UNIVERSIDADE FEDERAL DA BAHIA

DISSERTAÇÃO DE MESTRADO

UMA SÉRIE DE EXPERIMENTOS SOBRE A RELAÇÃO ENTRE COESÃO E O ESFORÇO DE COMPREENSÃO DE PROGRAMAS

Elienai Bitencourt Batista

Programa de Pós-Graduação em Ciência da Computação

Salvador
08 de Junho de 2016

PGCOMP-Msc-2016

ELIENAI BITENCOURT BATISTA

UMA SÉRIE DE EXPERIMENTOS SOBRE A RELAÇÃO ENTRE COESÃO E O ESFORÇO DE COMPREENSÃO DE PROGRAMAS

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Cláudio Nogueira Sant'Anna

Salvador
08 de Junho de 2016

Ficha catalográfica.

BATISTA, ELIENAI BITENCOURT

UMA SÉRIE DE EXPERIMENTOS SOBRE A RELAÇÃO ENTRE COESÃO E O ESFORÇO DE COMPREENSÃO DE PROGRAMAS/ Elienai Bitencourt Batista– Salvador, 08 de Junho de 2016.

117p.: il.

Orientador: Cláudio Nogueira Sant'Anna.
DISSERTAÇÃO DE MESTRADO– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 08 de Junho de 2016.

1. MÉTRICAS DE SOFTWARE. 2. COESÃO. 3. COMPREENSÃO DE PROGRAMAS. 4. EXPERIMENTOS.
I. SANT'ANNA, CLÁUDIO NOGUEIRA. II. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. III Título.

NUMERO CDD

TERMO DE APROVAÇÃO

ELIENAI BITENCOURT BATISTA

UMA SÉRIE DE EXPERIMENTOS SOBRE A RELAÇÃO ENTRE COESÃO E O ESFORÇO DE COMPREENSÃO DE PROGRAMAS

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 08 de Junho de 2016

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

Profa. Dra. Christina von Flach Garcia Chavez
Universidade Federal da Bahia

Prof. Dr. Renato Lima Novais
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

À minha família,

AGRADECIMENTOS

Em primeiro lugar, ao meu bom Deus que me concedeu a oportunidade de realizar esse mestrado e me deu forças para seguir até o fim, me cercando com pessoas maravilhosas que me ajudaram muito.

Ao meu orientador, professor Cláudio Nogueira Sant'Anna, por ter me direcionado durante toda pesquisa e por ter me incentivado nas publicações, contribuindo grandemente para meu crescimento intelectual.

Ao meu companheiro, amigo e maridão, Amaray da Silva Mota Júnior, por todo apoio, paciência, conselhos, carinho e amor.

Aos meus pais, Manoel Batista e Marizete Batista, pelo incentivo, encorajamento, educação e carinho.

Ao colega Bruno Silva, pelos conselhos, pelo exemplo e orientação constante.

À amiga, parceira e pesquisadora, Adriana Viriato Ribeiro, pelo companheirismo, pelas conversas, pelas dicas, por toda a ajuda e pela participação no estudo piloto.

Às amigas e pesquisadoras, Antônia Tamires Alves e Jhennifer Souza, pela participação no estudo piloto, pelas risadas, pelas conversas e pelas trocas de experiências.

Por fim, agradeço a Fapesb pelo apoio financeiro.

Aquele que leva a preciosa semente, andando e chorando, voltará, sem dúvida, com alegria, trazendo consigo os seus molhos.

—SALMOS 126:6 (Bíblia Sagrada)

RESUMO

Coesão pode ser definida como o grau pelo qual um módulo de um software está focado na realização de uma única responsabilidade. Acredita-se que a coesão pode interferir em atributos de qualidade de software, tais como, manutenibilidade e facilidade de compreensão. Em particular, a literatura de engenharia de software afirma que quanto maior for a coesão, mais fácil será para se compreender o código fonte de um módulo. Pesquisadores têm definido uma série de métricas para quantificar coesão. No entanto, faltam evidências experimentais sobre a existência de uma relação entre coesão, quantificada por meio de métricas, e o esforço para se compreender programas. Diante desse contexto, realizamos três quase-experimentos para avaliar a relação entre medidas de coesão e o esforço para se compreender o código fonte de classes de sistemas orientados a objetos. Além disso, avaliamos se dois diferentes tipos de métricas – estrutural e conceitual – apresentam relação diferente com o esforço de compreensão. Nesses experimentos, participantes executaram atividades que demandaram a compreensão do código fonte de diferentes classes com diferentes graus de coesão. Os resultados mostraram que a coesão não teve impacto na compreensibilidade de programas. Isso ocorreu principalmente porque a influência muito mais forte de outros atributos do código fonte acaba minimizando a influência da coesão.

Palavras-chave: Métricas de software; Coesão; Compreensão de programas; Experimentos.

ABSTRACT

Cohesion can be defined as the degree to which a software module is focused on implementing only one responsibility. Cohesion is believed to impact on software quality attributes, such as maintainability and understandability. In particular, software engineering literature claims that the more cohesive a module is, the easier it will be to understand its source code. Researchers have defined a series of metrics for quantifying cohesion. However, there is no empirical evidence about the relationship between cohesion, quantified by means of metrics, and the effort for understanding programs. In the context, we conducted three quasi-experiments to evaluate the relation between cohesion measurements and the effort spent to understand the source code of classes in object-oriented software. Moreover, we evaluated whether two different types of metrics – structural and conceptual – presented distinct relation with comprehension effort. In out experiments, participants carried out tasks that required them to understand the source code of classes with different degrees of cohesion. Our findings showed that the cohesion did not impact on program understandability. This occurred mainly because the much stronger influence of other attributes of the source code ended up minimizing the influence of cohesion.

Keywords: Software metrics; Cohesion; Program comprehension; Experiments.

SUMÁRIO

LISTA DE FIGURAS

Capítulo 1

INTRODUÇÃO

Coesão é um conceito bem conhecido e usado como atributo de qualidade interna de *design* do software. A coesão de um módulo é o grau pelo qual aquele módulo está focado em implementar apenas uma responsabilidade do sistema (Pflegger e Atlee, 2010; Sommerville, 2001). De acordo com a literatura, um software bem projetado tem os módulos bem coesos. Acredita-se que o grau de coesão dos módulos de um sistema pode influenciar atributos de qualidade externa importantes, como manutenibilidade, facilidade de compreensão e facilidade de reutilização (Pflegger e Atlee, 2010).

Várias métricas para quantificar coesão têm sido propostas no contexto de software orientado a objetos (Chidamber e Kemerer, 1991; Chidamber e Kemerer, 1994; Bieman e Kang, 1995; Hitz e Montazeri, 1995; Henderson-Sellers et al., 1996; Martin, 1997; Briand et al., 1998; Marcus e Poshyvanyk, 2005; Újházi et al., 2010; Silva et al., 2011). Essas métricas são de dois tipos: estruturais e conceituais.

As métricas estruturais consideram a dependência estrutural entre os métodos para quantificar o valor de coesão. Esse tipo de métrica considera que uma classe é bem coesa se a maioria dos seus métodos depende um do outro ou acessa pelo menos um mesmo atributo da classe. As métricas estruturais são mais conhecidas e existem em maior quantidade (Briand et al., 1998). Contudo, estudos recentes mostram que essas métricas não refletem a coesão da mesma forma que os desenvolvedores raciocinam sobre ela (Silva et al., 2014). Para mitigar essa e outras limitações, as métricas conceituais foram propostas. Métricas conceituais procuram levar em consideração aspectos semânticos do código fonte para calcular a coesão da classe (Counsell, et al., 2005; Újházi et al., 2010; Silva et al., 2012). Esse tipo de métrica se baseia em informações dos conceitos implementados pelos métodos e nas relações de dependência conceitual entre eles. Se a maioria dos métodos implementam os mesmos conceitos, a classe é considerada bem coesa. A maioria das métricas conceituais usa técnicas de mineração de textos para determinar os conceitos implementados por cada método (Marcus e Poshyvanyk 2005; Silva et al. 2012).

Acredita-se que, quanto maior for coesão, menor pode ser o esforço para se compreender um programa (Pflegger e Atlee, 2010). A compreensão de programas consiste na

realização de atividades para se obter conhecimento geral sobre o código fonte de um sistema, as funcionalidades que ele implementa e como ele está estruturado (Bois et al., 2006). Esse processo está presente em todo o ciclo de vida de um software, principalmente na etapa de manutenção (Siegmund et al., 2013). Compreender a implementação de um sistema ou parte dele não é uma atividade trivial para os engenheiros e mantenedores, sobretudo quando não há um conhecimento prévio do software (Eisenbarth et al., 2001). Durante o processo de manutenção, essa dificuldade é ainda maior, uma vez que a documentação na maioria das vezes está desatualizada ou é inexistente (Eisenbarth et al., 2001).

1.1 PROBLEMA DE PESQUISA

Quantificar a coesão não é uma atividade trivial. A literatura acredita que esse atributo interno apresenta forte influência sob o esforço para compreender o código fonte de programas. Portanto, módulos com baixa coesão seriam, teoricamente, mais difíceis de compreender, pois possuem código relativo a diferentes responsabilidades, o que pode atrapalhar o entendimento de cada uma delas. Por outro lado, módulos com alta coesão seriam, em teoria, mais fáceis de compreender, uma vez que tratam de uma única responsabilidade.

Apesar de existir uma série de métricas de coesão e de se acreditar que coesão influencia a compreensibilidade de programas, poucos estudos foram realizados para avaliar qual é a relação entre coesão, quantificada por meio de métricas, e o esforço para se compreender o código fonte de sistemas de software. Ou seja, faltam evidências empíricas que, de fato, comprovem essa relação entre a coesão e a compreensão de programas.

Os trabalhos relacionados a esse tema basicamente se dividem em duas categorias: (i) na primeira estão os trabalhos que dedicam-se a avaliar a relação entre outras características do código fonte, como tamanho e complexidade, e compreensibilidade de programas, como em Feigenspan et al. (2011) e (ii) na segunda categoria estão as pesquisas que estudam a relação entre coesão com outros atributos externos, como a tendência a mudanças, como em Silva et al. (2012). Não encontramos nenhum trabalho que avalie explicitamente se medidas de coesão estão associadas ao esforço para se compreender programas.

1.2 OBJETIVO E QUESTÕES DE PESQUISA

O objetivo geral dessa pesquisa é avaliar, por meio de experimentos controlados, em que nível o grau de coesão de módulos de sistemas de software está relacionados ao esforço para compreender o código fonte desses módulos. Para alcançar esse objetivo realizamos três quase-experimentos que tentaram responder as seguintes questões de pesquisa:

Questão 1: A coesão de uma classe apresenta alguma influência sobre o esforço de compreensão do seu código?

De acordo com a literatura a coesão é um dos atributos de *design* de software que pode influenciar a compreensibilidade do código fonte. Acredita-se, portanto, que módulos de software bem coesos são mais fáceis de compreender. Por outro lado, módulos com baixa

coesão seriam mais difíceis de compreender. Contudo, faltam evidências empíricas dessa relação entre coesão e o esforço de compreensão. Portanto, a partir dessa questão de pesquisa avaliamos a existência dessa relação, comparando o esforço de compreensão dos participantes ao analisar classes bem coesas (ou seja, coesa estruturalmente e conceitualmente) e classes de baixa coesão tanto estrutural como conceitual.

Questão 2: Há diferença no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural?

Para quantificar os valores de coesão, a literatura apresenta uma diversidade de métricas, que se dividem em dois tipos: estruturais e conceituais. Essas métricas quantificam a coesão de formas distintas, sendo assim, uma mesma classe pode ser coesa estruturalmente, contudo não coesa conceitualmente ou vice-versa. Portanto, a partir dessa questão de pesquisa analisamos se a coesão estrutural ou conceitual apresentam influências distintas sobre o esforço de compreensão do código fonte. Realizamos essa análise comparando o esforço de compreensão de classes com alta coesão estrutural com classes de baixa coesão estrutural, porém com coesão conceitual similar. E da mesma forma para análise da coesão conceitual.

1.3 METODOLOGIA

Para alcançar o objetivo global, realizamos três experimentos controlados onde os participantes executaram tarefas de compreensão de módulos de software com diferentes graus de coesão conceitual e estrutural. Os três experimentos seguiram uma mesma configuração básica: (i) foram realizados em laboratório; (ii) tiveram como participantes estudantes de graduação e pós-graduação; (iii) os participantes analisaram e responderam questões relativas ao código fonte de cada classe envolvida no experimento; (iv) as resposta às questões demandavam a compreensão do código fonte das classes por parte dos participantes; e (v) as classes tinham diferentes valores de coesão para que pudéssemos compará-los com o esforço de compreensão.

Em cada experimento seguimos uma mesma sequência de etapas, apresentada na Figura 1.1: (i) inicialmente selecionamos as métricas de coesão (estrutural e conceitual) usadas nos experimentos para estabelecer o grau de coesão de cada classe; (ii) em seguida selecionamos as classes com base nos valores de coesão quantificados com as métricas selecionadas na etapa anterior; (iii) tendo as classes do estudo, construímos os artefatos do experimento, dentre eles, o questionário com perguntas sobre o código fonte de cada classe; (iv) após a construção dos artefatos, realizamos um estudo piloto; (v) após a execução do estudo piloto, avaliamos se a configuração do experimento estava adequada e realizamos ajustes e correções necessários; (vi) finalizado o planejamento e configuração do estudo, executamos o experimento com os participantes; (vii) por fim, analisamos os dados coletados durante a execução do experimento.

• O primeiro experimento envolveu o menor número de classes e o menor número de participantes dentre os três experimentos. Foi um experimento preliminar que teve duração de três horas. Vale ressaltar que cada experimento envolveu classes diferentes. Selecionamos as classes para cada um dos experimentos procurando minimizar outros fatores que pudessem também influenciar o esforço de compreensão. Os chamados fatores

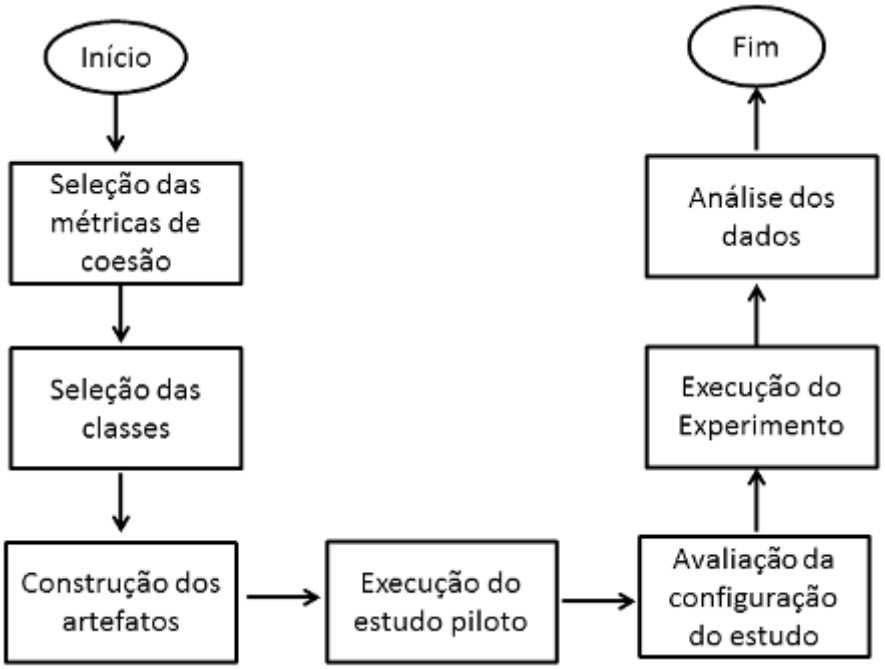


Figura 1.1 Etapas executadas nos experimentos.

de confusão. Cada experimento nos indicou novos fatores de confusão, que passamos a considerar no experimento seguinte. O primeiro experimento nos indicou, por exemplo, que acoplamento era um fator de confusão importante a ser considerado. Esse experimento foi importante também para verificarmos que as métricas usadas para quantificar o esforço de compreensão - tempo para responder as questões e número de respostas incorretas - de fato refletiam o esforço despendido pelo participante.

• O segundo experimento foi construído com base no primeiro, porém com maior número de classes e maior número de participantes. Por causa disso, tivemos que dividir sua execução em três dias, sendo três horas por dia. Além dos fatores de confusão já considerados no primeiro experimento (como tamanho e domínio), procuramos selecionar classes similares também em termos de acoplamento. No entanto, os resultados mostraram que, ainda assim, fatores de confusão (por exemplo, o domínio da classe) tiveram mais influência no esforço de compreensão do que coesão.

• O terceiro experimento envolveu apenas classes de um mesmo sistema. Nos experimentos anteriores selecionamos classes de diferentes sistemas, pois assim tínhamos mais opções. No terceiro experimento, selecionamos classes de um mesmo sistema com o objetivo de minimizar ainda mais a influência dos fatores de confusão. Classes de um mesmo sistema, e de porte pequeno como o que selecionamos, têm mais chances de serem similares em termos de organização do código, por terem sido implementadas por um grupo bem pequeno de desenvolvedores, além de serem de um mesmo domínio.

A partir dos resultados desses estudos podemos avaliar a relação entre a coesão e o esforço de compreensão de programas empiricamente e assim alcançar o objetivo final da pesquisa.

1.4 CONTRIBUIÇÕES

Essa pesquisa avaliou, por meio de experimentos controlados, se o grau de coesão, obtido por métricas estruturais e conceituais, está relacionado com o esforço de compreensão de software. Avaliamos também se os diferentes tipos de métricas – estrutural e conceitual – têm relação diferente com o esforço de compreensão. Para realizar a análise construímos três experimentos controlados, onde participantes executaram tarefas de compreensão do código fonte de classes com diferentes graus de coesão conceitual e estrutural, quantificados por meio de métricas de código fonte. Os resultados trouxeram evidência da dificuldade de se analisar isoladamente o impacto de coesão, uma vez que, outros atributos do código fonte também podem influenciar a compreensibilidade. Contudo, os resultados forneceram evidências de que a coesão não afeta seguramente o esforço de compreensão, como acredita a literatura.

1.5 PUBLICAÇÕES

Os resultados dos experimentos realizados nessa dissertação foram publicados em alguns eventos, listados a seguir:

• Batista, E.; Sant'Anna, C. Avaliação Experimental da Relação de Coesão e Acoplamento com o Esforço de Compreensão de Software. Em: *2nd Latin-American School on Software Engineering* (ELA-ES 2015). Position paper. Porto Alegre – Rio Grande do Sul, 2015, p. 78-81. Esse artigo foi um position paper que apresentava a proposta de mestrado.

• Batista, E.; Sant'Anna, C. Avaliação Experimental da Relação entre Coesão e o Esforço de Compreensão de Programas: Um Estudo Preliminar. Em: *3rd Workshop on Software Visualization, Evolution and Maintenance (VEM 2015)- 6th Brazilian Congress on Software: Theory and Practice (CBSoft 2015).* Belo Horizonte – Minas Gerais, 2015, p. 41-48. Esse artigo abordou a execução e os resultados do primeiro experimento.

• Batista, E.; Sant'Anna, C.; Silva, B. Avaliação Experimental da Relação entre Coesão e o Esforço de Compreensão de Programas. Em: *Simpósio Brasileiro de Engenharia de Software (SBES 2016).* (Submetido). Esse artigo descreve a execução e os resultados do segundo e terceiro experimentos.

1.6 ESTRUTURA DA DISSERTAÇÃO

A presente dissertação encontra-se estruturada da maneira definida a seguir:

• O capítulo 2 apresenta a revisão bibliográfica, descrevendo alguns conceitos alusivos ao tema da pesquisa e apresentando alguns trabalhos relacionados.

• O capítulo 3 descreve detalhadamente os experimentos realizados e discute seus resultados e conclusões.

• O capítulo 4 expõe as conclusões finais e os trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

Os estudos da literatura acreditam que quanto maior for a coesão, menor pode ser o esforço para se compreender um programa e vice-versa (Pflegger e Atlee, 2010). Podemos definir a coesão de um módulo como o grau pelo qual o módulo se dedica a implementar apenas uma responsabilidade do sistema (Pflegger e Atlee, 2010). Teoricamente, um módulo com alta coesão, por implementar apenas uma responsabilidade, seria mais fácil e ser compreendido. Por outro lado, módulos de baixa coesão dedicam-se a implementar uma variedade de responsabilidades e sendo assim seriam mais difícil ter seu código fonte compreendido. Portanto, para literatura, software bem projetado deve apresentar módulos bem coesos (Pflegger e Atlee, 2010).

Quantificar a coesão não é uma tarefa trivial e por isso existe uma diversidade de métricas para calcular os valores de coesão. Essas métricas são dividas em: estruturais e conceituais.

