que participam de OSP são movidos por esta paixão, acreditamos que os estudantes possam ser de algum modo influenciados por essa característica.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO240	Demonstrar habilidades de liderança.		(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(GHOSH; GLOTT, 2005)		Uso	1	Trabalhar em um projeto real, realizando a atividade em equipe, pode desenvolver essas habilidades.
LO242	Habilidade de lidar com incertezas e ambiguidades.				P	Uso	1	O OSP é um projeto real, portando o aluno ao lidar com um domínio e usuários reais, bem como, com restrições tecnológicas, ou seja, um ambiente propício ao surgimento de incertezas e ambiguidades, pode desenvolver essas habilidades.
LO266	Ser capaz de solucionar problemas.	CSCG	(RADERMACHER; WALIA; KNUDSON, 2014) (RADERMACHER; WALIA, 2013)	(HISLOP; ELLIS; MORELLI, 2009)		Uso	3	Ao gerar contribuições para o projeto, o aluno estará solucionando problemas reais do contexto de software.
LO267	Gerar soluções alternativas para os problemas.	CSCG	(RADERMACHER; WALIA, 2013)			Uso	3	Professor pode solicitar que os alunos proponham novas soluções (alternativas) para o projeto e discuta-as localmente ou com a comunidade.
LO268	Ter experiência com código desenvolvido por terceiros.			(CARRINGTON, 2003) (WHITEHURST, 2009) (XING, 2010) (SEITER, 2009)		Uso	3	As atividades de construção, manutenção e testes no OSP, bem como o projeto e implementação de novas funcionalidades vão exigir a leitura de código desenvolvido por terceiros.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO269	Ter vivenciado pelo menos um projeto real.	CSCG	(RADERMACHER; WALIA; KNUDSON, 2014)	(WHITEHURST, 2009) (REKHA et al., 2009) (BUCHTA et al., 2006) (CARRINGTON, 2003) (XING, 2010) (HISLOP; ELLIS; MORELLI, 2009) (ELLIS; MORELLI; HISLOP, 2008a) (ELLIS et al., 2007) (ELLIS et al., 2007)		Uso	3	O OSP é um projeto real, principalmente, quando ele está sendo utilizado e encontra-se evoluindo ao longo do tempo.
LO270	Compreender como teoria e prática influenciam uma a outra.	CSCG				Uso	3	Espera-se como resultado a partir da manipulação de um projeto real.
LO271	Compreender como a computação interage com os diferentes domínios.	CSCG				Uso	3	Compreender como OSP encaixa-se em determinado contexto. Discutir projetos OS que atuem em diversos domínios.
LO309	Descrever como o software pode interagir e participar de vários sistemas.	CSLO [SP-c1]				Familiaridade	3	Enxergar o contexto onde o OSP pode ser usado Buscar projetos OS que participem de outros sistemas, por exemplo, *plugins* do Eclipse.
LO272	Alcançar uma visão holística do desenvolvimento do produto de software.	CSCG	(RADERMACHER; WALIA; KNUDSON, 2014)	(REKHA et al., 2009) (AUER; JUNTUNEN; OJALA, 2011)		Uso	3	Espera-se como resultado, uma vez que o software completo de algum modo está sendo trabalhado. O aluno pode acompanhar toda a evolução do software.
LO273	Ser capaz de pensar o problema em vários níveis de detalhe e abstração.	CSCG		(REKHA et al., 2009)		Uso	2	O aluno enxerga o produto completo – o todo e suas partes. Elaborar modelos que documentem o software em diversos níveis de abstração. (O fato do projeto está pronto pode gerar dificuldade em pensar em modelos mais abstratos.)

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO243	Interagir com outras pessoas (interagir profissionalmente com os stakeholders).	CSCT [PC-c1]		(GHOSH; GLOTT, 2005) (STAMELOS, 2009) (MARMORSTEIN, 2011)		Uso	3	Ao contribuir com o projeto, os alunos precisam interagir com a comunidade. Localmente, os alunos podem interagir com os colegas quando o trabalho é em equipe.
LO244	Articular argumentos claramente (articular pontos de vista e prover explicações claras).		(RADERMACHER; WALIA; KNUDSON, 2014)	(GHOSH; GLOTT, 2005)		Uso	3	Ao reportar um erro (ajudar a comunidade na compreensão das condições de operação que produzem o erro), ao documentar uma parte do projeto, ao propor uma nova funcionalidade...
LO245	Aperfeiçoar o entendimento da língua inglesa, especialmente em discussões técnicas.			(GHOSH; GLOTT, 2005)		Uso	3	Leitura da documentação do projeto e com a interação com a comunidade.
LO246	Escrever documentação, clara, concisa e acurada, seguindo padrões previamente definidos de formato. Incluir tabelas, figuras e referências apropriadas.	CSLO [PC-c1]	(RADERMACHER; WALIA, 2013) (RADERMACHER; WALIA; KNUDSON, 2014)	(STAMELOS, 2009) (KUSSMAUL, 2009)		Uso	3	Ao documentar ou modelar uma parte do projeto.
LO247	Descrever os pontos fortes e fracos de várias formas de comunicação (virtual, face-a-face, documentos compartilhados).	CSLO [PC-c1]				Familiaridade	3	Experimentar os mecanismos de comunicação empregados no projeto. Avaliar as ferramentas de comunicação empregadas no projeto, seus pontos positivos e negativos.
LO248	Planejar interações com outros membros do projeto, de maneira que eles sejam capazes de ouvir e apreciar o ponto de vista de outrem, mesmo quando eles não concordam, e sejam capazes de transmitir aos outros o que eles ouviram.	CSLO [PC-c1]				Uso	3	A participação em um projeto real, desenvolvido por várias pessoas força o aluno a interagir com outras pessoas, ouvir e aceitar diferentes pontos de vista, críticas ao seu trabalho, ou mesmo, retransmitir informações a outros. As interações também podem ocorrer localmente, quando a atividade for desenvolvida em equipe.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO249	Desenvolver a habilidade de solicitar ajuda.		(RADERMACHER; WALIA, 2013)			Uso	3	Aumenta a probabilidade do aluno precisar de ajuda ao lidar com um projeto real. Em OSP isso é uma ação natural através das listas de discussão.
LO250	Usar ferramentas de comunicação como *wikis*, listas de discussão, fóruns de discussão, controle de versão, *issue tracker*.	CSCT [PC-c1]		(ELLIS et al., 2007) (ROBLES; CABALLÉ; GONZÁLEZ-BARAHONA, 2008) (KUSS-MAUL, 2009)		Uso	3	Utilizar as ferramentas disponibilizadas pela comunidade. Professor pode solicitar que o aluno apresente / reporte as interações realizadas com a comunidade. Ferramentas semelhantes podem ser usadas localmente.
LO310	Comparar e constratar várias ferramentas de colaboração.	CSLO [PC-c1]				Avaliação	3	Verificar as ferramentas de colaboração empregadas no projeto. Vericar quais outras ferramentas poderiam ser usadas. Avaliar vantagens e desvantagens de cada uma. Comparar ferramentas usadas em diferentes projetos OS.
LO327	Comunicar-se oralmente de modo adequando com os clientes, escutando suas solicitações e sem usar jargões técnicos.		(RADERMACHER; WALIA, 2013) (RADERMA-CHER; WALIA; KNUDSON, 2014)			Uso	0	
LO25	Leitura, compreensão e sumarização de material técnico (código fonte e documentação).	CSCT [PC-c1]				Uso	3	Necessário para a realizar as outras atividades. Aluno deve apresentar o que existe documentado no projeto.
LO311	Desenvolver e apresentar uma apresentação formal de boa qualidade.	CSLO [PC-c1]	(RADERMACHER; WALIA, 2013)			Avaliação	1	Somente ocorre se o professor solicitar apresentações na sala de aula. Apresentar resultados parciais das atividades realizadas no projeto.
Economia da Engenharia de Software								