Nas seções seguintes analisamos o processo de compreensão de programas e descrevemos os tipos de medidas utilizados pela literatura para quantificar o esforço de compreensão do código fonte. Apresentamos também as diferentes métricas de coesão e discutimos as diferentes definições de coesão apresentadas pela literatura. Por fim, descrevemos alguns trabalhos relacionados à este trabalho.

2.1 COMPREENSÃO DE PROGRAMAS

Os sistemas desenvolvidos atualmente são grandes e complexos, o que torna a compreensão deles, ainda mais difícil. Como a compreensão é a tarefa mais importante no processo de manutenção de software, essa etapa do ciclo de vida do sistema tem se tornado cada vez mais custosa (Boehm, 1981 apud Rugaber, 1995). A manutenção é a fase que mais consome recursos e tempo. Esse consumo é de em média 50% a 75% do consumo total para desenvolver um sistema (Boehm, 1981 apud Rugaber, 1995). Outra característica desse consumo é que a maior parte dele é correspondente às atividades de compreensão do sistema (Rugaber, 1995).

A compreensão de programas consiste na realização de atividades para se obter conhecimento geral sobre o código fonte de um sistema, as funcionalidades que ele implementa e como ele está estruturado (Bois et al., 2006). Esse processo está presente em todo o ciclo de vida do software, principalmente na etapa de manutenção (Rugaber, 1995; Siegmund et al., 2013). Uma das formas de avaliar a qualidade de design do código fonte é tentando quantificar o esforço necessário para compreender esse código. O foco dessa pesquisa envolve medir o esforço de compreensão do código fonte de diferentes classes para avaliar se há relação desse esforço com o grau de coesão da classe. Portanto, precisamos medir o esforço de compreensão e essa não é uma tarefa trivial, em especial pelo caráter subjetivo desse atributo (Dunsmore e Roper, 2000).

Uma maneira de predizer o esforço de compreensão de programas é utilizando medidas de código fonte, como por exemplo, número de linhas de código e complexidade ciclomática. Contudo, utilizar dessas métricas isoladamente, não é suficiente para predizer o esforço de compreensão do código (Siegmund et al., 2013). Uma forma de capturar a subjetividade, presente no processo de compreensão, é utilizar não somente uma avaliação quantitativa dos valores das métricas, mas também uma avaliação qualitativa, por meio de experimentos controlados, com a participação de sujeitos humanos (Siegmund et al., 2013). Dessa forma não haverá dependência somente das medidas de código fonte. A grande dificuldade desses experimentos são os fatores de confusão que estão presentes ao se trabalhar com participantes humanos, tais como, experiência e fadiga humana, que influenciam diretamente o comportamento dos sujeitos (Siegmund et al., 2013). Sendo assim, a intenção dessa pesquisa é justamente aumentar as evidências e buscar resultados mais precisos acerca da relação entre a coesão e o esforço de compreensão. Para isso utilizamos não somente métricas de coesão, mas também trabalhamos com uma análise qualitativa.

2.1.1 Medição do Esforço de Compreensão

Sabe-se que, não existe uma medida padrão para quantificar o esforço de compreensão de programas. Segundo Dunsmore e Roper (2000), muitos estudos usam de diferentes métodos para isso. Pode-se considerar que os métodos utilizados são formas de tentar avaliar o esforço de compreensão, contudo esses métodos são incapazes de computar precisamente o esforço de compreensão do código, devido o caráter subjetivo desse atributo (Dunsmore e Roper, 2000). O estudo feito por Dunsmore e Roper (2000), buscou examinar, na literatura, os diferentes métodos utilizados para avaliar o esforço de compreensão. A partir desse exame foram selecionados quatro tipos de medidas mais utilizados, para quantificar o esforço de compreensão. Essas medidas não calculam valores precisos e absolutos para o esforço de compreensão, pois trata-se de algo subjetivo, e por isso essas medidas são denominadas indiretas. Porém elas refletem bem o esforço necessário para a compreensão do código fonte. Os quatro tipos de medidas são: De Manutenção, Dinâmicas, Estáticas e Subjetivas. Essas medidas determinam o esforço de compreensão, por meio da realização de diversas atividades. Cada medida possui um nível de confiabilidade. Para avaliar a confiabilidade de cada uma, os autores realizaram experimentos com estudantes, aplicando as medidas encontradas. Ao final do experimento, os autores con-

cluíram que, a confiabilidade e a precisão dessas medidas aumentam, quando utilizadas em conjunto. (Dunsmore e Roper, 2000).

As medidas de manutenção são aquelas que correspondem à realização de atividades de manutenção, ou seja, os participantes devem realizar atividades como: correção de bugs, remoção, adição em partes do código, entre outras. Assim, o nível de compreensão do participante é baseado na realização da tarefa de forma completa e correta e no tempo necessário para isso. Segundo o experimento realizado por Dunsmore e Roper (2000), esse tipo de medida consegue capturar de forma consistente a compreensão do participante, uma vez que é necessário entender o código para que seja possível realizar a modificação corretamente. Contudo, são exigidos alguns cuidados quando se faz o uso desse tipo de medida. Deve-se dar atenção ao tipo de atividade de manutenção requerida. Por exemplo, em uma tarefa de manutenção, se for exigido acrescentar um novo método isolado dos métodos existentes, essa atividade não auxilia na medição da compreensão do programa. Nesse contexto, para adicionar o método não significa que seja necessário compreender o resto do código (Dunsmore e Roper, 2000).

As medidas dinâmicas correspondem às tarefas que envolvam simulação mental do código. No contexto de Dunsmore e Roper (2000), a simulação mental do código significa predizer o que pode acontecer a partir do fluxo do código. Essas tarefas podem requerer que o participante complete uma parte ausente do código ou podem ainda solicitar que, depois de um pequeno período de tempo em contato com o código, o participante possa reproduzir parte do código. Outra tarefa possível é realizar um conjunto de testes e otimizar partes do código que estão sendo executadas lentamente. Essa medida, de acordo com o experimento, é considerada confiável em demonstrar o nível de compreensão do participante, pois conforme o participante tem o contato constante com o código e realiza simulações mentais do código aumenta a quantidade de informação adquirida pelo participante e isso aumenta também a compreensão da tarefa. (Dunsmore e Roper, 2000).

As medidas estáticas são obtidas com a realização de atividades nas quais os participantes devem descrever determinada funcionalidade do código, da forma que ele compreende. Também envolvem tarefas como completar o grafo de chamadas do código. Dunsmore e Roper (2000) consideram que esse tipo de medida tem baixa confiabilidade, pois, em seus estudos, encontraram poucas evidências de que as respostas dadas pelos participantes refletiam sua compreensão sobre o programa.

As medidas subjetivas correspondem às tarefas que solicitam do participante o julgamento acerca da compreensão do código, ou seja, pergunta ao participante se ele compreendeu o código e justificar. Essas medidas são altamente criticadas por sua falta de precisão. De acordo com os estudos de Dunsmore e Roper (2000), esse tipo de medida possui baixa confiabilidade e para os autores o ideal é aplicar as medidas subjetivas em conjunto com outras, para um melhor resultado.

2.2 COESÃO

Existem diversas definições para descrever coesão e consequentemente ambiguidades são encontradas. Porém, existem algumas discussões sobre qual é, de fato, a definição de coesão e quais são as métricas adequadas para quantificar esse valor. Sabe- se que,

coesão é um importante atributo de qualidade dos módulos de software e que não existe uma definição e uma medida padrão para computá-la (Fenton e Peleeger, 1997; Ferreira et al.,2011). Uma das definições de coesão mais empregada é a de Stevens et al. (1974), que considera coesão como a medida do grau em que os elementos dos módulos pertencem uns aos outros.

Seguindo o mesmo ponto de vista de Stevens et al. (1974), a coesão é tradicionalmente definida como o grau de conectividade entre os elementos de um único módulo (Briand et al.,1998). Nesse caso, se os métodos, dentro de uma classe, acessam atributos em comum, eles estão conectados entre si e assim caracterizam a classe da qual pertencem como coesa. Por outro lado, se os métodos não acessam atributos em comum, fazem da classe da qual pertencem pouco coesa (Briand et al.,1998).

Porém existem outras definições que consideram a coesão de um módulo como o grau pelo qual o módulo se dedica a implementar apenas uma responsabilidade do sistema (Pflegger e Atlee, 2010). Nesse caso, se os métodos de uma classe implementam uma mesma responsabilidade do sistema, essa classe é caracterizada como bem coesa. Entretanto, se cada método da classe trata da implementação de responsabilidades distintas, essa classe é considerada de baixa coesão.

De acordo com a literatura, o modelo ideal de um software é aquele onde existem classes de alta coesão. Desta forma, seria muito mais simples ler, desenvolver, compreender, manter e reusar o sistema, além de permitir uma menor propensão a falhas (Briand et al.,1998; Silva et al., 2012).

2.3 MEDIÇÃO DE COESÃO

Segundo Fenton e Peleeger (1997), o processo de medição é aquele onde são capturadas informações sobre atributos de uma entidade. Os atributos são propriedades das entidades. Em um software quando vamos medir uma classe, estamos descrevendo uma entidade, a classe, por meio de seus atributos, como tamanho, coesão, acoplamento e complexidade. Portanto, precisamos atribuir valores para esses atributos (Fenton e Peleeger, 1997).

Nesta pesquisa, as entidades utilizadas são classes de software e os atributos das classes serão a coesão, acoplamento e tamanho da classe. Esses são caracterizados como atributos internos, pois podem ser quantificados sem que para isso o código precise ser executado. Esses atributos de código são muito úteis para o conhecimento de uma classe, porque eles podem afetar os atributos externos de produto de software, como por exemplo, manutenibilidade, facilidade de compreensão e reusabilidade. Para a quantificação desses atributos, métricas de software são utilizadas (Fenton e Peleeger, 1997).

A literatura descreve uma diversidade de métricas para quantificar a coesão baseadas em diferentes definições (Chidamber e Kemerer, 1991; Bieman e Kang, 1995; Hitz e Montazeri, 1995; Henderson-Sellers et al., 1996; Briand et al., 1998; Marcus e Poshyvanyk, 2005; Silva et al., 2011). Por causa dessas imprecisões na definição de coesão, muitas das métricas utilizadas possuem descrições ambíguas e poucas validações empíricas (Briand et al.,1998). Dentre as métricas existentes na literatura podemos destacar algumas mais tradicionais como:*Lack of Cohesion on Methods* (LCOM) desenvolvida por Chidamber e Kemerer (1991) e as diferentes variações existentes LCOM2 (Chidamber e Kemerer, 1994),

LCOM3 (Li e Henry, 1993), LCOM4 (Hitz e Montazeri, 1995), LCOM5 (Henderson-Sellers et al., 1996). Outras métricas são: *Tight Class Cohesion* -TCC (Bieman e Kang, 1995), *Lack of Concern-based Cohesion* - LCbC (Sant'Anna, 2008), *Conceptual Lack of Cohesion On Methods* - CLCOM5 (Újházi et al., 2010) e *The Conceptual Cohesion of Classes* - C3 (Marcus e Poshyvanyk, 2005). Utilizamos algumas dessas métricas nos experimentos realizados.

Essas métricas calculam o valor de coesão de diferentes maneiras, pois tomam por base diferentes definições de coesão. Baseado nessas definições, a literatura classifica coesão em dois tipos: coesão estrutural e coesão conceitual. Do ponto de vista conceitual, a coesão pode ser definida como o grau em que os módulos do sistema estão focalizados em uma única responsabilidade do software (Pflegger e Atlee, 2010; Marcus e Poshyvanyk, 2005). Eder et al.(1994) tratam de coesão a partir desse ponto de vista, e então consideram a coesão de uma classe como o grau de uniformidade das responsabilidades designadas aos diferentes métodos daquela classe. Segundo Eder et al. (1994) e Briand et al.(1998), em uma classe com alta coesão, todos os seus métodos estão relacionados à realização de uma única função. Por outro lado, do ponto de vista estrutural, coesão representa o grau de conectividade entre os métodos da classe. Por exemplo, ao definirem a métrica LCOM, Chidamber e Kemerer (1991) consideraram que dois métodos estão conectados se eles acessam pelo menos um atributo em comum.

A maioria das métricas quantifica a coesão segundo o ponto de vista estrutural e dessa forma consideram apenas o número de métodos conectados, ou seja, métodos que acessam o mesmo conjunto de atributos ou compartilham mensagens entre si (Bieman e Kang, 1995). Contudo, essas métricas são amplamente criticadas, porque ao computar o valor de coesão, elas não analisam outras características que podem afetar a coesão, de acordo com os estudos de Counsell et al. (2005), como por exemplo, acoplamento, os comentários da classe e características semânticas do código (Counsell, et al., 2005; Újházi et al., 2010; Silva et al., 2014). Para Counsell et al. (2005), classes coesas apresentam baixo acoplamento, grande número de linhas de comentários e métodos focados em uma mesma responsabilidade. Para mitigar essas limitações, foram desenvolvidas as métricas conceituais. Além de avaliar características semânticas do código, as métricas conceituais, segundo os estudos de Silva et al. (2014), estão mais alinhadas com a forma com que os desenvolvedores raciocinam sobre coesão.

As métricas de coesão conceitual baseiam-se em informações dos conceitos implementados pelos métodos e nas relações de dependência conceitual entre eles (Silva et al., 2014). Desta forma, se a maioria dos métodos implementam os mesmos conceitos, a classe é considerada bem coesa. Portanto, essas métricas fazem uso de informações semânticas e para determinar os conceitos implementados por cada método, a maioria das métricas conceituais usam técnicas de mineração de textos (Marcus e Poshyvanyk, 2005). São poucas as métricas de coesão que fazem parte dessa categoria (Silva et al., 2011). Dentre as existentes pode-se destacar: *Lack of Concern-Based Cohesion* - LCbC de Santa'Anna (2008), *Conceptual Lack of Cohesion On Methods* - CLCOM5 desenvolvida por Újházi et al. (2010) e *The Conceptual Cohesion of Classes* - C3 desenvolvida por Marcus e Poshyvanyk (2005).

2.3.1 Métricas de Coesão Estruturais

Nas próximas subseções descrevemos algumas das métricas de coesão estruturais mais citadas e utilizadas, são elas: *Lack of Cohesion on Methods* - LCOM (Chidamber e Kemerer,1994), *Tight Class Cohesion* -TCC (Bieman e Kang, 1995) e *Lack of Cohesion on Methods 5* – LCOM5 (Henderson-Sellers et al., 1996). Nos experimentos realizados utilizamos a métrica LCOM5 para calcular o valor de coesão estrutural das classes.

2.3.1.1 Lack of Cohesion in Methods – LCOM A métrica LCOM considera a noção de dependência entre os métodos para determinar a coesão. A dependência entre os métodos corresponde ao acesso a atributos em comum, ou seja, se os métodos compartilham de um mesmo conjunto de atributos dizemos então que esses métodos são coesos entre si, caso contrário, esses métodos não possuem coesão entre si. Dentro de uma classe quanto maior for a quantidade de métodos que acessam atributos em comum, maior é a coesão da classe e vice-versa (Chidamber e Kemerer, 1994).

Para determinar o valor da LCOM é feita uma associação do número de pares de métodos não dependentes ou não coesos (P) com o número de pares de métodos dependentes ou coesos (Q). Se (mod P) $\geq$ (mod Q), o valor de LCOM corresponde a (mod P) - (mod Q), caso contrário, LCOM é zero (Chidamber e Kemerer, 1994). Além disso, LCOM é uma métrica que calcula a falta de coesão entre os métodos e, portanto, quanto maior for o valor de LCOM, maior será a falta de coesão e menos coesa será a classe.

Por exemplo, a figura abaixo representa uma classe que possui três métodos (representado pelos números 1, 2 e 3) e quatro atributos. Dentre os métodos existentes, temos um par de métodos coesos, ou seja, acessam atributo em comum (o método 2 e o método 3, ambos acessam o primeiro atributo), nesse caso Q = 1. Por outro lado, temos dois pares de métodos não coesos (os pares de métodos 1 e 2/ 1 e 3), ou seja, que não acessam atributos em comum e assim P = 2. Substituindo os valores de Q e P na formula de LCOM ((mod P) - (mod Q)), temos que (mod 2) - (mod 1) e, portanto, o valor de LCOM para essa classe é 1.

Figura 2.1 Classe exemplificando cálculo de LCOM.

Nas seções seguintes, apresentamos outras métricas de coesão estruturais, porém, não exemplificamos cada uma delas, pois seguem uma abordagem similar de calcular coesão,

com base na dependência entre os métodos.

2.3.1.2 Tight Class Cohesion – TCC A métrica TCC toma por base os pares de métodos diretamente conectados. Em uma classe podem existir métodos diretamente conectados ou indiretamente conectados. Nesse contexto os métodos diretamente conectados são aqueles que são coesos entre si, por exemplo, M1 e M2 são considerados coesos se acessam um mesmo conjunto de atributos. Métodos indiretamente conectados são aqueles que não são coesos entre si, por exemplo, M1 é diretamente conectado a M2 e M2 é diretamente conectado a M3, portanto M1 e M3 são indiretamente conectados (Bieman e Kang, 1995).

Diante disso, TCC calcula a taxa dos métodos diretamente conectados dentro da classe. Sendo NDC, o número de conexões diretas na classe e NP, o número máximo de conexões diretas e indiretas entre os métodos da classe, TCC corresponde a NDC/NP (Bieman e Kang, 1995). Quanto maior o valor de TCC maior é a taxa de métodos diretamente conectados e, portanto, maior é a coesão da classe. Por exemplo, se uma classe possui sete conexões diretas (NDC = 7) e sendo 10 o número máximo de conexões diretas e indiretas (NP = 10), o valor de TCC para essa classe será 7/10, ou seja, TCC = 0,7. Isso significa que 70% dos métodos dessa classe são diretamente conectados, o que faz dessa classe bem coesa.

2.3.1.3 Lack of Cohesion in Methods 5 – LCOM5 A LCOM5 é a versão mais atual da métrica LCOM. Através de estudos mais detalhados da LCOM, foram notadas algumas possíveis deficiências nesta métrica ao computar os valores de coesão. Essas deficiências faziam com que os valores de LCOM não representassem, de fato, o grau de coesão das classes (Henderson-Sellers et al., 1996; Briand et al., 1998). Dentre essas carências podemos destacar: (i) a falta de clareza em descrever quais são as características que determinam um par de métodos dependentes entre si e (ii) ausência de valores normalizados para a métrica. Essas falhas foram tratadas definindo o que constitui um acesso entre os métodos conectados entre si (Hitz e Montazeri, 1995) e também realizando avaliações, do ponto de vista matemático, o que permitiu a normalização dos valores da métrica (valores entre [0,1]) (Henderson-Sellers et al., 1996). Essas mudanças na métrica LCOM levaram a desenvolvimento de novas métricas, que representam novas versões da LCOM, como: LCOM1 (Chidamber e Kemerer, 1991), LCOM2 (Chidamber e Kemerer, 1994), LCOM3 (Li e Henry, 1993), LCOM4 (Hitz e Montazeri, 1995) e LCOM5 (Henderson-Sellers et al., 1996). E entre elas está a LCOM5 (Henderson-Sellers et al., 1996).

A LCOM5 representa uma extensão da LCOM já normalizada, ou seja, seus valores podem variar entre [0,1]. O valor de LCOM5 é baseado na expressão:

$$\text{LCOM5} = [\tfrac{1}{a}\textstyle\sum_{j=1}^{a}\left(\mu(A_j)\right) - m] \;/\; (1-m)$$

Onde “a” representa o número de atributos acessados pelos métodos, “m” é o número

de métodos e "μ(Aj)" é o número de métodos que acessam cada atributo (Henderson-Sellers et al., 1996).

2.3.2 Métricas de Coesão Conceituais

Nas próximas subseções descrevemos algumas das métricas de coesão conceituais mais citadas na literatura, são elas:*Lack of Concern-based Cohesion* - LCbC (Sant'Anna, 2008), *Conceptual Lack of Cohesion On Methods* - CLCOM5 (Újházi et al., 2010) e *The Conceptual Cohesion of Classes* - C3 (Marcus e Poshyvanyk, 2005). Nos experimentos utilizamos a métrica LCbC para calcular o valor de coesão conceitual.

2.3.2.1 Lack of Concern-based Cohesion - LCbC O termo "*concern*" tem sido traduzido para "interesse" no contexto de engenharia de software. Portanto, quando utilizado na presente pesquisa os termos "interesse" e "*concern*" são tratados como sinônimos. Muitas definições são usadas para descrever ou definir um interesse. Um interesse pode ser definido como algo que os projetistas de software desejam considerar como uma unidade conceitual. Segundo essa definição, interesses podem ser, por exemplo, funcionalidades, requisitos não funcionais e operacionalização de requisitos, e por isso podem impactar a implementação de um sistema. Portanto, um interesse representa uma importante característica ou algum aspecto do sistema que se deseja tratar de forma modular. Alguns exemplos de interesse são: persistência, segurança, regras de negócio, distribuição e armazenamento, que são propriedades presente na maior parte dos sistemas e que podem estar distribuídas por diversas classes. (Robillard e Murphy, 2007; Eaddy et al., 2008).