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO252	Explicar e aplicar a relação entre SE e negócio. Entender a decisão entre: "construir", "comprar" ou "baixar".			(KUSSMAUL, 2009) (CONLON; HULICK, 2005)		Avaliação	3	Discutir para o contexto de negócio onde existe uma solução de projeto de código aberto disponível, qual é a melhor opção: "construir", "comprar" ou "baixar" o software. Apontar as vantagens e desvantagens de cada opção. Usar exemplos existentes de projetos OS.
LO254	Listar vários exemplos de riscos de software.	CSLO [SPM-c2]				Familiaridade	2	Detectar os riscos existentes para o projeto. Reconhecer exemplos de riscos que são comuns em outros projetos e que não ocorrem no projeto em mãos. Justificar porque não ocorrem. Usar ferramentas e planilhas que ajudem na avaliação de riscos.
LO255	Descrever as diferentes categorias de riscos para sistemas de software.	CSLO [SPM-c2]				Familiariadade	2	Detectar os riscos existentes para o projeto. Classificar esses riscos. Avaliar porque outras categorias de riscos não ocorrem para o projeto em mãos. Avaliar e discutir esses riscos em diferentes projetos OS.
LO256	Demonstrar uma abordagem sistemática para a tarefa de identificar os perigos e riscos de uma situação particular.	CSLO [SPM-e]				Uso	1	Aplicar uma abordagem sistemática para identificar os perigos e riscos do projeto.
LO257	Identificar riscos de segurançaa para um sistema de software.	CSLO [SPM-e]				Uso	3	Identificar os riscos de segurança do projeto.
LO258	Identificar os riscos e descrever abordagens para gerenciar os riscos (evitar, aceitar, transferir, atenuar), e caracterizar os pontos positivos e negativos de cada abordagem.	CSLO [SPM-e]				Familiaridade	1	Detectar os riscos para o projeto. Definir estratégias para gerenciar esses riscos. Usar ferramentas para gerenciar riscos.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO261	Descrever o impacto dos riscos para o ciclo de vida de desenvolvimento do software.	CSLO [SPM-c2]				Familiaridade	1	No histórico do projeto, detectar quais foram os riscos existentes e qual foi o impacto dos mesmos para o desenvolvimento do projeto.
LO259	Explicar como riscos afetam as decisões do processo de desenvolvimento do software.	CSLO [SPM-e]				Uso	1	No histórico do projeto, detectar que tipo de decisões foram influenciadas por riscos. Sugerir o que poderia ser alterado na condução do projeto, por causa dos riscos existentes para o mesmo. Discutir os resultados dessas atividades e consequentemente como os riscos podem afetar a condução do projeto.
LO260	Descrever o impacto da tolerância aos riscos para o processo de desenvolvimento do software.	CSLO [SPM-e]				Avaliação	1	No histórico do projeto, detectar que estratégia de tolerância aos riscos foram adotadas. Identificar quais as consequências da adoção das estratégias para o processo de desenvolvimento do software. Discutir essas consequências. Criar ou atualizar o plano de gerenciamento de riscos. Discutir aplicação do plano durante a manutenção de funcionalidades do software.
LO262	Aplicar os princípios básicos de gerenciamento de riscos em uma variedade de cenários simples, incluindo a situação de segurança.	CSLO [SPM-e]				Uso	2	Aplicar os princípios básicos de gerenciamento de riscos para o projeto. Desenvolver e aplicar plano de gerenciamento de riscos em manutenções realizadas no projeto.

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO263	Conduzir uma análise de custo benefício para uma abordagem de atenuação de risco.	CSLO [SPM-e]				Uso	2	Para um determinado risco do projeto, conduzir uma análise de custo benefício para a abordagem de atenuação escolhida. Realizar análises com opções reais existentes para atenuar riscos em um projeto. (Por exemplo, montar um servidor de backup ou contratar um serviço na nuvem. Qual seria o melhor custo benefício?)
LO264	Identificar e analisar alguns dos riscos para um sistema inteiro, oriundos de outros aspectos, que não o software.	CSLO [SPM-e]				Uso	1	Identificar e analisar os riscos do contexto onde o projeto de software encaixa-se.
LO265	Examinar o balanceamento entre fontes comuns de riscos para projetos de software, considerando tecnologia, estrutura/processo, qualidade, pessoas, mercado e financeiro.	CSLO [PC-e]				Uso	1	Examinar o balanceamento entre fontes comuns de riscos para o projeto, considerando tecnologia, estrutura/ processo, qualidade, pessoas, mercado e financeiro.
LO312	Sumarizar a razão para esforços anti-monopólios.	CSLO [EC-e]				Familiaridade	1	Discutir questões de monopólios usando exemplos como Linux, Mozilla Firefox, Libre-office. Realizar pesquisas e discutir ações realizadas internacionalmente na área.
LO313	Identificar várias formas pelas quais a indústria da tecnologia da informação é afetada pela escassez de mão-de-obra.	CSLO [EC-e]				Familiaridade	0	
LO314	Identificar a evolução das estratégias de preços para produtos e serviços de computação.	CSLO [EC-e]				Familiaridade	0	

Id	Descrição	CS2013	Indústria	Literatura	P	Nível	G	Justificativas
LO315	Discutir benefícios, desvantagens e implicações da terceirização e *off-shoring*.	CSLO [EC-e]				Familiaridade	0	
LO316	Investigar e defender maneiras de lidar com as restrições no acesso à computação.	CSLO [EC-e]				Uso	1	Discutir como a adoção de projetos OS podem favorecer o acesso à computação. Buscar e discutir projetos OS que lidem com facilidades para acesso a computação.
LO317	Descrever os benefícios econômicos de efeitos de rede.	CSLO [EC-e]				Familiaridade	0	

Apêndice

B

QUESTIONÁRIO LET - ESTUDO DE CASO 2

A seguir o formulário utilizado para o levantamento da experiência prévia dos estudantes com projetos reais e atividades de qualidade e/ou testes de software.

Dados iniciais sobre os participantes da pesquisa

O objetivo desse formulário é identificar a sua experiência com relação à participação em projetos reais.

Nome completo:
E-mail:
Curso:
Ano e semestre de ingresso na UFS:

Qual a sua experiência participando de projetos reais de software em outras oportunidades?

Projeto real: projeto criado para atender a necessidade de algum usuário, instituição, etc. Não corresponde a um projeto criado para atender as exigências de uma disciplina.

( ) Até o momento, ainda não tive experiências com projetos reais
( ) Já tive algumas experiências com projetos reais (até 2 projetos)
( ) Já tive várias experiências com projetos reais (3 ou mais projetos)

No caso de ter tido experiência em projetos, você poderia descrever rapidamente para cada um deles: o objetivo, qual o usuário ou instituição beneficiada, as atividades realizadas por você com relação ao projeto e principalmente se você foi responsável por alguma atividade relacionada à qualidade e/ou testes do software.

Obs.: Se você já participou de mais de 3 projetos reais, informe a quantidade de projetos e descreva apenas os 3 que você considerou mais interessantes.

Apêndice

C

QUESTIONÁRIO QT1 - ESTUDO DE CASO 2

O objetivo deste questionário foi capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a testes de software antes da adoção do OSP.

Convém salientar que na introdução do questionário, esclarecemos aos estudantes, o seu propósito e a confidencialidade das informações prestadas. Procuramos enfatizar a importância da sua participação e sinceridade de suas respostas, bem como que estas informações não influenciariam o seu resultado com relação à sua aprovação na disciplina.

Apresentamos as questões no formato de afirmações, onde o estudante tinha como opção não responder a questão ou escolher uma das opções da escala Likert: *concordo plenamente, concordo, discordo ou discordo plenamente*.

Ao final, fornecemos um campo aberto para que o estudante expressasse suas dúvidas, críticas e sugestões.

As afirmações utilizadas no questionário foram as seguintes:

1. Você é capaz de descrever os conceitos e práticas relacionadas ao teste de software.
2. Você é capaz de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).
3. Você é capaz de descrever e distinguir entre as diferentes técnicas caixa-preta para elaboração de casos de teste.
4. Você é capaz de descrever e distinguir entre as diferentes técnicas caixa- branca para elaboração de casos de teste.
5. Você é capaz de criar testes de unidade que não sejam redundantes (aplicar as técnicas de elaboração de casos de teste de modo que os casos gerados não sejam redundantes).
6. Você é capaz de planejar e gerar casos de testes para softwares de médio porte.