A métrica Lack of Concern-Based Cohesion (LCbC) também conhecida como LCC, é uma métrica de coesão dirigida por interesses (Silva et al., 2011). Essa métrica conta o número de interesses presentes em cada módulo (classes, por exemplo) do código. Quanto mais interesses um módulo possuir, menos coeso ele é (Sant'Anna, 2008) (Silva et al., 2012). Porém antes de computar o valor dessa métrica é necessário mapear os interesses no código fonte. O mapeamento de interesses no código é uma tarefa na qual se determina quais fragmentos do código são responsáveis pela execução de cada interesse do sistema (Silva et al., 2012). Essa tarefa pode ser realizada de forma totalmente manual, que consome muito tempo, ou com apoio de ferramentas de localização de *features* (Robillard e Weigand-Warr, 2005) ou mineração de aspectos (Hannemann e Kiczales, 2001 apud Sant'Anna, 2008).

Portanto, segundo essa métrica se uma classe contribui para implementação de diversos interesses, essa classe é pouco coesa, pois tem diversas responsabilidades. Isso a torna, potencialmente, mais complexa e de difícil compreensão (Sant'Anna, 2008).

Por exemplo, a figura abaixo representa uma classe que, após o mapeamento dos interesses no código, foi detectada com três interesses distintos, representados pelas cores. Assim, as linhas de código em preto implementam o interesse regra de negócio, por exemplo. Em azul, podemos ter o interesse persistência e em vermelho o interesse tratamento de exceções. Após mapeado os interesses conseguimos identificar o número de *concerns* que a classe implementa, isso significa que conseguimos estabelecer o valor de LCbC. Assim a classe apresentada na Figura 2.2 implementa três interesses distintos e portanto,

Figura 2.2 Classe exemplificando cálculo de LCbC.

o valor de LCbC é três.

2.3.2.2 Conceptual Lack of Cohesion On Methods - CLCOM5 É uma métrica baseada na LCOM5, pois possui um mecanismo de contagem inspirado na utilização de dependência entre métodos, feita pelas métricas de coesão estruturais (Újházi et al., 2010). Contudo, a forma de determinar se os métodos são dependentes entre si é feita de forma conceitual. Assim, de acordo com CLCOM5, a conexão entre os métodos é baseada nos conceitos que esses métodos implementam.

Nessa métrica o código fonte é analisado e é gerado um conjunto de documentos textuais, onde cada documento representa a implementação de um método. Com esses documentos é construída uma matriz de termos versus documentos, que captura a dispersão e a co-ocorrência de termos nos métodos. O propósito é capturar a dependência entre os métodos através do compartilhamento de informações entre eles (Újházi et al., 2010). Esse mecanismo de extração de informações semânticas do código fonte é baseado no *Latent Semantic Indexing* (LSI), um mecanismo bem conhecido de recuperação de informações (Marcus e Poshyvanyk, 2005). A ideia é encontrar métodos que compartilham os mesmo termos, nesse caso eles são coesos entre si, e, portanto, quanto mais pares métodos coesos existirem em uma classe, mais coesa essa classe é.

2.3.2.3 The Conceptual Cohesion of Classes - C3 A métrica *The Conceptual Cohesion of Classes* (C3) é semelhante à métrica anterior (CLCOM5). Ela transforma o código em um conjunto de documentos textuais, somente com comentários e identificadores extraídos dos métodos e, também constrói uma matriz para analisar a dependência entre os métodos (Marcus e Poshyvanyk, 2005). A principal diferença é que na métrica anterior (CLCOM5) é usado um mecanismo de contagem dos pares de métodos, inspirado nas métricas estruturais (Újházi et al., 2010).

Para extração de informações semânticas do código, ambas as métricas, C3 e CLCOM5, utilizam o mesmo mecanismo, ou seja, o baseado no *Latent Semantic Indexing* (LSI) (Marcus e Poshyvanyk, 2005).

Para determinar a coesão de uma classe é feita uma análise do grau de coesão dos métodos pertencentes a essa classe, a partir disso, é possível definir se a classe representa

um único interesse. Assim, se há uma forte coesão entre os métodos de uma classe o valor de C3 tende a 1, caso contrário, o valor de C3 tende a 0 (Marcus e Poshyvanyk, 2005).

2.4 TRABALHOS RELACIONADOS

A maior parte dos trabalhos relacionados ao nosso, pertence, de forma geral, a duas categorias: (i) na primeira estão os trabalhos que dedicam-se a avaliar a relação entre outras características do código fonte e facilidade de compreensão de programas, como Feigenspan et al. (2011), Bois et al. (2006) e Nishizono et al. (2011), e (ii) na segunda categoria estão as pesquisas que estudam a relação entre coesão com outros atributos externos, como a tendência a mudanças ou a tendência a defeitos, como Silva et al. (2012). Não encontramos nenhum trabalho que avalie explicitamente se medidas de coesão estão associadas ao esforço para compreender programas.

Feigenspan et al. (2011), por exemplo, analisam se e em que grau medidas de complexidade e tamanho (número de linhas código), estão relacionadas a facilidade de compreensão do programa. Para isso utilizam duas versões do sistema MobileMedia: uma versão escrita em Java (orientada a objetos) e outra em AspectJ (orientada à aspectos). Essas versões apresentam diferentes valores para as métricas avaliadas. Cada versão do MobileMedia é avaliada por um grupo de participantes. Cada grupo realizou seis tipos de tarefas baseadas na versão apresentada. No final do experimento foi avaliado o tempo e as respostas corretas de cada grupo, para então saber qual grupo obteve o melhor desempenho. Contudo, os resultados não demonstraram diferenças significativas na compreensibilidade dos grupos, relatando que as versões apresentavam níveis de compreensão equivalentes. Assim, os resultados concluíram que não há correlação entre essas características do código fonte e a facilidade de compreensão do programa. Os autores acreditam que possa existir uma relação entre coesão ou acoplamento com a compreensibilidade de programa e por isso recomendam uma investigação dessa relação.

Outro estudo que pertence a primeira categoria é o de Bois et al. (2006). Nesse estudo o objetivo dos autores foi verificar, por meio de experimentos, se refatorações para eliminar *God Class* (Fowler et al., 1999; Lanza et al., 2007) afetam a compreensibilidade do código fonte. As *God Class*, segundo os autores, são classes grandes que acumulam uma série de responsabilidades, tornando a compreensão do código cada vez mais complexa. Portanto, a decomposição ou refatoração dessas classes em classes menores, com uma única responsabilidade, pode facilitar o processo de compreensão. Para avaliar essa hipótese, os autores realizaram um experimento. Durante o experimento uma *God Class* foi refatorada de cinco maneiras diferentes. Depois o nível de compreensão do código resultante de cada refatoração foi avaliado. Essa análise foi feita por cinco grupos. Cada grupo realizou tarefas de manutenção no código de umas das refatorações. Por meio do tempo e da precisão da tarefa realizada, foi determinado à facilidade de compreensão do código da refatoração. Os resultados do estudo mostraram que as diferentes decomposições de *God Class* apresentam diferentes níveis de compreensibilidade. Sendo assim, os autores concluíram que existem decomposições que são melhores que outras, no que diz respeito a uma maior facilidade de compreensão do código fonte.

Outro estudo ainda dessa primeira categoria é o de Nishizono et al. (2011). Nishizono

et al. (2011) relacionam algumas características do código, diferente de *God Class*, que podem influenciar no esforço de compreensão. Nessa pesquisa, os autores tentam predizer, por meio de experimento controlado, estratégias utilizadas pelos desenvolvedores para compreender o código fonte. Com base nessas estratégias, os autores identificam métricas que poderiam ser usadas para quantificar o esforço de compreensão do código em tarefas de modificação e evolução. No experimento realizado, os participantes receberam a aplicação e o código para que pudessem tentar compreender através da leitura do código e do uso da aplicação. Em seguida, receberam o documento com as tarefas de modificação. Assim, os participantes tentaram entender as especificações de mudanças, compreender a parte do código correspondente à mudança e decidiram se as mudanças estavam ou não corretas. As variáveis analisadas ao final do experimento foram o tempo que o participante necessitou para compreender o código e a estratégia de compreensão utilizada, a qual foi descrita pelo participante. Concluindo o estudo Nishizono et al. (2011) examinam as estratégias utilizadas e por meio delas, determinaram métricas que podem está relacionadas com o esforço de compreensão. As métricas que demonstraram ter uma forte correlação com o esforço para compreender as tarefas, foram: número de variáveis referenciadas, acoplamento eferente, acoplamento aferente, número de linhas de código (LOC) e quantidade de código clonado. A partir das estratégias listadas no estudo, os autores perceberam que para compreender o código, os participantes tentam fazer uma varredura pelas partes consideradas mais importantes no código, e essa varredura é diretamente influenciada pelo número de linhas das classes. Além da varredura, os participantes buscam entender as linhas de código que se repetem em diversas classes, pois uma vez compreendidas, não precisam ser lidas novamente em outras partes do código. As métricas de acoplamento são justificadas a partir da utilização de estratégias de compreensão baseadas no fluxo de dependências de controle e de dados entre as classes.

Na segunda categoria, podemos destacar o estudo realizado por Silva et al (2012). O estudo analisa a relação existente entre a métrica de coesão conceitual LCbC e a propensão de mudanças da classe, por meio da métrica *Change Count*. De acordo com Silva et al. (2012), *Change Count* conta a quantidade de mudanças sofridas por cada classes ao longo da evolução do sistema. No estudo empírico, utilizado para validação das hipóteses, os autores selecionam seis sistemas de softwares de diferentes tamanhos e domínios. A partir desses sistemas foram calculadas algumas métricas de coesão estrutural. Além disso, foram computadas LCbC e *Change Count*. Para analisar a relação entre as métricas de coesão e *Change Count* os autores utilizam o coeficiente de correlação de *Spearman*. O coeficiente evidencia que apenas duas métricas de coesão, LCbC e LCOM2 estão fortemente relacionadas com a propensão de mudanças. Como o foco do estudo é na métrica de coesão conceitual, os resultados revelam a existência de uma correlação positiva entre a métrica LCbC e a tendência de mudança das classes. Significando que, quanto mais interesses uma classe implementar mais provável é a ocorrência de mudanças nessa classe, ou seja, quanto maior o valor de LCbC, maior será a possibilidade de acontecer mudanças.

Diante dos trabalhos realizados, que envolvem compreensibilidade e as demais características do código fonte, ainda há lacunas a serem preenchidas, principalmente quando se trata da relação entre a facilidade de compreensão e atributos internos do código fonte.

Com o objetivo de realizar uma nova análise, focada em avaliar a relação entre coesão e o esforço de compreensão de programas, essa pesquisa realizou três experimentos controlados, descritos no próximo capítulo.

Capítulo

3

EXPERIMENTOS

Essa pesquisa destina-se a avaliar em que nível o grau de coesão de módulos (classes) do sistema está relacionados com o esforço para compreender o código fonte desses módulos. Para alcançar esse objetivo realizamos três quase-experimentos. O quase-experimento é uma pesquisa empírica similar aos experimentos controlados, contudo, não há uma distribuição aleatória dos sujeitos, mas sim, uma distribuição baseada nas características dos sujeitos ou objetos (Wohlin et al., 2012). Consideramos os estudos como quase-experimentos principalmente porque a seleção das classes e os participantes foi feita por conveniência. No decorrer desse capítulo o termo "experimento" também será utilizado para referenciar os quase-experimentos. Os três experimentos tiveram como objetivo responder as seguintes questões de pesquisa:

Questão 1: A coesão de uma classe apresenta alguma influência sobre o esforço de compreensão do seu código?

Questão 2: Há diferença no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural?

Os estudos seguiram um mesmo padrão de configuração: No laboratório, os participantes realizaram atividades que demandaram a compreensão do código fonte de diferentes classes. A realização dessas atividades permitiu que medíssemos o esforço desprendido por cada participante para compreender cada classe. As classes tinham diferentes valores de coesão para que pudéssemos compará-los com o esforço de compreensão.

O primeiro experimento foi um estudo preliminar, onde avaliamos se seria possível quantificar o esforço de compreensão e compará-lo com os valores das métricas de coesão. Por isso selecionamos quatro classes de diferentes sistemas e com diferentes níveis de coesão e comparamos o esforço de compreensão exigido por essas classes. O segundo estudo foi construído com base no primeiro, porém com maior número de participantes e de classes. Utilizamos 12 classes de diferentes sistemas, assim tentamos aumentar a precisão dos resultados. Por fim, no terceiro estudo utilizamos seis classes que pertenciam a um mesmo sistema, o que não ocorreu nos dois estudos anteriores. O fato de utilizarmos classes de um mesmo sistema foi para tentar minimizar fatores que poderiam influenciar

o esforço de compreensão, como: diferença de domínio do código, a forma como as classes foram estruturadas e organizadas e a qualidade da nomenclatura dos identificadores. Ao final da execução e avaliação dos resultados dos estudos, concluímos que a coesão não influenciou fortemente o esforço de compreensão, como a literatura acredita.

As seções seguintes, descrevem cada um dos experimentos detalhadamente, mostrando suas particularidades.

3.1 EXPERIMENTO I

O primeiro experimento foi um estudo preliminar contendo poucas classes e um reduzido número de participantes. No laboratório, dezoito participantes realizaram atividades que demandaram a compreensão do código fonte de quatro classes, com distintos valores de coesão. A seguir mais detalhes do *design* do estudo são descritos.

3.1.1 Design do Experimento

Métricas: Utilizamos a métrica de coesão estrutural *Lack of Cohesion in Methods 5* (LCOM5) (Henderson- Sellers et al., 1996) e a métrica de coesão conceitual *Lack of Concern-Based Cohesion* (LCbC) (Silva et al., 2012).

Selecionamos LCOM5 por se tratar da versão mais recente da tradicional métrica de coesão LCOM (Chidamber and Kemerer, 1994). Para quantificar coesão, LCOM5 considera a dependência entre métodos por meio do acesso a atributos em comum. Quanto maior a quantidade de métodos que acessam atributos em comum, maior é a coesão da classe e vice-versa. Os valores de LCOM5 variam de zero a um. Quanto menor o valor de LCOM5 mais coesa é a classe. Nos estudos, calculamos os valores de LCOM5 por meio da ferramenta *Metrics*[1].

Selecionamos a métrica conceitual LCbC por já ter sido usada em outros estudos que a comparam com métricas estruturais (Silva et al., 2012; Silva et al., 2014). LCbC conta o número de interesses presentes em cada classe (Silva et al., 2012). Quanto mais interesses uma classe implementa, menos coesa ela é, e vice-versa. Para quantificar essa métrica é preciso determinar quais interesses cada método de uma classe implementa (Silva et al., 2012). Isso pode ser feito manualmente ou usando alguma técnica de mineração de texto, como acontece com outras métricas conceituais (Liu et al., 2009). Devido o pequeno número de classes, foi possível determinar manualmente os valores de LCbC para as classes do estudo.

Medição do esforço de compreensão: Para quantificar o esforço necessário para compreender cada classe usamos duas métricas: (i) o tempo utilizado pelos participantes para responder um conjunto de perguntas sobre cada uma das classes e (ii) o número de perguntas com respostas erradas. Ao final do experimento calculamos para cada classe o tempo médio de análise e o número médio de erros. O tempo médio foi calculado somando-se o tempo que os participantes levaram para responder as perguntas de cada classe, dividido pelo número de participantes. Por outro lado, o número médio de erros por classe é a soma do número de erros da classe dividido pelo número de participantes.

[1]http://metrics.soucerforge.net/

As perguntas apresentadas nos questionários exigiam uma simulação mental do código fonte, como por exemplo, "Se o método m receber como entrada os valores x e y, qual será seu retorno?" Essa estratégia está alinhada com o que a literatura recomenda para se medir esforço de compreensão (Bois et al., 2006) (Feigenspan et al., 2011). Segundo Dunsmore and Roper (2000) esse tipo de questão representa atividades dinâmicas, que refletem melhor a compreensão do participante acerca do código fonte. Para cada classe existiram quatro perguntas de diferentes estilos (Apêndice C): (i) uma questão de completar o código, preenchendo determinada linha em branco, (ii) duas questões sobre características específicas dos métodos existentes na classe e (iii) uma questão que solicitava ao participante encontrar uma determinada linha do código responsável por uma função específica.

Além disso, depois de responder as questões sobre todas as classes, pedimos aos participantes para informar que classe eles acharam mais difícil de entender. Pedimos também para eles dizerem quais os motivos que levaram eles a considerar tal classe como a mais difícil de entender. Dessa forma, pudemos ter também uma resposta qualitativa para complementar as medições de tempo e número de erros.

Estudo Piloto: Antes da execução do experimento realizamos um estudo piloto, para avaliar a configuração do experimento. Esse piloto contou com a participação de quatro alunos da pós-graduação em Ciência da Computação, da Universidade Federal da Bahia. Dentre esses participantes, tivemos programadores que consideramos experientes (trabalhavam com programação por três anos ou mais) e pouco experientes. Por meio do estudo piloto: (i) verificamos o tempo médio de execução do experimento, (ii) corrigimos alguns erros ortográficos encontrados nos questionários e (iii) verificamos alguns esclarecimentos que deveriam ser comunicados aos participantes antes da execução do estudo.

Artefatos: Um conjunto de documentos é usado nos experimentos e é compartilhados nos apêndices para possibilitar futuras replicações do estudos. Para esse primeiro estudo, construímos os seguintes artefatos:

- Termo de consentimento (Apêndice A): Apresentava uma descrição da pesquisa, os seus responsáveis e convidada o aluno a participar de forma voluntária. O participante no início do experimento lê e assina o termo, consentindo a participação.

- Questionário de caracterização (Apêndice B): Esse questionário é aplicado após o termo de consentimento. Esse questionário busca construir o perfil dos participantes, portanto, solicita informações quanto à experiência do participante com programação orientada a objeto.

- Questionários de atividades (Apêndice C): Esses questionários apresentam as perguntas alusivas às classes analisadas. Para cada classe existe um questionário. Também são nesses questionários que os participantes informam o tempo inicial e final da análise da classe.

- Questionário de *feedback* (Apêndice F): Nesse questionário os participantes expressam suas opiniões a respeitos do nível de dificuldade das classes, e os fatores que determinam esse nível de dificuldade, além de avaliar as questões propostas no experimento. O objetivo desse questionário é obter um retorno por parte dos participantes sobre a execução do estudo e as principais dificuldades encontradas no experimento. Além disso,

por meio da opinião dos participantes sobre o nível de dificuldade das classes podemos comparar os dados quantitativos com a opinião dos participantes, representando assim dados qualitativos para pesquisa.

Seleção de Classes: Representa a etapa mais crítica do estudo, pois ao selecionar as classes teríamos que minimizar ao máximo a influência de fatores que pudessem afetar o esforço de compreensão (fatores de confusão). Selecionamos, portanto, classes com as seguintes características: (i) tamanhos semelhantes em termos do número de linhas de código, (ii) domínio simples e acessível a todos os participantes, (iii) nomenclatura de identificadores com qualidade semelhante e (iv) ausência de comentários.

Além de considerar os fatores de confusão ao selecionar as classes, consideramos os valores das métricas. Selecionamos classes com valores de LCOM5 e LCbC que permitissem a comparação entre diferentes valores de coesão estrutural e conceitual (Tabela 3.1). Portanto, comparamos classe que apresentava alto nível de coesão tanto conceitual como estrutural, com classe que possuía baixo nível de coesão conceitual e estrutural. Também comparamos classe que tinha alto nível de coesão conceitual com classe que possuía baixo nível de coesão conceitual e fizemos o mesmo para coesão estrutural. É importante lembrar que as classificações “alta” e “baixa” são relativas à comparação dos valores entre si e não podem ser generalizadas. Por exemplo, não podemos considerar que o valor 0,6 de LCOM5 para uma classe signifique genericamente baixa coesão. Porém, nos experimentos, consideramos que esse valor de LCOM5 significa baixa coesão se comparado ao valor zero. Ou seja ao compararmos duas classes, uma com valor 0,6 e a outra com o valor zero para LCOM5, consideramos a primeira como ”baixa coesão”e a segunda como ”alta coesão”apenas nesse contexto de comparação entre as duas. O mesmo se aplica para a métrica LCbC. Não significa que, de forma geral, um classe com LCbC igual a 4 tenha ”baixa coesão”. Porém, podemos considerá-lo como ”baixa coesão”se comparado ao valor de LCbC igual a 1. Assim as classes selecionadas tinham as seguintes características: (i) uma classe com alta coesão estrutural e alta coesão conceitual (Contatos), (ii) uma classe com baixa coesão estrutural e baixa coesão conceitual (Locadora2), (iii) uma classe com alta coesão estrutural e baixa coesão conceitual (BibliotecaUI2) e (iv) uma classe com baixa coesão estrutural e alta coesão conceitual (Person2).

Tabela 3.1 Valores das métricas das classes selecionadas

Métrica	BibliotecaUI2	Person2	Locadora2	Contatos
LCOM5	0	0,93	0,6	0
LCbC	4	1	4	1
Número de Linhas (LOC)	274	254	240	200

As classes foram retiradas de sistemas *open source* armazenados no repositório de código, o *GitHub*[2] . A classe BibliotecaUI2 foi obtida de um sistema de gerenciamento de biblioteca. Essa classe constrói a estrutura do menu do sistema e a interface desse menu. A classe Person2 pertence a um sistema construído com proposito de pesquisa, denominado FamilyTree, que permite criar uma árvore genealógica de uma família. A classe Person2 é uma classe pertencente ao pacote modelo do sistema e representa um

[2]https://github.com/

membro da árvore, contendo todos seus atributos e características. A classe Locadora2 foi obtida de um sistema de gerenciamento de locadora de filmes. Essa classe é responsável pela construção de um menu inicial, além disso, possui o controle de cadastro e exclusão de filmes e clientes, assim como o controle sob as buscas de filmes e clientes cadastrados. Por fim, a classe Contatos pertence a um sistema de gerenciamento de agenda telefônica. Essa classe controla o cadastramento de pessoas na agenda, permitindo também excluir ou alterar os contatos cadastrados.

Para cada classe realizamos um mapeamento dos concerns que foram levados em consideração para calcular o valor da métrica LCbC. Cada classe apresentou os seguintes interesses demonstrados na Tabela 3.2:

Tabela 3.2 Interesses encontrados nas classes selecionadas.

Classes	Interesses
BibliotecaUI2	Livro Usuário Empréstimo Tratamento de exceções
Person2	Pessoa
Locadora2	Cliente Filmes Entrada de dados Menu
Contatos	Contato

As classes foram analisadas na seguinte ordem: BibliotecaUI2, Person2, Locadora2 e Contatos. Essa ordem foi estabelecida com base no tamanho da classe, em termos de número de linhas. Assim as classes foram aplicadas da maior para menor. Com essa ordem poderíamos minimizar a fadiga e o cansaço dos participantes, já que as últimas classes eram menores que as iniciais.

Participantes: A amostra foi composta por 18 participantes, desses 14 eram alunos de graduação e quatro eram alunos de pós-graduação em Ciência da Computação da Universidade Federal da Bahia. A amostra foi selecionada por conveniência e foi representada por turmas as quais os pesquisadores tinham acesso.

Sequência da execução das atividades: Esse estudo foi executado em três horas. Na execução, as tarefas foram realizadas na seguinte sequência: No laboratório as classes já estavam disponibilizadas para os participantes por meio da interface do editor de texto *WinEdit*. Inicialmente foi comunicado aos participantes o que eles deveriam fazer e foram feitos alguns esclarecimentos sobre o estudo e sobre os questionários que os participantes iriam receber. Em seguida, os participantes receberam o termo de consentimento (Apêndice A) e anexado a esse documento estava o questionário de caracterização

(Apêndice B). Após o preenchimento desses artefatos o participante poderia solicitar o questionário de atividade (Apêndice C) da primeira classe. Quando o participante terminava de analisar e responder o questionário da primeira classe ele então, solicitava o questionário da próxima classe e assim sucessivamente até completar a análise de todas as classes. Desta forma poderíamos controlar a ordem na qual as classes eram analisadas. Finalizada a análise das classes, antes de sair, o participante recebia o questionário de *feedback* (Apêndice F). Depois de preenchê-lo, o participante encerrava sua participação no experimento.

3.1.2 Ameaças à Validade

Abaixo apresentamos possíveis ameaças à validade do experimento e as ações realizadas para minimizá-las.

Validade Interna: A fadiga dos participantes ao analisar o código fonte é uma possível ameaça à validade interna. Para minimizar a influência da fadiga, configuramos o estudo de forma que sua execução não ultrapassasse uma hora de duração. Participantes com diferentes níveis de experiência em programação poderiam ser outra ameaça. Verificamos a experiência de cada participante, por meio de um questionário de caracterização, e levamos em conta as informações ao analisar os resultados, comparando o desempenho dos participantes experientes e dos participantes pouco experientes. Também nos preocupamos com o fato da falta de similaridade entre as classes poder influenciar os resultados. Tentamos minimizar essa ameaça levando em consideração alguns fatores de confusão (Seção 3.1) ao selecionar as classes. Por fim, a ordem na qual as classes foram analisadas pode ser mais uma ameaça ao experimento. O fato de todos os participantes analisarem as classes na mesma ordem, pode levá-los a: (i) analisar as últimas classes com menos interesse devido à fadiga ou (ii) analisar o código das últimos classes com mais facilidade devido ao fato de terem aprendido o domínio do sistema do qual as classes fazem parte. Não vemos nenhum desses dos problemas como forte ameaça à validade desse primeiro experimento, pois (i) o tempo do experimento foi curto, o que evita a fadiga e (ii) as classes faziam parte de sistemas e domínios distintos.

Validade Externa: O estudo possui a limitação de não poder ser generalizado devido à pequena quantidade de classes e à pequena amostra de participantes, composta somente por estudantes. Outra ameaça está relacionada ao fato dos participantes terem usado um editor de texto (*WinEdit*) mais limitado que os ambientes de programação comumente usados em ambientes profissionais. Fizemos essa opção de propósito, pois o *WinEdit* não fornece notificações de erros no código nem sugere soluções. Como existia uma pergunta que pedia para o participante completar uma linha em branco do código, não queríamos que ele tivesse ajuda do editor.

Validade de Construto: A forma como medimos o esforço de compreensão poderia ser outra ameaça, pois se trata de algo subjetivo. Para isso, utilizamos procedimentos indicados pela literatura e já utilizados em outros estudos (Dunsmore and Roper, 2000)(Feigenspan et al., 2011). A clareza das perguntas apresentadas aos participantes poderia representar uma ameaça à medição do esforço de compreensão, uma vez que diferentes participantes poderiam interpretar uma mesma pergunta de maneiras distintas.

Por isso, avaliamos e ajustamos as perguntas com base no estudo piloto. Os diferentes níveis de dificuldades das perguntas pode ser outra ameaça. Por mais que tentamos minimizar esse fator ao criar as questões, elas são questões distintas e assim possuem níveis distintos de dificuldade. Por meio do estudo piloto também foi possível avaliar se haviam diferenças significativas com respeito o nível de dificuldade das questões.

3.1.3 Resultados e Discussões

A Figura 3.1 mostra o tempo gasto pelos participantes para analisar e responder as perguntas relativas a cada classe. E a Figura 3.2 mostra o tempo médio exigido por cada classe. A Figura 3.1 apresenta uma simples legenda apresentando o nível de coesão de cada classe, onde o sinal '+' representa alta coesão e o sinal '-' baixa coesão.

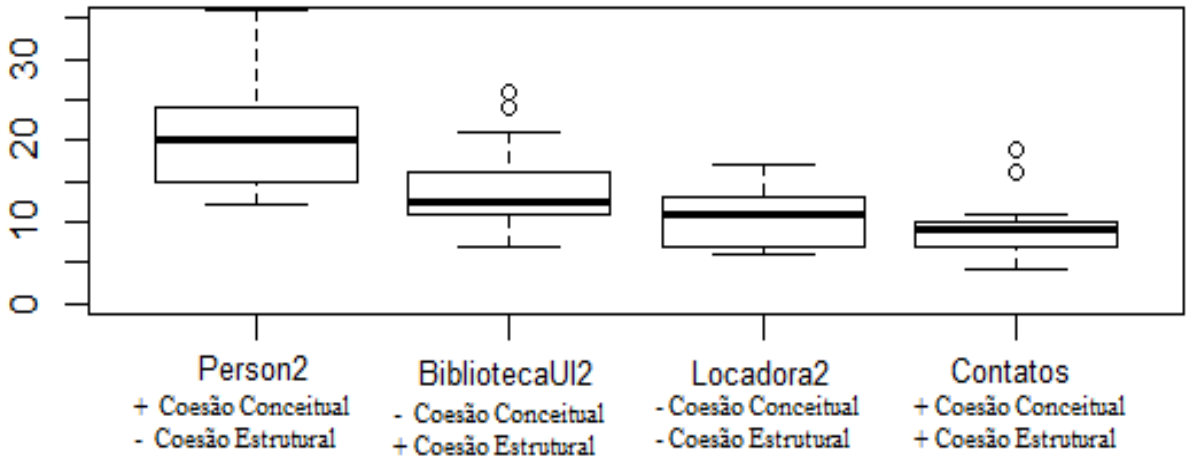


Figura 3.1 Tempo de análise de cada classe, em minutos.

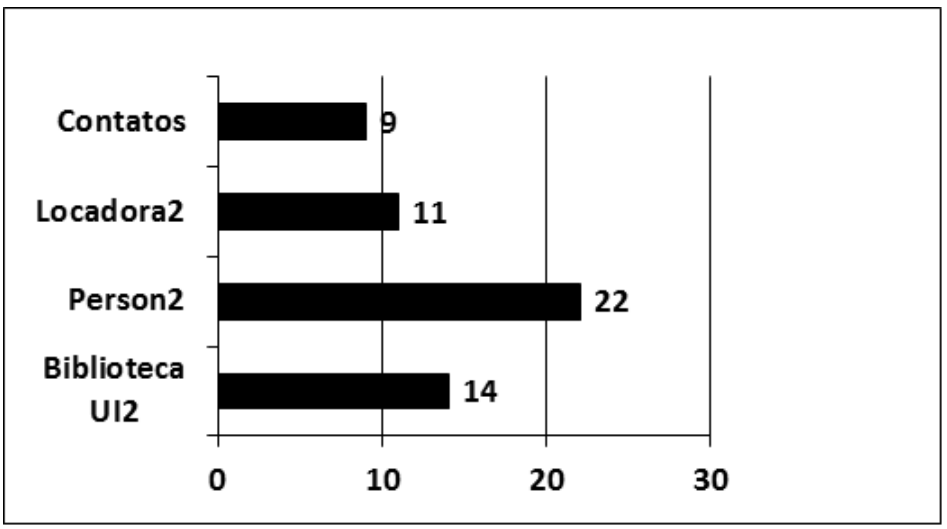


Figura 3.2 Tempo médio de análise de cada classe, em minutos.

Podemos observar que a classe Person2 foi a que demandou mais tempo de análise: em média 22 minutos e de acordo com o valor da mediana, metade dos participantes levou até 20 minutos para analisar a classe. Essa classe possui baixa coesão estrutural (LCOM5 = 0,93) e alta coesão conceitual (LCbC = 1). É importante destacar que, apesar de termos selecionados classes com tamanhos similares, não foi possível encontrar classes com exatamente o mesmo número de linhas de código. Porém, Person2 não é a maior classe. Ela tem LOC = 254, enquanto BibliotecaUI2 tem LOC = 274 e é a maior classe.

BibliotecaUI2 demandou o segundo maior tempo para ser analisada: 14 minutos em média, sendo que metade dos participantes levou até 12,50 minutos para analisar a classe. Mesmo assim esse tempo não é muito maior que os tempos de análise requeridos pelas demais classes: Locadora2 (11 minutos em média) e Contatos (9 minutos em média). Em termos de coesão, BibliotecaUI2 apresenta características inversas a Person2: alta coesão estrutural (LCOM5 = 0) e baixa coesão conceitual (LCbC = 4).

A classe Contatos é a mais bem coesa tanto conceitualmente (LCbC = 1) quanto estruturalmente (LCOM5 = 0). Por outro lado, a classe Locadora2 apresenta baixa coesão conceitual (LCbC = 4) e baixa coesão estrutural (LCOM5 = 0,6). Apesar do contraste em relação aos valores de coesão, as duas classes requereram os dois menores tempo de análise: em média 9 minutos para Contatos e 11 minutos para Locadora2.

A Figura 3.3 mostra o número de erros cometidos pelos participantes durante a análise de cada classe, lembrando que na legenda o sinal '+' representa alta coesão e o sinal '-' baixa coesão. E a Figura 3.4 mostra o número médio de erros de cada classe.

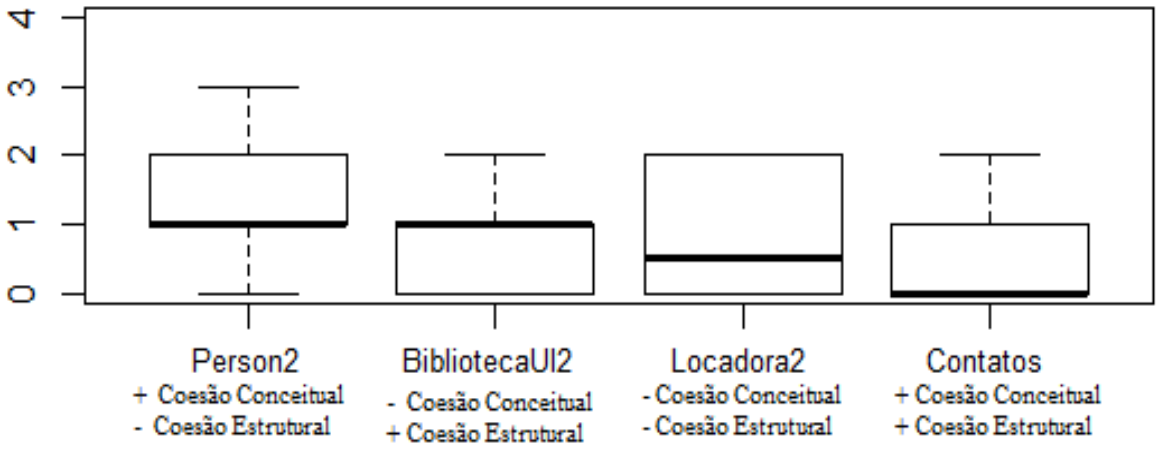


Figura 3.3 Número de erros de cada classe.

Os resultados observados na Figura 3.3 são bem semelhantes aos resultados do tempo de análise (Figura 3.1). Considerando tanto o tempo médio quanto o número médio de erros, a classe Person2 foi a classe que demandou maior esforço para ser compreendida, enquanto a classe Contatos demandou o menor esforço. Isso aumenta a confiança do estudo, pois demonstra que não houve casos em que o participante cometeu menos erros em uma classe porque gastou mais tempo analisando-a.

Considerando a experiência dos participantes declarada no questionário de caracte-

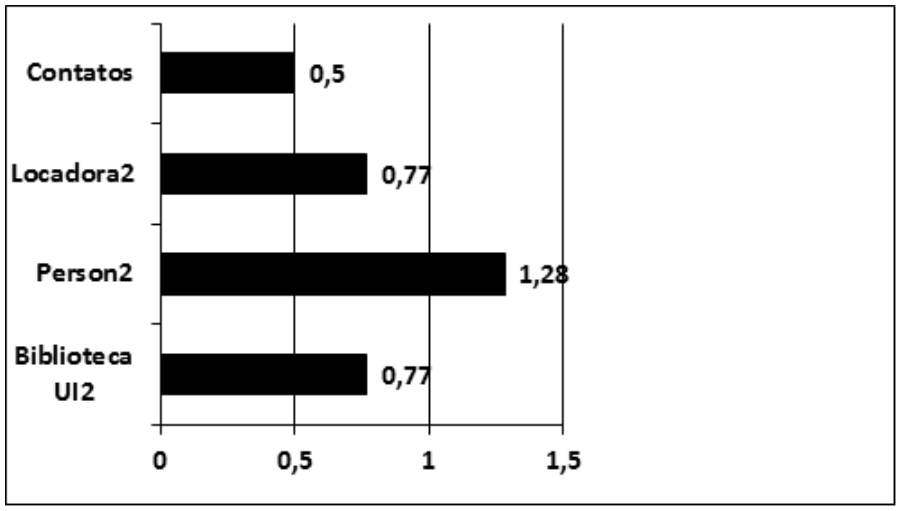


Figura 3.4 Número médio de erros de cada classe.

rização, observamos que os sujeitos considerados experientes tiveram melhor desempenho. Para esse grupo o maior número de erros obtidos foi quatro e o maior tempo de análise foi de 68 minutos. Para o grupo considerado pouco experiente o maior número de erros foi oito e o maior tempo gasto foi de 81 minutos.

Ao final do experimento, um questionário de *feedback* pedia ao participante para indicar a classe que ele achou mais difícil de compreender. As respostas a essa questão confirmaram os dados quantitativos: todos elegeram a classe Person2 como a classe mais difícil de compreender. O principal fator apontado pelos participantes como algo que dificultou o entendimento da classe foi o fato da classe analisada usar outras classes. Para eles, quanto mais dependente de outras classes, mais difícil foi para entender uma classe. É importante destacar aqui que os participantes não tiveram acesso ao código fonte das outras classes dos sistemas. Essa informação nos levou a coletar para cada uma das classes o valor de uma métrica de acoplamento. O objetivo foi saber até que ponto, as classes analisadas dependiam de outras classes do sistema. Usamos a bem conhecida métrica *Coupling Between Objects* – CBO, definida por Chidamber and Kemerer (Chidamber and Kemerer, 1994).

Escolhemos a métrica CBO por se tratar de uma métrica tradicional de acoplamento que é bastante citada e utilizada pela literatura. Essa métrica conta o número de classes com as quais uma determinada classe A está acoplada. Ela considera que duas classes estão acopladas quando métodos declarados, em uma classe, usam métodos ou variáveis de instâncias definidos por outras classes. No estudo, calculamos os valores de CBO por meio da ferramenta *JHawk*[3]. Essa ferramenta calcula o valor de CBO considerando somente o acoplamento *fan-out*, ou seja, calculamos somente o número de classe que uma determinada classe A usa e não as classes que usam A. Os valores dessa métrica podem ser visualizados na Tabela 3.3.

A classe Person2 tem o maior grau de acoplamento (CBO = 5) dentre as quatro classes

[3]http://www.virtualmachinery.com/jhdownload.htm

Tabela 3.3: Valor do acoplamento das classes.

Métrica	BibliotecaUI2	Person2	Locadora2	Contatos
CBO	3	5	3	1

utilizadas: ela depende de cinco outras classes do sistema. Provavelmente tenha sido esse o motivo de sua compreensão ter sido considerada a mais difícil, tanto pelos resultados quantitativos, quanto pelos resultados qualitativos (onde os participantes descrevem suas opiniões).

Diante dos resultados, discutimos as questões de pesquisa da seguinte forma:

Questão 1: A coesão de uma classe apresenta alguma influência sobre o esforço de compreensão do seu código? Os resultados do estudo não foram suficientes para responder a essa questão. A classe que se mostrou mais difícil de entender (Person2) é a que tem coesão estrutural mais baixa. No entanto, parece que a dificuldade para compreender seu código deveu-se mais ao fato dela ser a classe que mais depende de outras classes do sistema. A classe mais fácil de entender (Contatos) foi a que apresentou coesão, tanto conceitual quanto estrutural, mais alta. No entanto, também é a classe com o menor grau de acoplamento.

Questão 2: Há diferenças no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural? Os resultados do estudo também não foram suficientes para responder a essa questão. Se tomarmos com base as classes Person2 e Contatos, poderíamos dizer que coesão estrutural tem relação com compreensão, pois como observamos respondendo a questão anterior, essas classes apresentam esforço de compreensão divergentes e níveis de coesão estrutural também divergentes. Porém, temos de novo o fator do alto acoplamento da classe Person2 influenciando. Por outro lado, as classes BibliotecaUI2 e Locadora2 tem: (i) aproximadamente o mesmo nível de esforço de compreensão, (ii) o mesmo grau de coesão conceitual (valores iguais de LCbC) e (iii) o mesmo valor de acoplamento. Como elas apresentam diferentes graus de coesão estrutural (LCOM5 = 0 para BibliotecaUI2 e LCOM5 = 0.6 para Locadora2) e demandam esforço similar para serem compreendidas, podemos dizer que, nesse caso, não houve relação entre coesão estrutural e esforço de compreensão.

3.1.4 Conclusão

O estudo preliminar desenvolvido teve objetivo de analisar em que nível o grau de coesão de classes está relacionado ao esforço para compreender o código fonte dessas classes. Os resultados obtidos não permitiram responder as questões de pesquisa, principalmente por não termos anulado um fator de confusão importante: o grau de acoplamento das classes. **Um dos pontos positivos do estudo foi o fato de termos tido sucesso em medir o esforço de compreensão, uma vez que os resultados quantitativos foram compatíveis com a opinião dos participantes**.

Para os estudos seguintes utilizamos um número maior de participantes e classes. Também tomamos mais cuidado para minimizar os fatores de confusão, principalmente os valores de acoplamento das classes. Os demais estudos foram focados em continuar avaliando a influência da coesão sobre o esforço de compreensão do código fonte, para

assim conseguir resultados e conclusões mais concretos e precisos.

3.2 EXPERIMENTO II

Nesse segundo experimento os participantes realizaram atividades que demandaram a compreensão do código fonte de 12 classes, no período de 12 horas, divididas em quatro horas por dia. As classes tinham diferentes valores de coesão, representadas por três cenários distintos, para que pudéssemos compará-los com o esforço de compreensão. Cada cenário foi aplicado em um dia.