7. Você é capaz de criar e documentar um conjunto de testes para um segmento de código de médio porte.
8. Você é capaz de usar ferramentas de testes.
9. Você entende a diferença entre testar pequenos programas e testar software de grande porte.
10. Você é capaz de avaliar uma suíte de testes para um segmento de código de médio porte.
11. Você é capaz de avaliar os testes sendo executados por meio de métricas.
12. Você é capaz de usar ferramentas de teste de cobertura eficientemente.
13. Você é capaz de descrever o processo de testes de regressão e seu papel no gerenciamento de releases.
14. Você é capaz de descrever como selecionar bons testes de regressão e automatiza-los.
15. Você tem algum conhecimento sobre ferramentas de integração continua e testes de regressão.
16. Você é capaz de descrever como é um processo de gerenciamento de defeitos.
17. Você é capaz de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (usar a ferramenta para reportar erros, monitorar os erros e submeter correções).

Apêndice

D

QUESTIONÁRIO QT2 - ESTUDO DE CASO 2

O objetivo deste questionário foi capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a testes de software após a adoção do OSP.

Convém salientar que na introdução do questionário, esclarecemos aos estudantes, o seu propósito e a confidencialidade das informações prestadas. Procuramos enfatizar a importância da sua participação e sinceridade de suas respostas, bem como que estas informações não influenciariam o seu resultado com relação à sua aprovação na disciplina. A intenção era somente avaliar a eficácia do recurso didático aplicado.

Apresentamos as questões no formato de afirmações, onde o estudante tinha como opção não responder a questão ou escolher uma das opções da escala Likert: *concordo plenamente, concordo, discordo ou discordo plenamente*.

Ao final, fornecemos um campo aberto para que o estudante expressasse suas críticas e sugestões de melhorias.

As afirmações utilizadas no questionário foram as seguintes:

1. Você acredita que o projeto empregado (JabRef) permitiu que você tivesse uma experiência de mundo real (vivenciasse um projeto real de software).

2. Você acredita que o projeto empregado (JabRef) representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software.

3. O uso do projeto de código aberto (JabRef) permitiu que você visualizasse a complexidade e as dificuldades existentes em um projeto real de software.

4. A participação no projeto permitiu que você compreendesse melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software.

5. Ao realizar as atividades, você conseguiu compreender como o conhecimento teórico sobre testes de software é utilizado na execução de um projeto real.

6. As atividades realizadas mostraram-lhe como a prática em projetos reais pode ser importante para aprender sobre testes de software.

7. Depois de trabalhar no projeto, você está confiante de que será capaz de realizar atividades semelhantes em outro projeto.

8. A experiência com a realização das atividades no projeto vai contribuir para a melhoria de seu desempenho futuramente na vida profissional.

9. As atividades realizadas no projeto contribuíram para você sentir-se mais preparado tecnicamente para o mercado de trabalho, considerando as atividades de testes de software.

10. Você é capaz de descrever os conceitos e práticas relacionadas ao teste de software.

11. A participação no projeto contribuiu para que você seja capaz de descrever os conceitos e práticas relacionadas ao teste de software.

12. Você é capaz de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).

13. A participação no projeto contribuiu para que você seja capaz de descrever e distinguir entre tipos e níveis diferentes de testes (unidade, integração, sistema e aceitação).

14. Você é capaz de descrever e distinguir entre as diferentes técnicas caixa-preta para elaboração de casos de teste.

15. A participação no projeto contribuiu para que você seja capaz de descrever e distinguir entre as diferentes técnicas caixa-preta para elaboração de casos de teste.

16. Você é capaz de descrever e distinguir entre as diferentes técnicas caixa- branca para elaboração de casos de teste.

17. A participação no projeto contribuiu para que você seja capaz de descrever e distinguir entre as diferentes técnicas caixa- branca para elaboração de casos de teste.

18. Você é capaz de criar testes de unidade que não sejam redundantes (aplicar as técnicas de elaboração de casos de teste de modo que os casos gerados não sejam redundantes).

19. A participação no projeto contribuiu para que você seja capaz de criar testes de unidade que não sejam redundantes (aplicar as técnicas de elaboração de casos de teste de modo que os casos gerados não sejam redundantes).