No primeiro cenário, comparamos classes de alta coesão estrutural e conceitual com classes de baixa coesão estrutural e conceitual. O objetivo desse cenário era analisar o esforço de compreensão de classes com alta coesão total, ou seja, tanto conceitual e estrutural e de classes com baixa coesão total, desta forma poderíamos avaliar se a coesão, de fato, influência no esforço de compreensão. No segundo cenário, as classes possuíam valores de coesão estrutural semelhantes, porém valores de coesão conceitual distintos. Nesse cenário o objetivo era analisar se a coesão conceitual apresentava alguma influência sobre o esforço de compreensão do código. No terceiro cenário, as classes possuíam valores de coesão conceitual semelhantes, porém valores de coesão estrutural diferentes. Nesse terceiro cenário o objetivo era avaliar se a coesão estrutural influenciava o esforço de compreensão do código.

3.2.1 Design do Experimento

Métricas: Assim como no primeiro experimento utilizamos a métrica de coesão estrutural *Lack of Cohesion in Methods 5* (LCOM5) (Henderson- Sellers et al., 1996) e a métrica de coesão conceitual *Lack of Concern-Based Cohesion* (LCbC) (Silva et al. 2012) (Seção 3.1). No experimento anterior não consideramos o fator acoplamento durante a seleção das classes, contudo, nesse segundo estudo selecionamos classes com o mesmo nível de acoplamento em cada cenário. Portanto, também usamos a métrica de acoplamento *Coupling Between Objects* – CBO (Chidamber and Kemerer, 1994) na seleção das classes.

A CBO é uma métrica tradicional de acoplamento, bem utilizada na literatura. Essa métrica determina o nível de acoplamento de uma classe e considera que duas classes estão acopladas quando métodos declarados, em uma classe, usam métodos ou variáveis de instâncias definidos por outras classes. Usamos a ferramenta *JHawk* para calcular os valores de CBO. Essa ferramenta, como mencionado na seção 3.1.2, calcula o valor de CBO considerando somente o acoplamento *fan-out*, ou seja, calculamos somente o número de classe que uma determinada classe A usa e não as classes que usam A.

Medição do esforço de compreensão: Para quantificar o esforço de compreensão usamos as mesmas métricas do primeiro estudo: (i) o tempo utilizado pelos participantes para responder um conjunto de perguntas sobre cada uma das classes e (ii) o número de perguntas com respostas erradas. Por fim, calculamos para cada classe o tempo médio de análise e o número médio de erros (Seção 3.1).

As perguntas apresentadas nos questionários, semelhante ao estudo anterior, também caracterizavam atividades dinâmicas, as quais, segundo Dunsmore and Roper (2000) refle-

tem melhor a compreensão do participante acerca do código. Para cada classe fizemos três perguntas de diferentes estilos (Apêndice D). Diminuímos para três o número de questões, pois o número de classes era maior que o do primeiro experimento. Se não fizéssemos isso, esse experimento iria ficar muito longo. As questões foram: (i) uma questão de completar o código, preenchendo determinada linha em branco, (ii) uma questão sobre características específicas dos métodos existentes na classe e (iii) uma questão de implementação de um método, responsável por uma nova função.

Estudo Piloto: Antes da execução do experimento novamente realizamos um estudo piloto, para avaliar a configuração do estudo. Esse piloto contou com a participação de três alunos da pós-graduação em Ciência da Computação, da Universidade Federal da Bahia. Desses participantes, um foi considerado como experiente (trabalhava com programação por três anos ou mais) e dois pouco experientes. Por meio do estudo piloto: (i) verificamos o tempo médio de execução do experimento, (ii) corrigimos alguns erros ortográficos encontrados nos questionários e (iii) verificamos alguns esclarecimentos que deveriam ser comunicados aos participantes antes da execução do estudo.

Artefatos: Semelhante ao primeiro estudo, esse segundo experimento apresentou um conjunto de artefatos (Seção 3.1): (i) o termo de consentimento (Apêndice A), (ii) questionário de caracterização (Apêndice B), (iii) questionário de atividades (Apêndice D) e (iv) questionário de *feedback* (Apêndice G).

O questionário de *feedback*, assim como no estudo anterior, teve o objetivo de obter um retorno por parte dos participantes sobre o experimento realizado e também realizar uma análise qualitativa do estudo. Nesse experimento, para adquirir essa análise qualitativa, os participantes deveriam colocar no questionário de *feedback* as classes ordenadas da mais fácil para a mais difícil e citar os critérios utilizados para fazer a ordenação. Para cada cenário do experimento foi aplicado um questionário de *feedback*.

Seleção de Classes: Essa foi mais uma vez a etapa mais crítica do estudo, pois ao selecionar as classes teríamos que minimizar ao máximo a influência de fatores que também pudessem influenciar no esforço de compreensão (fatores de confusão), descritos na seção 3.1. Além desses fatores, outro aspecto importante para a seleção das classes foram os valores das métricas LCOM5 e LCbC. A Tabela 3.4 apresenta os valores de coesão das classes selecionadas, para os três cenários. Para permitir a comparação entre diferentes valores de coesão estrutural e conceitual, escolhemos:

• No primeiro cenário, para avaliar a influência da coesão sobre o esforço de compreensão, selecionamos duas classes com valor alto de coesão estrutural e conceitual (RepositorioCliente e ContatoDAO) e duas classes com valor baixo de coesão estrutural e conceitual (Locadora e PassagemControle).

• No segundo cenário, para avaliar a influência da coesão conceitual sobre o esforço de compreensão, todas as classes possuíam valores semelhantes de coesão estrutural, porém duas classes possuíam valor alto de coesão conceitual (RepositorioFuncionario e Compromisso) e duas classes possuíam valor baixo de coesão conceitual (Jogo e Sala).

• No terceiro cenário, para avaliar a influência da coesão estrutural sobre o esforço de compreensão, todas as classes possuíam valores semelhantes de coesão conceitual, porém duas classes possuíam valor alto de coesão estrutural (ControleConvenio e Droga) e duas classes possuíam valor baixo de coesão estrutural (BombaDeInsulina e RepositorioEnde-

reco).

Mais uma vez, assim como explicado para o primeiro experimento, é importante lembrar que as classificações “alta” e “baixa” são relativas à comparação dos valores entre si e não podem ser generalizadas. Outro fator importante ao selecionar as classes foi o nível de acoplamento. Selecionamos as classes de forma que as classes de um mesmo cenário apresentassem o mesmo nível de acoplamento, calculado por meio da métrica CBO (Tabela 3.4).

Tabela 3.4: Valores das métricas das classes selecionadas.

Classes	LCOM5	LCbC	CBO	LOC
Primeiro Cenário				
RepositorioCliente	0,2	2	3	173
ContatoDAO	0,21	2	3	121
Locadora	0,6	4	3	213
PassagemControle	0,66	4	3	160
Segundo Cenário				
RepositorioFuncionario	0	2	4	147
Compromisso	0,1	2	4	135
Jogo	0	5	4	150
Sala	0	5	4	123
Terceiro Cenário				
ControleConvenio	0,1	2	8	143
Droga	0	2	8	113
BombaDeInsulina	1	2	8	115
RepositorioEndereco	1	2	8	155

Essas classes também foram retiradas de sistemas *open source* armazenados no repositório de código, denominado *GitHub*.

No primeiro cenário, a classe RepositorioCliente foi retirada de um sistema de gerenciamento de uma loja de aluguel de roupas. Essa classe faz conexão com o banco de dados permitindo inserir, excluir, atualizar e buscar os clientes no banco de dados. A classe ContatoDAO foi obtida de um sistema de gerenciamento de agenda e também faz conexão com o banco de dados permitindo listar, excluir, atualizar e adicionar as informações do contato no banco de dados. A classe Locadora pertence a um sistema de gerenciamento de uma locadora de filmes e constrói a estrutura do menu do sistema e a interface desse menu. Por fim, a classe PassagemControle pertence a um sistema de simulação de vendas de passagens aéreas. Ela realiza o controle sobre a venda das passagens, verificando todos os dados da passagem, validando os dados do voo, os dados do cliente e por fim, valida ou não a compra.

No segundo cenário, a classe RepositorioFuncionario foi retirada de um sistema de gerenciamento de uma loja de aluguel de roupas. Essa classe faz conexão com o banco de dados permitindo inserir, atualizar, excluir e listar os dados do funcionário no banco de dados. A classe Compromisso pertence a um sistema de gerenciamento de uma agenda e trata de controlar todos os compromissos e de determinar as regras de negócio do sistema para que cada compromisso seja avisado no período determinado. A classe Jogo foi obtida

de um sistema de simulação do jogo da forca e realiza o controle do jogo, verificando as palavras, os erros e acertos de cada participante e determinando quem vence o jogo. Já a classe Sala, pertence a um sistema de gerenciamento de controle climático e trata de verificar os sensores externos e internos de uma sala para analisar se a temperatura da sala está aumentando ou diminuindo, determinando assim, a quantidade de pessoas dentro da sala por meio da temperatura.

No terceiro cenário, a classe ControleConvenio foi retirada de um sistema de gerenciamento de uma agência de plano de saúde. Ela realiza conexão com o banco de dados e faz todo o controle de busca, atualização, adição e exclusão dos dados do convênio no banco de dados. A classe Droga pertence a um sistema de gerenciamento de um hospital. Essa classe também realiza conexão com o banco de dados e é responsável por carregar todos os dados sobre a droga no banco de dados. A classe BombaDeInsulina foi obtida de um sistema de simulação de um sensor que controla o nível de açúcar no sangue. Ela faz a leitura dos níveis de açúcar no sangue e determina se a pessoa precisa ou não receber insulina e determina a dosagem de insulina que a pessoa precisa. Por fim, a classe RepositorioEndereco pertence a um sistema de gerenciamento de uma loja de aluguel de roupas e essa classe também realiza uma conexão com o banco de dados e controla a busca, a adição, exclusão e atualização dos dados sobre o endereço nas tabelas do banco.

Assim como no primeiro estudo, para cada classe realizamos um mapeamento dos concerns para então calcular o valor da métrica LCbC. Cada classe apresentou os seguintes interesses demonstrados na Tabela 3.5:

Como mencionado, cada cenário do experimento foi aplicado em dias distintos, porém seguidos. Em cada dia as classes analisadas foram ordenadas de acordo com o tamanho da classe, em termos do número de linhas, e de acordo com as características de coesão da classe. Assim, ordenamos as classes da maior para menor e intercalamos as classes, de forma que as classes com características semelhantes em termo de coesão, não fossem analisadas juntas. Por exemplo, no segundo cenário as classes foram analisadas na seguinte ordem: (i) Jogo, por ser a maior (LOC = 150), (ii) RepositorioFuncionario, por ser a segunda maior classe (LOC = 147) e por apresentar características com a relação a coesão diferente da classe Jogo, (iii) Sala, apesar de não ser a terceira maior classe (LOC = 123) possui coesão diferente da classe RepositorioFuncionario e por fim, (iv) Compromisso com LOC = 135 e com coesão diferente da classe Sala.

Participantes: A amostra foi composta por alunos de uma turma de graduação (28 alunos) e alunos de uma turma de pós-graduação (24 alunos) em Ciência da Computação da Universidade Federal da Bahia. O tamanho da amostra variou para os três cenários, já que cada cenário foi aplicado em dias distintos: 37 participantes no primeiro cenário, 40 no segundo cenário e 35 no terceiro cenário. Contudo, após uma limpeza dos dados, caracterizada pela retirada dos valores considerados como *outliers* (valores muito mais alto que a média ou valores muito abaixo da média), o tamanho da amostra variou para: 35 participantes no primeiro cenário (18 alunos de pós-graduação e 17 de graduação), 35 participantes no segundo cenário (11 alunos de pós-graduação e 24 de graduação) e 34 participantes no terceiro cenário (19 alunos de pós-graduação e 24 de graduação). Essa limpeza dos dados é necessária para que quando os dados passarem por uma análise estatística, os resultados sejam mais precisos, uma vez que os valores da amostra repre-

Tabela 3.5: Interesses encontrados nas classes selecionadas.

Classes	Interesses
RepositorioCliente	Cliente Tratamento de exceções
ContatoDAO	Contato Tratamento de exceções
Locadora	Cliente Filmes Entrada de dados Menu
PassagemControle	Passagem Vôo Validação de CPF Data
RepositorioFuncionario	Funcionário Tratamento de exceções
Compromisso	Compromisso Tratamento de exceções
Jogo	Entrada de dados Tratamento de exceções Jogo Palavra Get
Sala	Sala Sensores Temperatura Climatizador Ambiente externo
ControleConvenio	Convênio Tratamento de exceções
Droga	Droga Tratamento de exceção
BombaDeInsulina	Controle de dose de insulina Tratamento de exceção
RepossitorioEndereco	Endereço

sentam um conjunto de dados válidos, ou seja, dados que pertencem à curva normal e representem de fato a população (Wohlin et al., 2012). Nesse caso, não teremos valores muito mais alto que a média nem valores muito abaixo da média. Quanto à experiência, a amostra estava bem dividida com cerca de 50% dos participantes experientes e 50% pouco experientes. Mais uma vez, a amostra foi selecionada por conveniência e foi representada por turmas as quais os pesquisadores tinham acesso.

Sequência da execução das atividades: A execução das tarefas do segundo experimento seguiu a mesma ordem da sequência feita no primeiro experimento: (i) esclarecemos, aos participantes, o que eles deveriam fazer e explicamos sobre o estudo e sobre os questionários que eles iriam receber, (ii) entregamos o termo de consentimento (Apêndice A) e anexado a esse documento estava o questionário de caracterização (Apêndice B), (iii) após o preenchimento desses artefatos o participante poderia solicitar o questionário de atividade (Apêndice D) da primeira classe. Finalizada a análise da primeira classe, o participante solicitava o questionário da próxima classe e assim sucessivamente até completar a análise de todas as classes daquele cenário. Os participantes analisaram os códigos das classes por meio da interface do editor de texto Gedit e por fim, (iv) antes de sair, o participante recebia o questionário de *feedback* (Apêndice G), depois de preenchido o participante encerrava o experimento. Essa sequência de atividades foi realizada no primeiro dia de aplicação do experimento. Nos dias subsequentes repetimos os esclarecimentos sobre o estudo e em seguida o participante já recebia o questionário de atividades das classes e por fim o questionário de *feedback*. Para os participantes ausentes no primeiro dia, era entregue o termo de consentimento e o questionário de caracterização.

3.2.2 Ameaças à Validade

Abaixo apresentamos possíveis ameaças à validade do experimento e as ações realizadas para minimizá-las.

Validade Interna: A fadiga dos participantes ao analisar o código fonte é, mais uma vez, uma possível ameaça à validade interna. Para minimizar a influência da fadiga, realizamos o estudo em três dias de forma que o tempo de execução para cada dia fosse inferior à uma hora e meia. Também realizamos um estudo piloto para avaliar o tempo de execução do estudo e o cansaço dos participantes. Participantes com diferentes níveis de experiência em programação poderiam ser outra ameaça. Verificamos a experiência de cada participante, por meio de um questionário de caracterização, e levamos em conta as informações ao analisar os resultados, comparando o desempenho dos participantes experientes e dos participantes pouco experientes e notamos, nesse segundo estudo, que a experiência não apresentou influência determinante sobre o esforço de compreensão. Também nos preocupamos com o fato da falta de similaridade entre as classes poder influenciar os resultados. Tentamos minimizar essa ameaça levando em consideração alguns fatores de confusão ao selecionar as classes, inclusive o fator acoplamento, que não foi considerado no primeiro estudo. Por fim, a ordem na qual as classes foram analisadas pode ser mais uma ameaça ao experimento. O fato de todos os participantes analisarem as classes na mesma ordem, pode levá-los a: (i) analisar as últimas classes com menos interesse devido à fadiga ou (ii) analisar o código das últimos classes com mais facilidade

devido ao fato de terem aprendido o domínio do sistema do qual as classes fazem parte. Não vemos nenhum desses dos problemas como forte ameaça à validade desse segundo experimento, pois (i) dividimos o experimento em três dias justamente para minimizar a questão da fadiga e (ii) as classes faziam parte de sistemas e domínios distintos.

Validade Externa: O estudo possui a limitação de não poder ser generalizado devido à pequena quantidade de classes e à pequena amostra de participantes, composta somente por estudantes. Outra ameaça está relacionada ao fato dos participantes terem usado um editor de texto (*Gedit*) mais limitado que os ambientes de programação comumente usados em ambientes profissionais. Fizemos essa opção de propósito, pois o *Gedit* não fornece notificações de erros no código nem sugere soluções. Como existia uma atividade que pedia para o participante completar uma linha em branco do código, não queríamos que ele tivesse ajuda do editor.

Validade de Construto: Da mesma forma como no primeiro estudo, a forma como medimos o esforço de compreensão poderia ser outra ameaça, pois se trata de algo subjetivo. Para isso, utilizamos procedimentos indicados pela literatura e já utilizados em outros estudos (Dunsmore and Roper, 2000)(Feigenspan et al., 2011). A clareza das perguntas apresentadas aos participantes poderia representar uma ameaça ao esforço de compreensão, uma vez que diferentes participantes poderiam interpretar uma mesma pergunta de maneiras distintas. Por isso, como no primeiro estudo, avaliamos e ajustamos as perguntas com base no estudo piloto. Também como o primeiro estudo, os níveis de dificuldades das perguntas pode ser outra ameaça, pois as questões são distintas e assim possuem níveis distintos de dificuldade. Por meio do estudo piloto também foi possível avaliar se haviam diferenças significativas com respeito o nível de dificuldade das questões.

3.2.3 Resultados e Discussões

As duas variáveis dependentes desse experimento são: (i) o tempo médio de análise de cada classe, calculado por meio da média aritmética dos tempos que os participantes levaram para responder o questionário de atividades de cada classe, e (ii) número médio de erros para cada classe, calculado por meio da média aritmética do número de questões do questionário de atividades que os participantes responderam de forma errada as classes. Resolvemos usar essas duas métricas para representar esforço de compreensão, pois participantes poderiam responder rapidamente as questões, mas de forma errada. Vale a pena lembrar, que o objetivo era saber se essas variáveis dependentes variaram de acordo com a variação do valor das medidas de coesão das classes.

Em todos os três cenários, utilizamos o teste Shapiro-Wilks (Wohlin, et al., 2012) para verificar se os valores obtidos para as variáveis dependentes seguiam a distribuição normal. Nos três cenários, o teste mostrou que o tempo de análise de cada classe não seguia a distribuição normal, enquanto o número de erros seguia. Dessa forma, nos três cenários, para verificar se existiam diferenças estatisticamente significante entre o tempo médio de análise de cada classe usamos o teste de Kruskal Wallis (Wohlin, et al., 2012). Por outro lado, usamos o teste o T-test (Wohlin, et al., 2012) para avaliar a variação entre os valores do número médio de erros de cada classe. Tanto o T-test quanto o teste de Kruskal Wallis são utilizados para analisar a variância entre médias de duas amostras independentes. O

primeiro serve para amostras que seguem distribuição normal e segundo para amostras que não seguem. No nosso caso, tivemos amostras independentes, pois tivemos valores para cada classe. Calculamos todos os testes estatísticos por meio da ferramenta R[4].

Primeiro Cenário: Aplicamos os testes de variância para cada classe, comparando-a com as demais classes do mesmo cenário. Os resultados podem ser observados na Tabela 3.6 e Tabela 3.7. Os números nas tabelas representam os valores de p-value, que são os resultados da análise da variabilidade entre a classe da linha e a classe da coluna. Se o p-value for menor que 0,05 significa que há diferenças significativas entre as médias obtidas para classes. Por outro lado, se o p-value for maior que 0,05 não há diferenças significativas.

Tabela 3.6: Teste Kruskal Wallis para o tempo médio de análise das classes do primeiro cenário.

Variância do Tempo	PassagemControle	RepositorioCliente	ContatoDAO
Locadora	0,2589	0,2741	0,1572
PassagemControle		0,2738	0,329
RepositorioCliente			0,4811

Tabela 3.7: T- Test para o número médio de erros das classes do primeiro cenário.

Variância do Número de Erros	PassagemControle	RepositorioCliente	ContatoDAO
Locadora	0,3825	0,2215	3,365e-05
PassagemControle		0,03914	1,585e-06
RepositorioCliente			0,001861

A Tabela 3.6 mostra que não houve diferenças significativas nos valores de tempo médio das classes desse primeiro cenário. Para todo par de classes o p-value foi maior que 0,05. Por outro lado, na Tabela 3.7, observamos diferenças significativas (p-value $\leq$ 0,05) na comparação entre as classes PassagemControle e RepositorioCliente. Além disso, obtivemos p-value $\leq$ 0,05 na comparação da classe ContatoDAO com todas as demais classes. Isso mostra que, de fato, o número médio de erros dessa classe foi significativamente diferente que o das outras classes.

É importante relembrar aqui, que as figuras que apresentam os boxplots possuem uma legenda indicando o nível de coesão de cada classe, onde o sinal '+' representa alta coesão e o sinal '-' baixa coesão. A Figura 3.5 mostra a variação dos valores do tempo que os participantes gastaram para analisar e responder as perguntas relativas a cada classe. E a Figura 3.6 a média de tempo exigida por cada classe. Podemos observar que os valores de cada classe realmente variaram em faixas próximas.

[4]https://www.r-project.org/

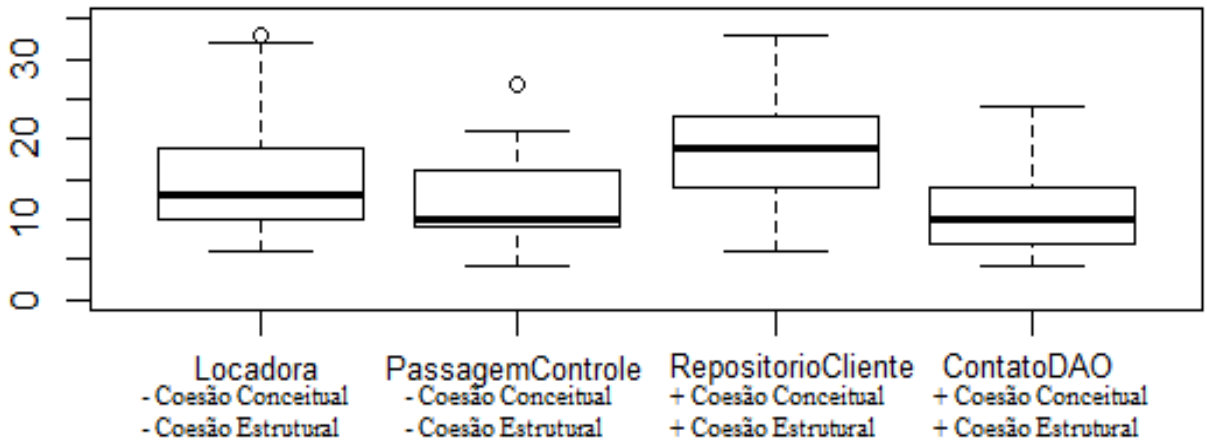


Figura 3.5 Tempo de análise de cada classe do primeiro cenário, em minutos.

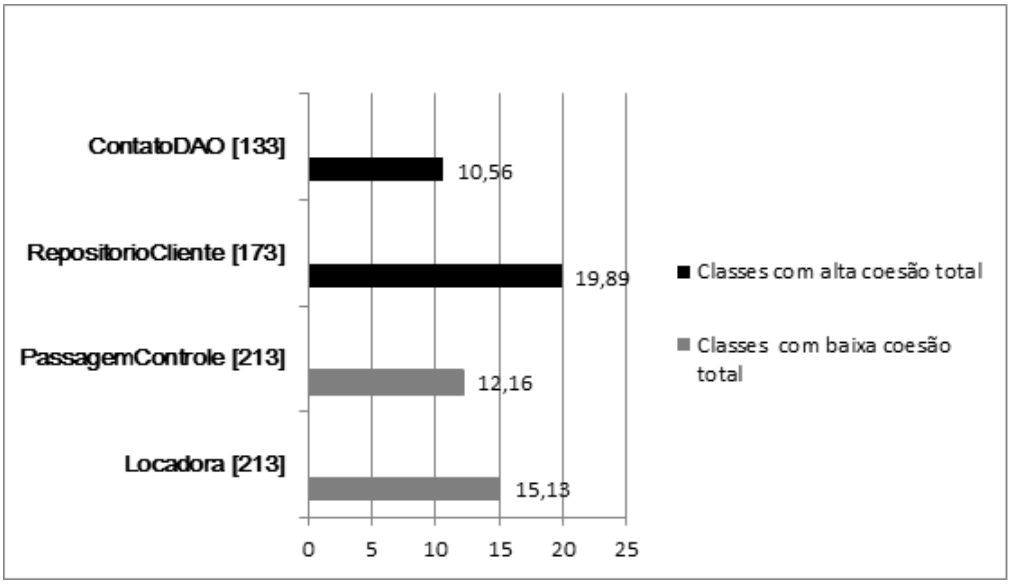


Figura 3.6 Tempo médio de análise de cada classe do primeiro cenário, em minutos.

Podemos observar no primeiro cenário que a classe RepositorioCliente foi a que demandou mais tempo para análise, em média 19,89 minutos e metade dos participantes levou até 19 minutos para analisar a classe, conforme a mediana apresentada no boxplot. Essa classe se caracteriza por apresentar alta coesão estrutural e conceitual (LCOM5 = 0,2 e LCbC = 2).

As classes PassagemControle e Locadora demandaram um tempo de análise bem menor se comparado ao tempo de RepositorioCliente: em média 12,16 minutos para PassagemControle (sendo que metade dos participantes levou até 10 minutos para analisar a classe) e em média 15,13 minutos para Locadora (sendo que metade dos participantes levou até 13 minutos para analisar a classe). PassagemControle (LCOM5 = 0,66 e LCbC = 4) e Locadora (LCOM5 = 0,6 e LCbC = 4) representam classes com baixa coesão conceitual e estrutural.

A classe ContatoDAO é caracterizada por apresentar alta coesão estrutural e conceitual (LCOM5 = 0,21 e LCbC = 2), porém, ao contrário da classe RepositórioCliente, ContatoDAO possuiu o menor tempo de análise, em média 10,56 minutos.

O tamanho das classes, em termos de número de linhas, é um dos fatores considerados ao selecionar as classes, portanto, tentamos selecionar classes, dentro de um mesmo cenário, que possuíam tamanhos semelhantes ou iguais, pois esse fator pode influenciar o tempo de análise da classe. Contudo, nos resultados do experimento notamos que o tamanho das classes não apresentou influências determinantes sobre o tempo de análise. Observando os resultados percebemos que a classe RepositorioCliente apesar de não ser a maior classe em termos de número de linhas (LOC = 173) demandou o maior tempo de análise (19,89 minutos em média). As maiores classes em termos de número de linhas foram PassagemControle e Locadora, ambas possuem um LOC = 213. Contudo, tiveram um tempo de análise de 12,16 minutos para PassagemControle e em média 15,13 minutos para Locadora. Por fim, a classe ContatoDAO apresentou o menor tempo de análise (10,56 minutos em média) e representa a menor classe em termos de número de linhas (LOC = 133).

A Figura 3.7 mostra a variação do número de erros cometidos pelos participantes durante a análise de cada classe. E a Figura 3.8 a média de erros de cada classe. Podemos perceber o que o resultado do T-test mostrou em relação a classe ContatoDAO: a maior parte dos seus valores é diferente dos das outras classes.

O maior número de erros foi da classe ContatoDAO (1,1 erros em média) que mesmo exigindo o menor tempo de análise e possuindo o menor tamanho, aparentemente não foi tão bem compreendida. RepositorioCliente demandou um maior esforço de compreensão, levando mais tempo para ser analisada e possuiu o segundo maior número de erros (0,65 erros em média). Essas duas classes representavam classes de alta coesão tanto conceitual como estrutural.

A classe ContatoDAO, portanto, foi a que demandou menos tempo para ser analisada, porém teve o maior número de respostas erradas a suas perguntas. Inclusive, a diferença do número de erros foi estatisticamente significante. Talvez o número alto de erros tenha ocorrido justamente ao fato das pessoas terem feito sua análise mais apressadamente. Isso pode ser explicado também por essa classe ter sido a última a ser analisada para esse cenário. Outra possível explicação pode ser o fato dessa classe ser dedicada a fazer

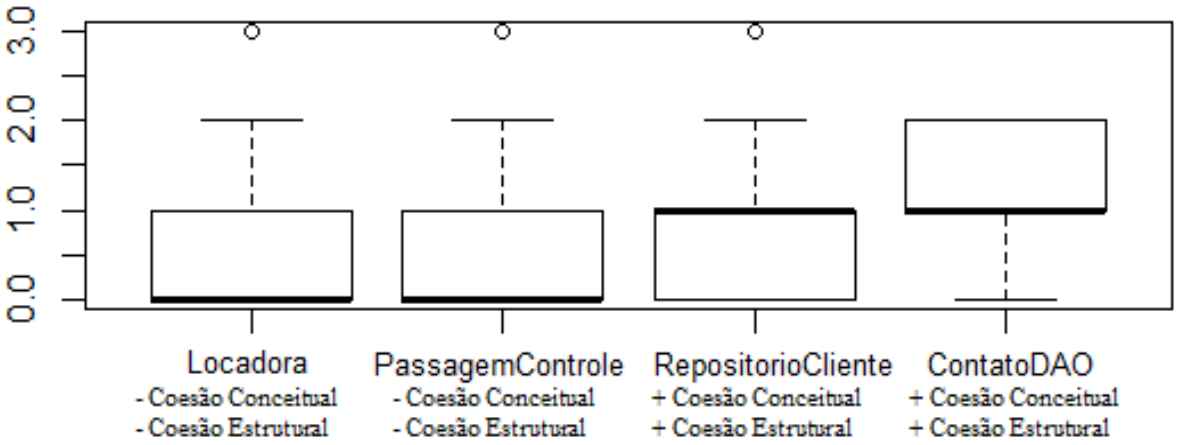


Figura 3.7 Número de erros de cada classe do primeiro cenário.

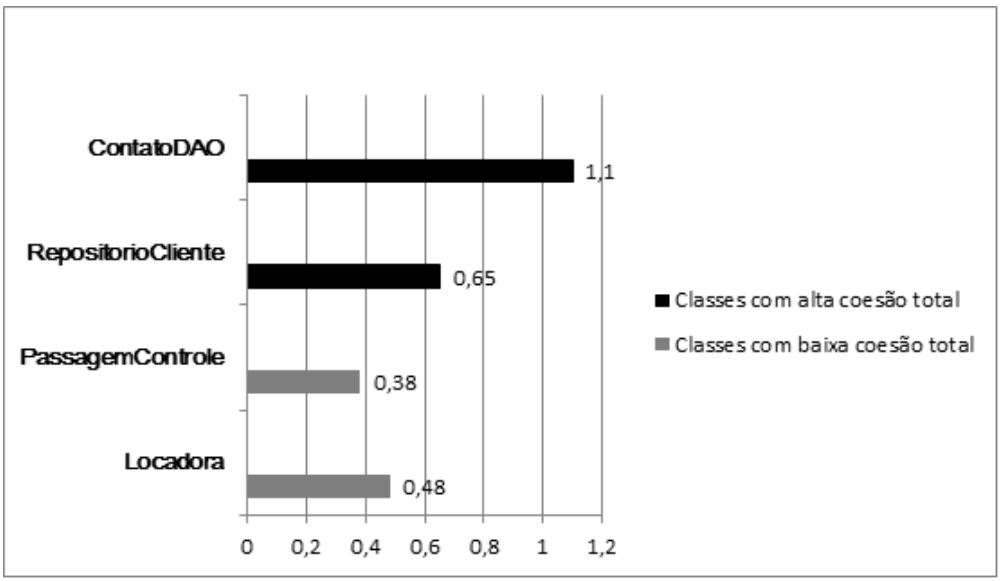


Figura 3.8 Número médio de erros de cada classe do primeiro cenário.

acesso a banco de dados, cujo domínio poderia ser menos natural aos participantes.

Por outro lado, as classes que apresentavam baixa coesão conceitual e estrutural também obtiveram os menores valores de erros médios: 0,38 erros para PassagemControle e 0,48 erros para Locadora. Além disso, elas representam as maiores classes e exigiram menos tempo de análise. Diante disso, podemos perceber que ambas demandaram menos esforço de compreensão.

No questionário de *feedback* os participantes deveriam ordenar as classes da mais fácil para a mais difícil, de acordo com sua opinião, e em seguida justificar os critérios de ordenação. Analisando as respostas do questionário de *feedback*, os participantes consideraram Locadora a classe mais fácil de se compreender, o que de fato condiz com os dados quantitativos. Porém, para eles a classe que demandou maior esforço de compreensão foi PassagemControle, a classe que de acordo com os dados quantitativos apresenta as mesmas características da classe Locadora. Dentre os critérios dessa seleção os participantes mencionaram: simplicidade e objetividade do código, tamanhos dos métodos e clareza do código. Reanalisando o código de cada uma dessas classes observamos que a classe Locadora, de fato, apresenta um código simples e claro, por outro lado, a classe PassagemControle possui um código que utiliza algumas bibliotecas do Java, o que pode ter dificultado na compreensão do participante. De fato, não nos preocupamos com esse possível critério de confusão ao selecionar as classes.

Ao selecionar as classes todo o código da classe é analisado e tentamos selecionar códigos que apresentem similaridades em termos dos fatores de confusão. Também realizamos o estudo piloto e verificamos se há alguma classe muito complexa se comparada às demais. Mesmo assim, as dificuldades encontradas pelos participantes aparentemente foi devido às características do código, seja por falta de conhecimento do participante acerca do domínio da classe ou linguagem ou pela ausência do código do sistema que a classe faz parte, já que o participante tem acesso somente ao código da classe.

Segundo Cenário: Nesse cenário todas as classes tem valor similar de coesão estrutural. O que varia é a coesão conceitual. A análise e apresentação dos dados para esse segundo cenário são feitas da mesma forma que no primeiro cenário. As tabelas 3.8 e 3.9 mostram os valores de p-value para os testes estatísticos. Semelhantemente aos resultados do primeiro cenário, não houve diferenças significativas entre os valores de tempo para analisar as classes (Tabela 3.8). Para todos pares de classes o p-value foi maior que 0,05. Porém, em termos do número de erros, a classe Compromisso obteve valor médio significativamente diferente das demais classes (p-value $\leq$ 0,05) (Tabela 3.9).

Tabela 3.8: Teste Kruskal Wallis para o tempo médio de análise das classes do segundo cenário.

Variância do Tempo	Compromisso	Jogo	Sala
RepositorioFuncionario	0,2872	0,4791	0,5048
Compromisso		0,1437	0,6057
Jogo			0,1539

Tabela 3.9: T- Test para o número médio de erros das classes do segundo cenário.

Variância do Número de Erros	Compromisso	Jogo	Sala
RepositorioFuncionario	0,01658	0,5993	0,8451
Compromisso		0,04808	0,008953
Jogo			0,4529

Analisando os valores do tempo de análise das classes do segundo cenário (Figura 3.9) e a média de tempo exigida por cada classe (Figura 3.10), observamos que a classe Compromisso demandou o maior tempo de análise: em média 18 minutos (com uma mediana de 19 minutos).

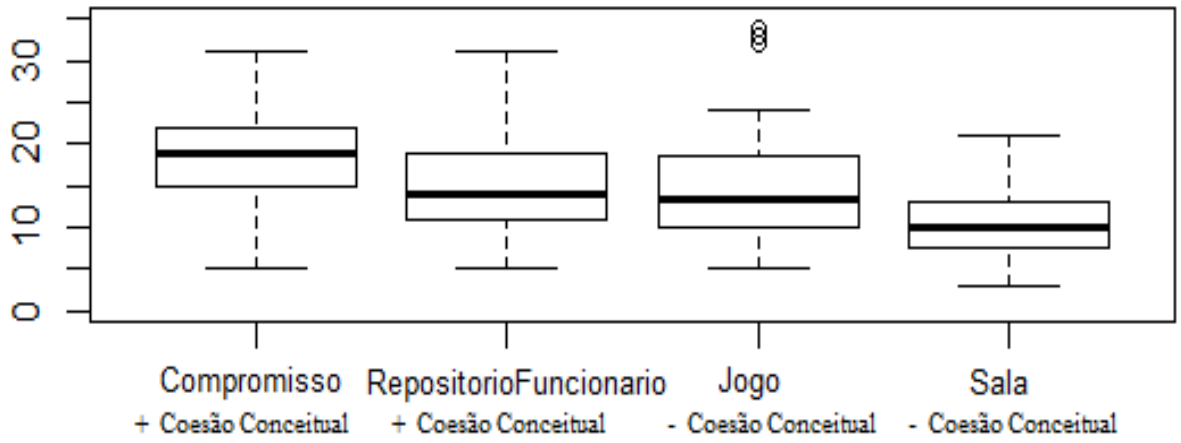


Figura 3.9 Tempo de análise de cada classe do segundo cenário, em minutos.

A classe Jogo demandou em média 15,2 minutos para ser analisada. Observando essas classes veremos que elas possuem valores de coesão distintos: Compromisso apresenta alta coesão conceitual (LCOM5 = 0,1 e LCbC = 2), enquanto que Jogo possui baixa coesão conceitual (LCOM5 = 0 e LCbC = 5).

RepositorioFuncionario exigiu um tempo de análise de em média 15,65 minutos. Apesar de representar classe com alta coesão conceitual (LCOM5 = 0 e LCbC = 2), assim como a classe Compromisso e possuir mais linhas de código, ela demanda menos esforço.

A classe Sala apresentou o menor tempo de análise: 10,8 minutos. Assim como a classe Jogo, essa classe representa classes com baixa coesão conceitual (LCOM5 = 0 e LCbC = 5).

Avaliando os tamanhos das classes, em termos do número de linhas, temos que: a classes Compromisso mesmo não sendo a maior classe (LOC = 135) obteve o maior tempo médio de análise (18 minutos). A maior classe é Jogo possuindo um LOC = 150 e demandou em média 15,2 minutos para ser analisada. RepositorioFuncionario é a segunda maior classe em termos de número de linhas (LOC = 147), contudo, também

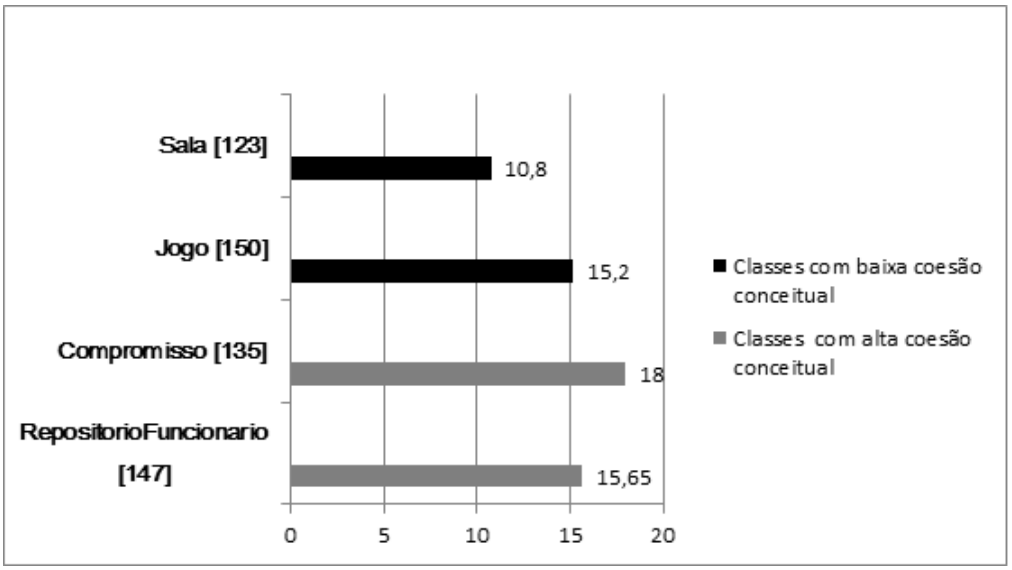


Figura 3.10 Tempo médio de análise de cada classe do segundo cenário, em minutos.

requereu um tempo médio de análise (15,65 minutos) menor que a classe Compromisso. A classe Sala é a menor em termos de número de linhas e também exigiu o menor tempo de análise (em média 10,8 minutos).

Observando os dados a cerca do número de erros de cada classe do segundo cenário (Figura 3.11) e o número médio de erros de cada classe (Figura 3.12), notamos que os resultados mais uma vez são semelhantes aos resultados acerca do tempo de análise.

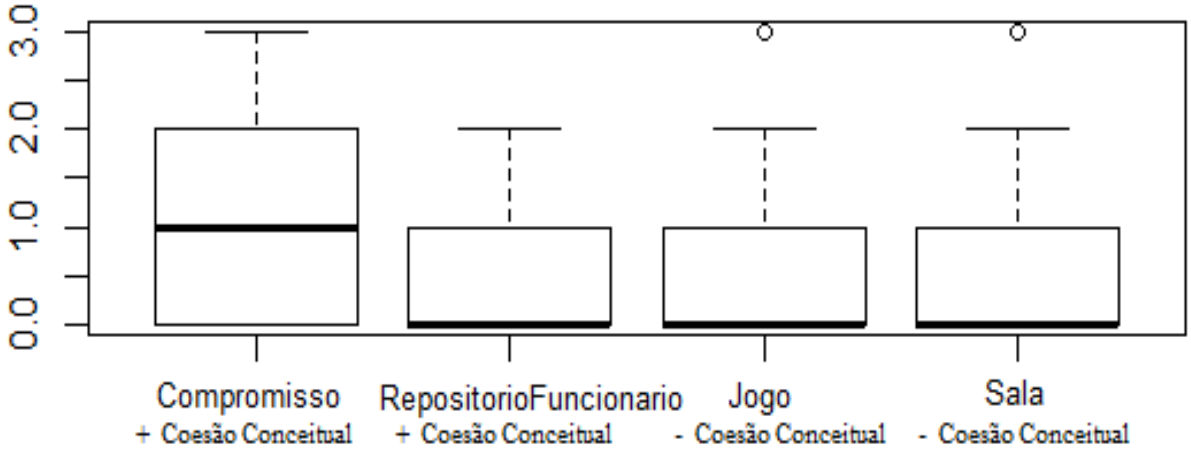


Figura 3.11 Número de erros de cada classe do segundo cenário.