20. Você é capaz de planejar e gerar casos de testes para softwares de médio porte.

21. A participação no projeto contribuiu para que você seja capaz de planejar e gerar casos de testes para softwares de médio porte.

22. Você é capaz de criar e documentar um conjunto de testes para um segmento de código de médio porte.

23. A participação no projeto contribuiu para que você seja capaz de criar e documentar um conjunto de testes para um segmento de código de médio porte.

24. Você é capaz de usar ferramentas de testes.

25. A participação no projeto contribuiu para que você seja capaz de usar ferramentas de testes.

26. Você entende a diferença entre testar pequenos programas e testar software de grande porte.

27. A participação no projeto contribuiu para que você entendesse melhor a diferença entre testar pequenos programas e testar software de grande porte.

28. Você é capaz de avaliar uma suíte de testes para um segmento de código de médio porte.

29. A participação no projeto contribuiu para que você seja capaz de avaliar uma suíte de testes para um segmento de código de médio porte.

30. Você é capaz de avaliar os testes sendo executados por meio de métricas.

31. A participação no projeto contribuiu para que você seja capaz de avaliar os testes sendo executados por meio de métricas.

32. Você é capaz de usar ferramentas de teste de cobertura eficientemente.

33. A participação no projeto contribuiu para que você seja capaz de usar ferramentas de teste de cobertura eficientemente.

34. Você é capaz de descrever o processo de testes de regressão e seu papel no gerenciamento de releases.

35. A participação no projeto contribuiu para que você seja capaz de descrever o processo de testes de regressão e seu papel no gerenciamento de releases.

36. Você é capaz de descrever como selecionar bons testes de regressão e automatiza-los.

37. A participação no projeto contribuiu para que você seja capaz de descrever como selecionar bons testes de regressão e automatiza-los.

38. Você tem algum conhecimento sobre ferramentas de integração continua e testes de regressão.

39. A participação no projeto contribuiu para que você tivesse algum conhecimento sobre ferramentas de integração continua e testes de regressão.

40. Você é capaz de descrever como é um processo de gerenciamento de defeitos.

41. A participação no projeto contribuiu para que você seja capaz de descrever como é um processo de gerenciamento de defeitos.

42. Você é capaz de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (usar a ferramenta para reportar erros, monitorar os erros e submeter correções).

43. A participação no projeto contribuiu para que você seja capaz de usar uma ferramenta de rastreamento de defeitos para gerenciar os defeitos em um software de pequeno porte (usar a ferramenta para reportar erros, monitorar os erros e submeter correções).

44. A participação no projeto contribuiu para que você desenvolvesse a habilidade de aceitar de maneira construtiva as críticas realizadas ao seu trabalho.

45. A participação no projeto contribuiu para que você desenvolvesse a habilidade de avaliar o trabalho desenvolvido por terceiros.

46. A participação no projeto contribuiu para que você tivesse uma experiência com código desenvolvido por terceiros.

47. A participação no projeto contribuiu para que você desenvolvesse a habilidade de ser proativo.

48. A participação no projeto contribuiu para que você desenvolvesse a habilidade de ser criativo.

49. A participação no projeto contribuiu para que você desenvolvesse a habilidade de solucionar problemas.

50. A participação no projeto contribuiu para que você desenvolvesse a habilidade de gerar soluções alternativas para os problemas encontrados.

51. A participação no projeto contribuiu para que você tivesse uma visão do processo como um todo do desenvolvimento de software.

52. A participação no projeto contribuiu para que você desenvolvesse a habilidade de compreender material técnico (código fonte e documentação).

53. A participação no projeto contribuiu para que você desenvolvesse a habilidade de sumarizar material técnico (código fonte e documentação).

54. A participação no projeto contribuiu para que você desenvolvesse a habilidade de escrever documentação clara, concisa e acurada.

55. A participação no projeto contribuiu para que você compreendesse a importância de da especificação de requisitos.

56. A participação no projeto contribuiu para que você compreendesse a importância de um software bem documentado (modelos, documentação do código etc.)

57. A participação no projeto contribuiu para que você compreendesse a importância da construção de testes automatizados dentro do processo de desenvolvimento e evolução do software.

58. A participação no projeto contribuiu para que você compreendesse a importância de aplicar técnicas apropriadas para testar o software.