A classe Compromisso foi a que demandou maior esforço de compreensão, pois apesar de necessitar de mais tempo para ser analisada ainda sim obteve o maior número de erros médios: 0,97 erros em média. O que justifica os resultados dos T-test, de fato a classe Compromisso gerou significativamente o maior número de erros.

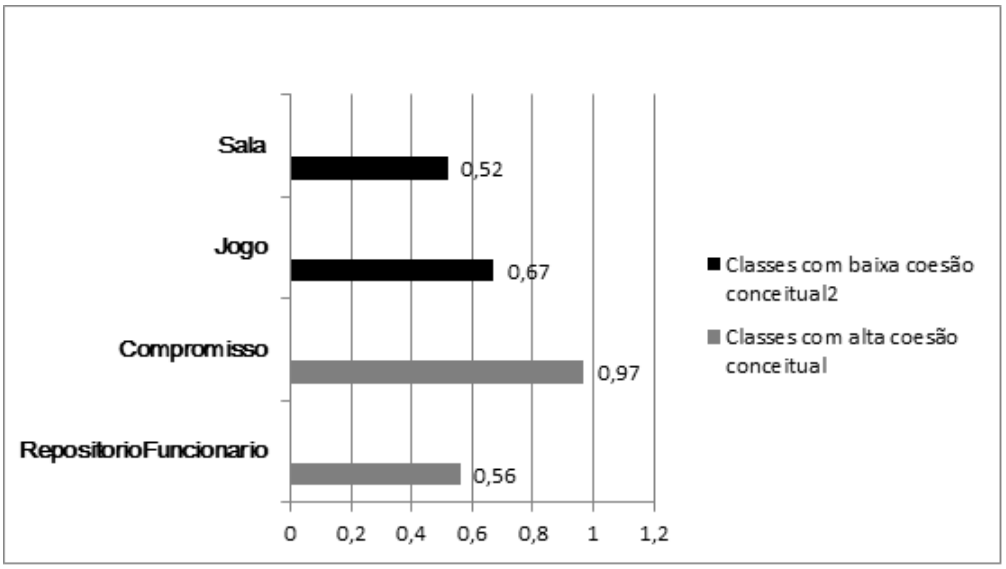


Figura 3.12 Número médio de erros de cada classe do segundo cenário.

A classe Jogo e RepositorioFuncionario apesar de apresentarem valores diferentes de coesão conceitual aparentemente demandaram o mesmo esforço de compreensão. Os tempos médios de cada uma foram bem semelhantes (15,2 minutos para Jogo e 15,65 minutos para RepositorioFuncionario) como também o número médio de erros (0,67 erros para Jogo e 0,56 erros para RepositorioFuncionario).

A classe Sala, assim como na análise do tempo médio, possuiu o menor número médio de erros (0,52 erros) e de acordo com os dados foi a que demandou menos esforço de compreensão.

De acordo com a opinião dos participantes, expressa por meio do questionário de *feedback*, a classe que demandou mais esforço de compreensão foi: Compromisso, o que de fato condiz com os resultados quantitativos. Por outro lado, segundo os participantes a classe que exigiu o menor esforço de compreensão foi a classe Jogo, porém quanto aos dados quantitativos foi a classe Sala. Dentre os critérios da escolha os participantes mencionaram: legibilidade do código, elementos de programação utilizados, organização do código, uso de SQL, tamanho dos métodos e uso de banco de dados. Comparando os códigos dessas duas classes notamos que o código da classe Compromisso utiliza banco de dados e SQL e possui métodos extensos. Por outro lado, a classe Jogo possui um código simples, faz o uso de poucas variáveis e possui métodos pequenos e objetivos. É importante mencionar que, ao selecionar as classes, tentamos encontrar classes com códigos fontes o mais similares possível em termos dos fatores de confusão. Contudo, não é possível achar classes sem diferença alguma. Portanto, o fato da classe Compromisso fazer o uso de SQL pode ter dificultado sua compreensão.

Terceiro Cenário: Nesse cenário todas as classes tem valor similar de coesão conceitual. O que varia é a coesão estrutural. Aplicando também os testes de significância, alcançamos os resultados apresentados na Tabela 3.10 e na Tabela 3.11. Novamente, não observamos diferenças significativas relacionadas ao tempo médio das classes (Tabela

3.10). Contudo, em relação ao número médio de erros existiu diferenças significativas (p-value $\leq$ 0,05) ao se comparar as classes ControleConvenio e Droga (Tabela 3.11). Além disso, a classe RepositorioEndereco apresentou diferenças significativas quando comparada com todas as outras classes (Tabela 3.11).

Tabela 3.10: Teste Kruskal Wallis para o tempo médio de análise das classes do terceiro cenário.

Variância do Tempo	RepositorioEndereco	ControleConvenio	Droga
BombaDeInsulina	0,9773	0,5683	0,5456
RepositorioEndereco		0,3306	0,4635
ControleConvenio			0,285

Tabela 3.11: T- Test para o número médio de erros das classes do terceiro cenário.

Variância do Número de Erros	RepositorioEndereco	ControleConvenio	Droga
BombaDeInsulina	0,000581	0,383	0,08432
RepositorioEndereco		0,003547	8,565e-08
ControleConvenio			0,003154

Avaliando os dados a cerca do tempo de análise das classes do terceiro cenário (Figura 3.13) e a média de tempo demandado por cada classe (Figura 3.14), podemos observar que a classe BombaDeInsulina foi a que demandou mais tempo para análise: em média 15,08 minutos (com uma mediana de 14 minutos). A classe RepositorioEndereco exigiu o segundo maior tempo de análise, em média 13,34 minutos (com mediana de 12 minutos). Ambas as classe possuem baixa coesão estrutural: LCOM5 = 1 e LCbC = 2.

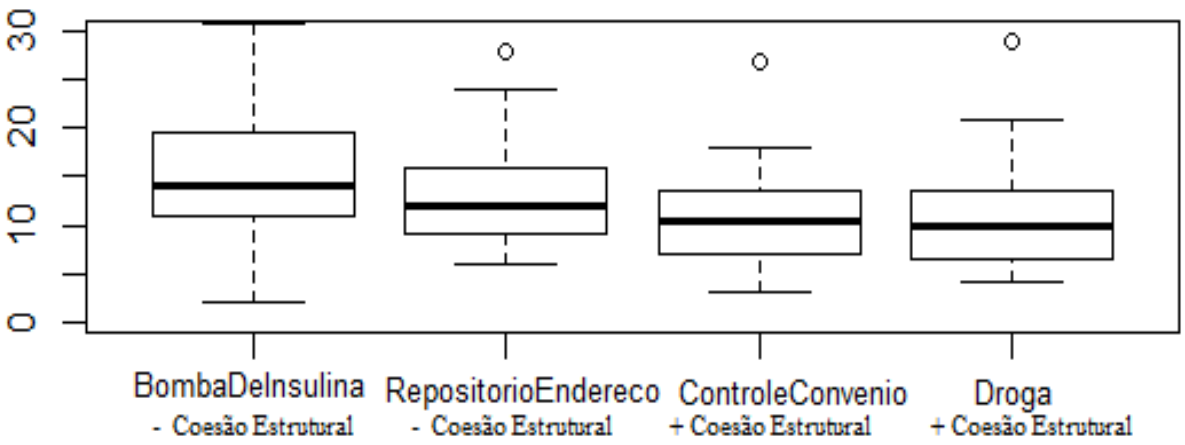


Figura 3.13 Tempo de análise de cada classe do terceiro cenário, em minutos.

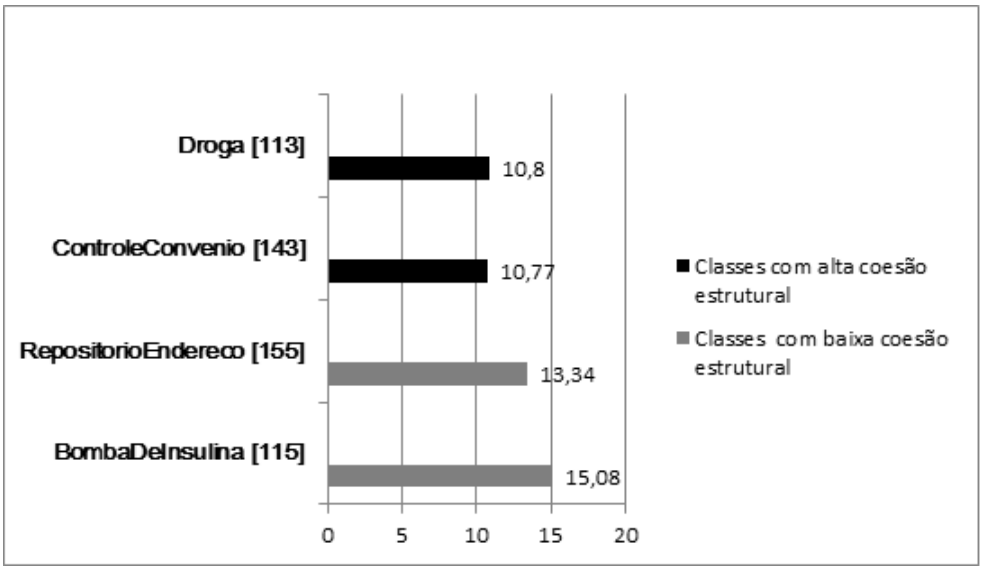


Figura 3.14 Tempo médio de análise de cada classe do terceiro cenário, em minutos.

Analisando as classes com alta coesão estrutural (Droga com LCOM5 = 0 e LCbC = 2 e ControleConvenio com LCOM5 = 0,1 e LCbC = 2) percebemos que elas demandaram tempos de análise bem semelhantes: em média 10,8 minutos para Droga e 10,77 minutos para ControleConvenio.

Avaliando o tamanho das classes, em termos do número de linhas, e comparando com os tempos de análises temos que: a classe BombaDeInsulina apesar de ser a segunda menor classe, com um LOC = 115, foi a que demandou mais tempo de análise (15,08 minutos em média). A maior classe, em termos de número de linhas, é RepositórioEndereço com LOC = 155 e demandou um tempo de análise de em média 13,34 minutos. Droga e ControleConvenio apesar de apresentarem tamanhos bem distintos: LOC = 113 para Droga e LOC= 143 para ControleConvenio, tiveram tempos de análise semelhantes (10,8 minutos para Droga e 10,77 minutos para ControleConvenio). Apesar de ser a segunda maior classe ControleConvenio exigiu um tempo de análise menor que o tempo da classe mais curta.

Considerando os valores de erros por classe (Figura 3.15) e o número médio de erros de cada classe (Figura 3.16), percebemos que a classe RepositorioEndereco apresentou o maior número médio de erros: 1,22 erros. A classe BombaDeInsulina também apresentou uma média alta de erros, 0,62 erros. De acordo com os dados, ambas as classes aparentemente exigiram um maior esforço de compreensão e ambas possuem uma baixa coesão estrutural.

A classe ControleConvenio apesar de demandar o menor tempo de análise apresentou o segundo maior número de erros médios: 0,87 erros. Por outro lado, a classe Droga demandou um dos menores tempos de análise e apresentou o menor número de erros médio: 0,42 erros, caracterizando-se, portanto, como a classe que exigiu o menor esforço de compreensão.

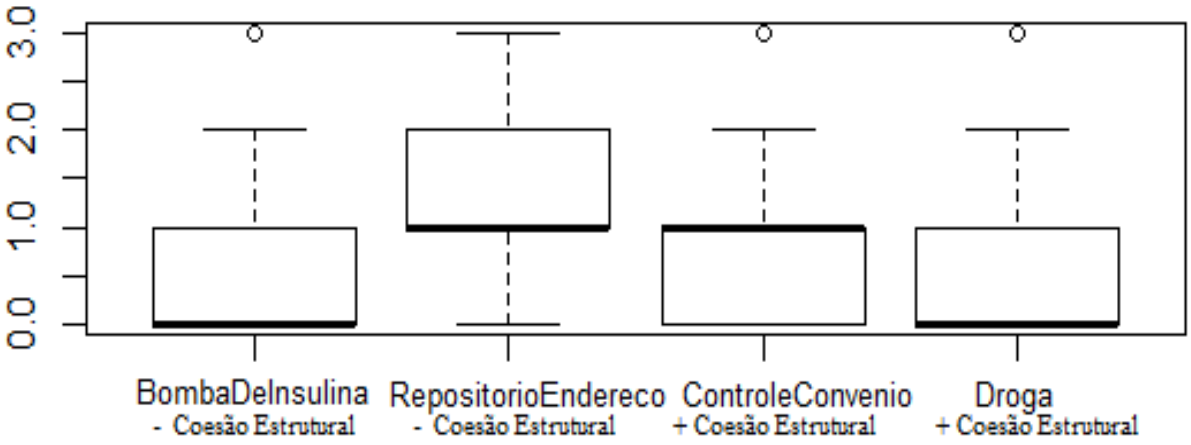


Figura 3.15 Número de erros de cada classe do terceiro cenário.

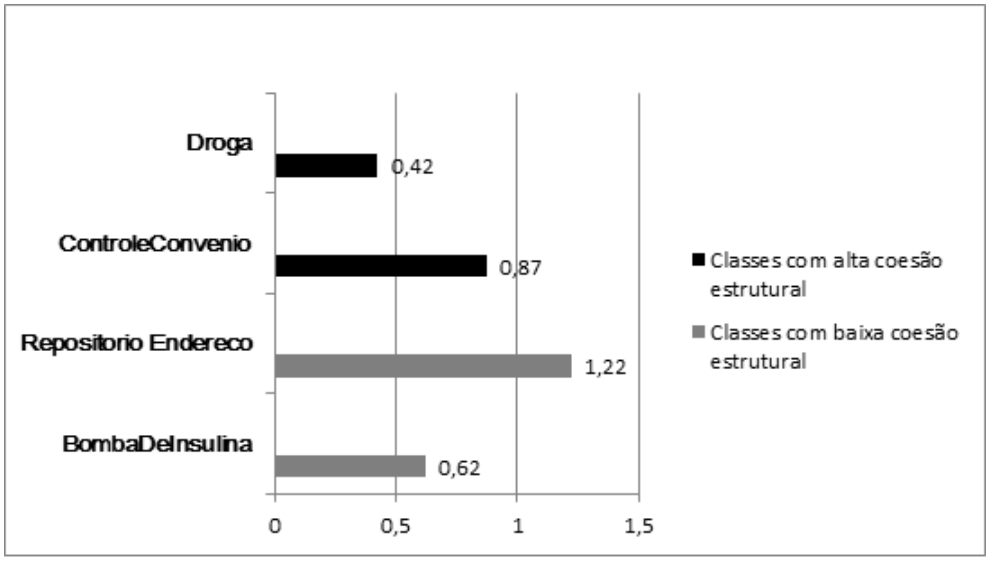


Figura 3.16 Número médio de erros de cada classe do terceiro cenário.

Examinando a opinião dos participantes, a classe que apresentou o maior esforço de compreensão foi ControleConvenio e a que exigiu o menor esforço de compreensão foi RepositorioEndereco. De fato, a classe ControleConvenio apesar de ter tido o menor tempo de análise foi a que apresentou uma grande quantidade de erros, no entanto, a classe RepositorioEndereco, além de demandar uma grande quantidade de tempo de análise foi a que apresentou o maior número de erros e de acordo com os dados quantitativos seria considerada como uma classe que exigiu um maior esforço de compreensão, contradizendo a opinião dos participantes. Isso pode indicar deficiência do método utilizado para medir o esforço de compreensão. Cogitamos estudar essa questão em trabalhos futuros. Avaliando os critérios mencionados pelos participantes temos: Tamanho e quantidade de métodos, complexidade ciclomática, simplicidade e organização do código, regras de negócios, nomenclatura dos identificadores e consultas ao banco de dados. Examinando o código dessas duas classes percebemos que a classe ControleConvenio possui consultas a banco de dados, faz uso de SQL e utiliza muitas bibliotecas Java.

Diante dos resultados, discutimos as questões de pesquisa da seguinte forma:

Questão 1: A coesão de uma classe apresenta alguma influência sobre o esforço de compreensão do seu código? De acordo com os resultados do estudo do primeiro cenário do experimento, onde comparamos classes de alta coesão estrutural e conceitual com classes de baixa coesão estrutural e conceitual, notamos que os valores de coesão não estão influenciando significativamente no esforço de compreensão. Portanto, a resposta para essa pergunta é: **a coesão não influenciou o esforço de compreensão**. Percebemos, por exemplo, nos testes de análise de variância que as classes não apresentaram diferenças significativas em relação ao tempo de análise e quanto ao número de erros notamos maiores diferenças relacionadas aos dados da classe ContatoDAO. Observamos que as classes com baixa coesão foram as que demonstraram um menor esforço de compreensão, enquanto que as classes com alta coesão exigiram um maior esforço para serem compreendidas. Por exemplo, a classe ContatoDAO é uma das classes com alta coesão estrutural e conceitual e contudo, foi a classe que exigiu um maior esforço de compreensão.

Questão 2: Há diferenças no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural? Observando os resultados correspondentes ao segundo e terceiro cenário, onde comparamos classes com valores de coesão estrutural e conceitual variando temos que:

Quanto à coesão conceitual notamos também que ela não apresentou influencia determinante no esforço de compreensão. Assim podemos responder que: **não houve relação entre os valores de coesão conceitual e o esforço de compreensão**. Um exemplo disso é que no segundo cenário, onde variamos a coesão conceitual, existem duas classes (Jogo e RepositorioFuncionario) que possuem valores distintos de coesão conceitual, a classe Jogo com baixa coesão conceitual e a classe RepositorioFuncionario com alta coesão conceitual, porém o esforço de compreensão exigidos por essas classes foram o mesmo. Além disso, a classe que exigiu o maior esforço de compreensão (Compromisso) possui alta coesão conceitual. Mais uma vez, os testes de variância não apontaram a existência de diferenças significativas entre as classes com relação ao tempo de análise. Quando se trata do número de erros somente a classe Compromisso apresentou diferenças

significativas quando comparada as demais classes.

Analisando os resultados do terceiro cenário, onde avaliamos a influência da coesão estrutural sobre o esforço de compreensão notamos também que a coesão estrutural não influencia determinadamente no esforço de compreensão das classes. Assim podemos responder que: **não houve relação entre os valores de coesão estrutural e o esforço de compreensão**. De acordo, com os dados quantitativos notamos que a classe com baixa coesão estrutural, RepositorioEndereco, exigiu um maior esforço de compreensão, assim como a classe ControleConvenio, que apresentar uma alta coesão estrutural e ainda sim demandou um grande esforço de compreensão. E de acordo com os testes de variância, não existiu diferenças entre as classes em termos do tempo de análise. Em relação ao número de erros notamos diferenças significativas especialmente na classe RepositorioEndereco com relação as demais classes do cenário.

Assim como no primeiro cenário, esses dois últimos cenários demonstram, a partir dos resultados e da opinião dos participantes, que outros fatores ligados à estrutura e organização do código fonte, assim como os elementos de programação utilizados no código, aparentemente evidenciam uma maior influência no esforço de compreensão das classes, diferentemente da coesão.

3.2.4 Conclusão

Esse segundo experimento também tinha por objetivo avaliar a relação entre os valores de coesão das classes, calculados por meio de métricas, e o esforço de compreensão do código fonte das classes. Foi possível nesse estudo utilizar um maior número de classes e também uma amostra maior, e ao selecionar as classes consideramos cuidadosamente cada fator de confusão, inclusive o acoplamento, buscando reduzir, ao máximo, a influência desses fatores nos resultados do estudo.

De acordo com os dados quantitativos, **notamos que classes bem coesas não representaram necessariamente classes de fácil compreensão, assim também como classes de baixa coesão não necessariamente eram classes de difícil compreensão**. Segundo os resultados, alguns fatores, diferente da coesão, podem apresentar maior influência sobre o esforço de compreensão, esses fatores estão relacionados à organização do código fonte e ao domínio do código, como por exemplo: Tamanho e quantidade de métodos, simplicidade e organização do código, regras de negócios, nomenclatura dos identificadores, consultas ao banco de dados e uso de bibliotecas java. Esses resultados também são semelhantes à opinião dos participantes, de fato, os participantes consideram outros fatores, diferentes de coesão, como influenciadores no processo de compreensão de código, esses fatores se destacam por representarem características físicas e estruturais do código.