59. A participação no projeto contribuiu para que você compreendesse a importância da criação de casos de testes de maneira sistemática.

Apêndice

E

QUESTIONÁRIO LER – ESTUDO DE CASO 3

A seguir o formulário utilizado para o levantamento da experiência prévia dos estudantes com projetos reais.

O objetivo desse formulário é identificar a sua experiência com relação à participação em projetos reais.

Nome completo:
E-mail:
Curso:
Ano e semestre de ingresso na UFS:

Qual a sua experiência participando de projetos reais de software em outras oportunidades?

Projeto real: projeto criado para atender a necessidade de algum usuário, instituição, etc. Não corresponde a um projeto criado para atender as exigências de uma disciplina.

( ) Até o momento, ainda não tive experiências com projetos reais
( ) Já tive algumas experiências com projetos reais (até 2 projetos)
( ) Já tive várias experiências com projetos reais (3 ou mais projetos)

No caso de ter tido experiência em projetos, você poderia descrever rapidamente para cada um deles: o objetivo, qual o usuário ou instituição beneficiada, as atividades realizadas por você com relação ao projeto.

Obs.: Se você já participou de mais de 3 projetos reais, informe a quantidade de projetos e descreva apenas os 3 que você considerou mais interessantes.

Apêndice

F

QUESTIONÁRIO QR1 - ESTUDO DE CASO 3

O objetivo deste questionário foi capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a requisitos de software antes da adoção do OSP.

Convém salientar que na introdução do questionário, esclarecemos aos estudantes, o seu propósito e a confidencialidade das informações prestadas. Procuramos enfatizar a importância da sua participação e sinceridade de suas respostas, bem como que estas informações não influenciariam o seu resultado com relação à sua aprovação na disciplina.

Apresentamos as questões no formato de afirmações, onde o estudante tinha como opção não responder a questão ou escolher uma das opções da escala Likert: *concordo plenamente, concordo, discordo ou discordo plenamente*.

Ao final, fornecemos um campo aberto para que o estudante expressasse suas dúvidas, críticas e sugestões.

As afirmações utilizadas no questionário foram as seguintes:

1. Você é capaz de modelar e especificar os requisitos de software de médio porte, usando métodos comuns (ou seja, não formal/matemático).

2. Você é capaz de usar ferramentas de análise de requisitos em suporte ao desenvolvimento de um software de médio porte.

3. Você é capaz de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte.

4. Você é capaz de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.

5. Você é capaz de comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.

6. Você é capaz de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos.

7. Você é capaz de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.

8. Você é capaz de descrever os principais componentes de um caso de uso para algum comportamento requerido de um software.

9. Você é capaz de diferenciar o rastreamento de 'construção' (realizado a partir dos requisitos) e 'recuperação' (realizado a partir do código) e explicar seus papeis no processo de validação dos requisitos.

Apêndice

G

QUESTIONÁRIO QR2 - ESTUDO DE CASO 3

O objetivo deste questionário foi capturar a percepção dos estudantes sobre o conhecimento e habilidades possuídas com relação a requisitos de software após a adoção do OSP.

Convém salientar que na introdução do questionário, esclarecemos aos estudantes, o seu propósito e a confidencialidade das informações prestadas. Procuramos enfatizar a importância da sua participação e sinceridade de suas respostas, bem como que estas informações não influenciariam o seu resultado com relação à sua aprovação na disciplina. A intenção era somente avaliar a eficácia do recurso didático aplicado.

Apresentamos as questões no formato de afirmações, onde o estudante tinha como opção não responder a questão ou escolher uma das opções da escala Likert: *concordo plenamente, concordo, discordo ou discordo plenamente*.

Ao final, fornecemos um campo aberto para que o estudante expressasse suas críticas e sugestões de melhorias.

As afirmações utilizadas no questionário foram as seguintes:

1. Você acredita que o projeto empregado (JabRef) permitiu que você tivesse uma experiência de mundo real (vivenciasse um projeto real de software).

2. Você acredita que o projeto empregado (JabRef) representa um exemplo de tamanho e complexidade semelhantes aos que são trabalhados na indústria de software.

3. O uso do projeto de código aberto (JabRef) permitiu que você visualizasse a complexidade e as dificuldades existentes em um projeto real de software.

4. A participação no projeto permitiu que você compreendesse melhor as habilidades necessárias ao profissional que trabalha no desenvolvimento e manutenção de software.

5. Ao realizar as atividades, você conseguiu compreender como o conhecimento teórico sobre requisitos de software é utilizado na execução de um projeto real.

6. As atividades realizadas mostraram-lhe como a prática em projetos reais pode ser importante para aprender sobre requisitos de software.

7. Depois de trabalhar no projeto, você está confiante de que será capaz de realizar atividades semelhantes em outro projeto

8. A experiência com a realização das atividades no projeto vai contribuir para a melhoria de seu desempenho futuramente na vida profissional.

9. As atividades realizadas no projeto contribuíram para você sentir-se mais preparado tecnicamente para o mercado de trabalho, considerando as atividades relacionadas a requisitos de software.

10. Você é capaz de modelar e especificar os requisitos de software de médio porte, usando métodos comuns (ou seja, não formal/matemático).

11. A participação no projeto contribuiu para que você seja capaz de modelar e especificar os requisitos de software de médio porte, usando métodos comuns (ou seja, não formal/matemático).

12. Você é capaz de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte.

13. A participação no projeto contribuiu para que você seja capaz de aplicar a notação UML na especificação de requisitos de uma aplicação de médio porte.

14. Você é capaz de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.

15. A participação no projeto contribuiu para que você seja capaz de descrever como o processo de engenharia de requisitos provê a elicitação e a validação de requisitos comportamentais.

16. Você é capaz de comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.

17. A participação no projeto contribuiu para que você seja capaz de comparar as abordagens direcionada por planos (Cascata) e ágil, para a especificação e análise de requisitos, descrevendo os benefícios e riscos associados a cada uma.

18. Você é capaz de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos.

19. A participação no projeto contribuiu para que você seja capaz de descrever a importância e as atividades envolvidas no gerenciamento de mudanças de requisitos.

20. Você é capaz de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.

21. A participação no projeto contribuiu para que você seja capaz de identificar requisitos funcionais e não funcionais na especificação de requisitos de um software.

22. Você é capaz de diferenciar o rastreamento de 'construção' (realizado a partir dos requisitos) e 'recuperação' (realizado a partir do código) e explicar seus papeis no processo de validação dos requisitos.

23. A participação no projeto contribuiu para que você seja capaz de diferenciar o rastreamento de 'construção' (realizado a partir dos requisitos) e 'recuperação' (realizado a partir do código) e explicar seus papeis no processo de validação dos requisitos.

24. A participação no projeto contribuiu para que você desenvolvesse a habilidade de aceitar de maneira construtiva as críticas realizadas ao seu trabalho.

25. A participação no projeto contribuiu para que você desenvolvesse a habilidade de avaliar o trabalho desenvolvido por terceiros.

26. A participação no projeto contribuiu para que você tivesse uma experiência com código desenvolvido por terceiros.

27. A participação no projeto contribuiu para que você desenvolvesse a habilidade de ser proativo.

28. A participação no projeto contribuiu para que você desenvolvesse a habilidade de ser criativo.

29. A participação no projeto contribuiu para que você desenvolvesse a habilidade de solucionar problemas.

30. A participação no projeto contribuiu para que você desenvolvesse a habilidade de gerar soluções alternativas para os problemas encontrados.

31. A participação no projeto contribuiu para que você tivesse uma visão do processo como um todo do desenvolvimento de software.

32. A participação no projeto contribuiu para que você desenvolvesse a habilidade de compreender material técnico (código fonte e documentação).

33. A participação no projeto contribuiu para que você desenvolvesse a habilidade de sumarizar material técnico (código fonte e documentação).

34. A participação no projeto contribuiu para que você desenvolvesse a habilidade de escrever documentação clara, concisa e acurada.

35. A participação no projeto contribuiu para que você compreendesse a importância da especificação de requisitos.

36. A participação no projeto contribuiu para que você compreendesse a importância de um software bem documentado (modelos, documentação do código etc).

37. A participação no projeto contribuiu para que você compreendesse a importância da engenharia de requisitos.