No próximo estudo, permanecemos estudando a relação entre coesão e o esforço de compreensão buscando um aprofundamento melhor na pesquisa. Porém, nesse terceiro estudo tentamos além de minimizar a influência dos fatores de confusão já mencionados, seção 3.1, tentamos minimizar também fatores que tiveram forte influência nos resultados desse segundo experimento. Por isso, no terceiro experimento utilizamos classes de um mesmo sistema, ou seja, classes que apresentam uma mesma estrutura física, um mesmo

domínio, um mesmo estilo de organização e programação.

3.3 EXPERIMENTO III

Nesse estudo os participantes analisaram e realizaram atividades de compreensão do código fonte de seis classes, no período de quatro horas. Essas classes possuíam valores de coesão distintos e pertenciam a um mesmo sistema denominado MobileMedia. O MobileMedia é um sistema que manipula mídias em geral (música, vídeos e fotos) e possui suporte que permite gerenciar essas mídias e utilizá-las em dispositivos móveis. Esse sistema foi desenvolvido na University of British Columbia baseado no sistema MobilePhoto (Young, 2005). Selecionamos esse sistema, pois foi construído com o propósito de pesquisa e já foi utilizado em trabalhos relacionados (Feigenspan et al., 2011). Os participantes tiveram acesso as classes selecionadas e ao sistema completo do MobileMedia por meio de um ambiente de programação denominado Eclipse. Para evitar influência das marcações do Eclipse no estudo, configuramos o ambiente de programação para não mostrar nenhuma marcação ou notificação do compilador.

Nos experimentos anteriores utilizamos classes de sistemas diferentes e buscamos controlar alguns fatores de confusão como: domínio, tamanho e qualidade da nomenclatura dos identificadores. Contudo, a partir do segundo experimento notamos que esses fatores ligados à estrutura e a forma como o código foi construído e organizado demonstram ter grande influência sobre o esforço de compreensão. Sendo assim, nesse terceiro experimento selecionamos classes de um único sistema, desta forma essas classes apresentaram: o mesmo domínio, a mesma qualidade da nomenclatura dos identificadores e um mesmo padrão de estrutura e organização do código. Portanto, a influência desses fatores de confusão foi ainda menor. Por outro lado, o fato de considerarmos apenas um sistema, nos limitou no sentido de encontrar classes com valores de coesão que servissem ao propósito do experimento. Por isso, só conseguimos selecionar seis classes.

3.3.1 Design do Experimento

Métricas: Assim como nos experimentos anteriores, utilizamos a métrica de coesão estrutural *Lack of Cohesion in Methods 5* (LCOM5) (Henderson- Sellers et al., 1996) e a métrica de coesão conceitual *Lack of Concern-Based Cohesion* (LCbC) (Silva et al. 2012) (Seção 3.1). Ao selecionar as classes consideramos também o fator acoplamento, semelhante ao segundo estudo. Por isso, usamos a métrica de acoplamento *Coupling Between Objects* – CBO (Chidamber and Kemerer, 1994), considerando somente um acoplamento *fan-out*, ou seja, calculamos somente o número de classe que uma determinada classe A usa e não as classes que usam A.

Medição do esforço de compreensão: Para quantificar o esforço de compreensão usamos as mesmas métricas dos estudos anteriores: (i) o tempo utilizado pelos participantes para responder um conjunto de perguntas sobre cada uma das classes e (ii) o número de perguntas com respostas erradas. Por fim, calculamos para cada classe o tempo médio de análise e o número médio de erros (Seção 3.1).

As perguntas apresentadas nos questionários também eram caracterizadas como ati-

vidades dinâmicas, as quais, segundo Dunsmore and Roper (2000) refletem melhor a compreensão do participante acerca do código. Para cada classe fizemos três perguntas de diferentes estilos (Apêndice E): (i) uma questão de completar o código, preenchendo determinada linha em branco e (ii) duas questões sobre características específicas dos métodos existentes na classe. Mantemos esse número de questões, pois os participantes iriam analisar todas as classes (seis) de uma só vez. Se não fizéssemos isso, esse experimento iria ficar muito longo.

Estudo Piloto: Antes da execução do experimento novamente realizamos um estudo piloto, para avaliar a configuração do estudo. Esse piloto contou com a participação de dois alunos da pós-graduação em Ciência da Computação, da Universidade Federal da Bahia. Tivemos um participante considerado como experiente (trabalhava com programação por três anos ou mais) e um pouco experiente. Mais uma vez, por meio do estudo piloto: (i) verificamos o tempo médio de execução do experimento, (ii) corrigimos alguns erros ortográficos encontrados nos questionários e (iii) verificamos alguns esclarecimentos que deveriam ser comunicados aos participantes antes da execução do estudo.

Artefatos: Semelhante aos outros estudos, esse experimento apresentou um conjunto de artefatos (Seção 3.1): (i) o termo de consentimento (Apêndice A), (ii) questionário de caracterização (Apêndice B), (iii) questionário de atividades (Apêndice E) e (iv) questionário de *feedback* (Apêndice H).

O questionário de *feedback*, assim como nos estudos anteriores, teve o objetivo de obter um retorno por parte dos participantes sobre o experimento realizado e também adquirir uma análise qualitativa do estudo. Nesse experimento, para adquirir essa análise qualitativa, os participantes deveriam colocar no questionário de *feedback* a classe considerada como a mais fácil de compreender e a classe considerada a mais difícil e citar os critérios utilizados para essa escolha. Como o participante tiver acesso ao código completo do MobileMedia, perguntamos se ele acessou alguma outra classe do sistema além das classes selecionadas, se sim, qual foi essa classe e por quê ele consultou essa classe.

Seleção das Classes: Além de considerar os fatores de confusão (seção 3.1) ao selecionar as classes outro aspecto importante também são os valores de coesão, representados pelas métricas LCOM5 e LCbC. Apesar de selecionar seis classes, conseguimos organizá-las de forma que tivéssemos três cenários, quanto os valores de coesão, como no experimento anterior. Mais uma vez, salientamos que as classificações "alta" e "baixa" são relativas à comparação dos valores entre si e não podem ser generalizadas. A Tabela 3.12 apresenta as classes dividas em cenários e os valores das métricas. Cada cenário tinha quatro classes, desta forma uma mesma classe pertencia a diferentes cenários:

• No primeiro cenário comparamos classes de alta coesão estrutural e conceitual (PlayVideoScreen e PhotoViewScreen) com classes de baixa coesão estrutural e conceitual (ImageMediaAcessor e MusicMediaAcessor).

• No segundo cenário comparamos classes com níveis de coesão estrutural semelhantes, porém a coesão conceitual alta (PlayVideoScreen e PhotoViewScreen) e coesão conceitual baixa (PlayVideoController e MusicPlayController).

• No terceiro cenário comparamos classes com níveis de coesão conceitual semelhantes, mas com coesão estrutural alta (PlayVideoController e MusicPlayController) e coesão estrutural baixa (ImageMediaAcessor e MusicMediaAcessor).

Também buscamos controlar o nível de acoplamento para as classes de um mesmo cenário.

Tabela 3.12: Valores das métricas das classes selecionadas.

Classes	LCOM5	LCbC	CBO	LOC
Primeiro Cenário				
PlayVideoScreen	0,18	3	0	78
PhotoViewScreen	0,09	3	0	35
ImageMediaAcessor	0,67	5	0	53
MusicMediaAcessor	0,60	5	0	58
Segundo Cenário				
PlayVideoScreen	0,18	3	0	78
PhotoViewScreen	0,09	3	0	35
PlayVideoController	0,17	5	1	72
MusicPlayController	0,17	5	1	72
Terceiro Cenário				
PlayVideoController	0,17	5	1	72
MusicPlayController	0,17	5	1	72
ImageMediaAcessor	0,67	5	0	53
MusicMediaAcessor	0,60	5	0	58

Como mencionado, todas as classes selecionadas pertenciam ao sistema MobileMedia. Selecionamos duas classes que tratam da construção de telas (PlayVideoSreen e PhotoViewScreen), duas classes que realizam a função de controladoras (PlayVideoController e MusicPlayController) e por fim, duas classes que tratam organizar os acessos (ImageMediaAcessor e MusicMediaAcessor).

Assim como nos estudos anteriores, para cada classe realizamos um mapeamento dos concerns para então calcular o valor da métrica LCbC, portanto, cada classe apresentou os seguintes interesses demonstrados na Tabela 3.13:

Todas as classes foram aplicadas em um único dia de experimento, contudo, ao analisarmos os resultados dividimos as classes em cenários. Cada participante analisou cada classe apenas uma vez. Ou seja, não significa que uma classe que faz parte de mais de um cenário foi analisada mais de uma vez pelo mesmo participante. Organizamos as classes em cenários, apenas no momento de analisarmos os resultados. Quanto a ordem de aplicação, as classes foram ordenadas de acordo com o tamanho, em termos do número de linhas, e de acordo com as características do domínio. Assim, ordenamos as classes da maior para menor e intercalamos as classes, de forma que as classes com domínios semelhantes não fossem analisadas juntas. As classes foram analisadas na seguinte ordem: PlayVideoScreen, MusicPlayController, ImageMediaAcessor, PhotoViewScreen, PlayVideoController e MusicMediaAcessor.

Participantes: A amostra foi composta por 17 alunos de graduação em Ciência da Computação da Universidade Salvador (UNIFACS) e por 19 alunos de pós-graduação da Universidade Federal da Bahia (UFBA). Mais uma vez, escolhemos a amostra por conveniência uma vez que tínhamos acesso às turmas e em cada turma aplicamos o estudo em dias distintos. O tamanho da amostra foi o mesmo para os três cenários, após

Tabela 3.13: Interesses encontrados nas classes selecionadas.

Classes	Interesses
PlayVideoScreen	Screen Tratamento de exceções Execução do vídeo (Play)
PhotoViewScreen	Imagem SMS Screen
MusicPlayController	Controle da música Tratamento de exceções Nome da mídia Cópia da música Salvar música
PlayVideoController	Controle do vídeo Tratamento de exceções Nome da mídia Cópia do vídeo Salvar vídeo
ImageMediaAcessor	Resetar dados Adicionar imagens de teste Imagens Tratamento de exceções Conversão de dados
MusicMediaAcessor	Resetar dados Adicionar músicas de teste Label Tratamento de exceções Conversão de dados

a limpeza dos dados, caracterizada pela retirada dos valores considerados como *outliers*, ou seja, dados que não pertencem à curva normal e não representam de fato a população, o tamanho da amostra variou de 36 para 31 participantes.

Sequência da execução das atividades: A execução das tarefas seguiu a mesma ordem da sequência feita nos experimentos anteriores: (i) esclarecemos, aos participantes, o que eles deveriam fazer e explicamos sobre o estudo e sobre os questionários que eles iriam receber, (ii) entregamos o termo de consentimento (Apêndice A) e anexado a esse documento estava o questionário de caracterização (Apêndice B), (iii) após o preenchimento desses artefatos o participante poderia solicitar o questionário de atividade (Apêndice E) da primeira classe, finalizada a análise da primeira classe, o participante solicitava o questionário da próxima classe e assim sucessivamente até completar a análise de todas as classes e por fim, (iv) antes de sair o participante recebia o questionário de *feedback* (Apêndice H), depois de preenchido o participante encerrava o experimento.

3.3.2 Ameaças à Validade

Abaixo apresentamos possíveis ameaças à validade do experimento e as ações realizadas para minimizá-las.

Validade Interna: A fadiga dos participantes ao analisar o código fonte de seis classes e realizar atividades de compreensão, em um único dia, é uma possível ameaça à validade interna. Para minimizar a influência da fadiga, realizamos um estudo piloto que determinou o tempo médio do experimento de 55 minutos, o qual consideramos um tempo factível. Participantes com diferentes níveis de experiência em programação poderiam ser outra ameaça. Verificamos a experiência de cada participante, por meio de um questionário de caracterização, e levamos em conta as informações ao analisar os resultados, comparando o desempenho dos participantes experientes com os participantes pouco experientes. Também nos preocupamos com a seleção das classes e tentamos minimizar ao máximo alguns fatores de confusão ao selecionar as classes. Por fim, nesse terceiro experimento, o fato de todos os participantes terem analisados as classes numa mesma ordem pode representar um ameaça mais significativa à validade, pois as classes pertencem a um mesmo sistema. Portanto, os participantes podem ter aprendido mais sobre o domínio das classes à medida que analisavam cada uma delas. Tendo conhecimento disso, levamos essa possibilidade em consideração ao analisarmos os dados. Em replicações futuras desse experimento ou execução de experimentos similares, adotaremos medidas para que diferentes participantes analisem as classes em ordens diferentes.

Validade Externa: O estudo possui a limitação de não poder ser generalizado devido à pequena quantidade de classes e à pequena amostra de participantes, composta somente por estudantes.

Validade de Construto: Assim como nos estudos anteriores, a medição do esforço de compreensão também pode ser outra ameaça e, portanto, utilizamos procedimentos indicados pela literatura e já utilizados em outros estudos (Dunsmore and Roper, 2000)(Feigenspan et al., 2011). A clareza das perguntas apresentadas aos participantes e também os diferentes níveis de dificuldade das questões poderiam também representar ameaças ao esforço de compreensão. Por isso, realizamos o estudo piloto e assim modificamos

e ajustamos as perguntas, para melhor entendimento dos participantes e analisamos se haviam diferenças significativas com respeito o nível de dificuldade das questões.

3.3.3 Resultados e Discussões

Da mesma forma que o experimento anterior, o teste de Shapiro-Wilks mostrou que, para os três cenários, o número de erros seguia uma distribuição normal, enquanto o tempo de análise não seguia. Dessa forma, nos três cenários, usamos os testes Kruskal Wallis e T-test em cada um dos casos. Assim como nos estudos anteriores, as figuras que apresentam os boxplots possuem uma legenda indicando o nível de coesão de cada classe, onde o sinal '+' representa alta coesão e o sinal '-' baixa coesão.

Primeiro Cenário: As tabelas 3.14 e 3.15. mostram os resultados dos testes estatísticos para esse cenário. Mais uma vez não notamos diferenças significativas entre os dados que correspondem ao tempo de análise das classes. O p-value foi maior que 0,05 para todas as classes (Tabela 3.14). Por outro lado, a Tabela 3.15 mostra que houve diferenças significativas (p-value $\leq$ 0,05) no número de erros da classe PhotoViewScreen quando comparada com as demais classes.

Tabela 3.14: Teste Kruskal Wallis para o tempo médio de análise das classes do primeiro cenário.

Variância do Tempo	PhotoViewScreen	ImageMediaAcessor	MusicMediaAcessor
PlayVideoScreen	0,2298	0,3527	0,384
PhotoViewScreen		0,08133	0,2583
ImageMediaAcessor			0,4376

Tabela 3.15: T- Test para o número médio de erros das classes do primeiro cenário.

Variância do Número de Erros	PhotoViewScreen	ImageMediaAcessor	MusicMediaAcessor
PlayVideoScreen	3,765e-05	0,2422	0,5276
PhotoViewScreen		0,004838	0,0247
ImageMediaAcessor			0,8835

A Figura 3.17 mostra os boxplots do tempo de análise das classes do primeiro cenário e a Figura 3.18 os valores de tempo médio de cada classe. Podemos observar que a classe PlayVideoScreen demandou mais tempo para análise, em média 12,89 minutos, sendo que, de acordo com a mediana metade dos participantes levaram até 11 minutos para analisar a classe. Essa classe se caracteriza por apresentar alta coesão estrutural e conceitual (LCOM5 = 0.18 e LCbC = 3).

A classe PhotoViewScreen também possui alta coesão conceitual e estrutural (LCOM5 = 0.09 e LCbC = 3), contudo demandou um tempo de análise de 7,2 minutos em média e metade dos participantes levaram até seis minutos para analisar a classe.

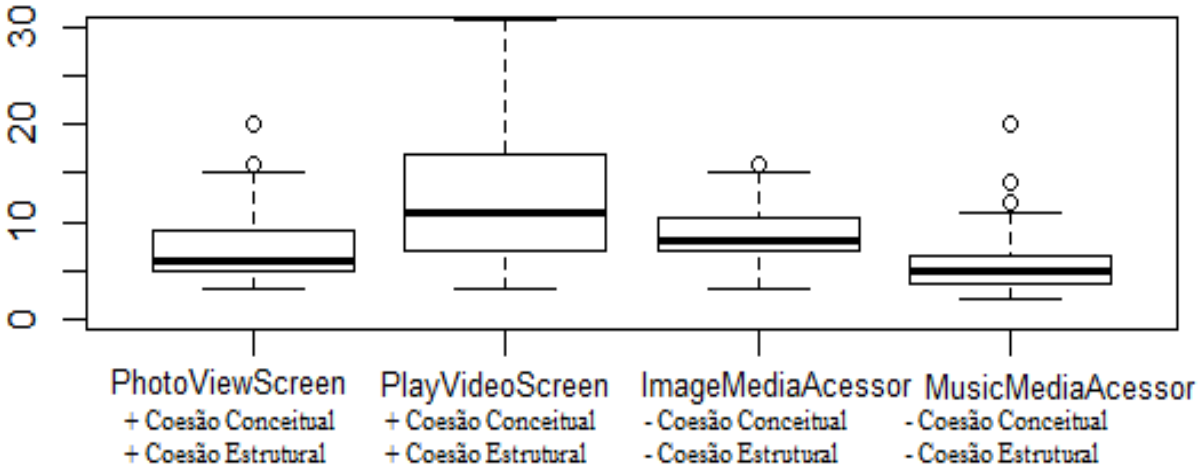


Figura 3.17 Tempo de análise de cada classe do primeiro cenário, em minutos.

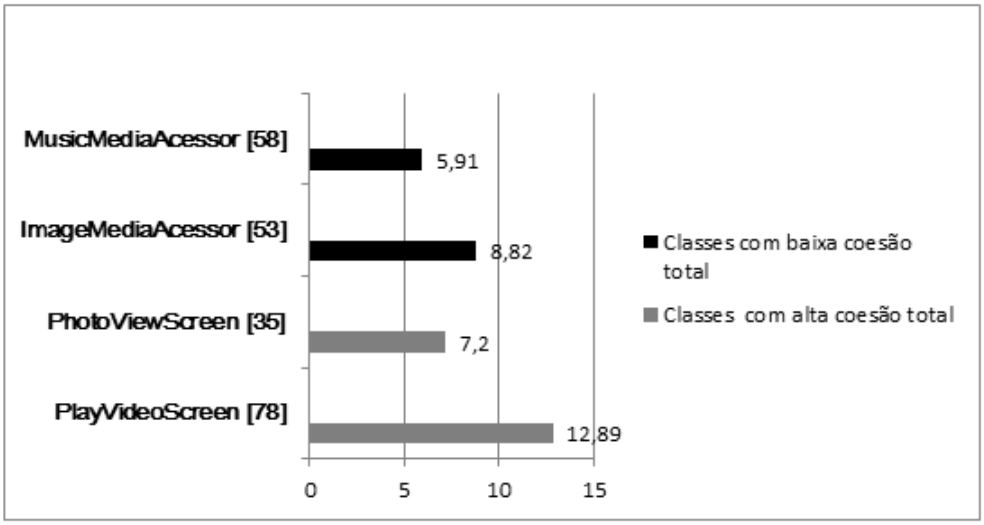


Figura 3.18 Tempo médio de análise de cada classe do primeiro cenário, em minutos.

MusicMediaAcessor foi a classe que demandou o menor tempo de análise, em média 5,91 minutos e metade dos participantes levaram até cinco minutos para analisá-la. Essa classe possui baixa coesão conceitual e estrutural (LCOM5 = 0.60 e LCbC = 5).

A classe ImageMediaAcessor também possui baixa coesão conceitual e estrutural (LCOM5 = 0.67 e LCbC = 5) e necessitou em média de 8,82 minutos para ser analisada, sendo que metade dos participantes levaram até oito minutos para finalizar a análise.

Quanto ao número de linhas de cada classe temos que: PlayVideoScreen é a maior classe me termos de número de linhas (LOC = 78) e demandou o maior tempo de análise (12,89 minutos em média). PhotoViewScreen representa a menor classe (LOC = 35), porém exigiu o segundo menor tempo de análise (7,2 minutos em média). A classe MusicMediaAcessor mesmo sendo a segunda maior classe em termos de número de linhas (LOC = 58) demandou o menor tempo de análise (5,91 minutos em média). ImageMediaAcessor é a segunda menor classe (LOC = 53), porém necessitou de 8,82 minutos em média para análise.

A Figura 3.19 apresenta os boxplots do número de erros de cada classe do primeiro cenário e a Figura 3.20 o número médio de erros de cada classe. Notamos que a classe PlayVideoScreen além de demandar maior tempo também obteve o maior número médio de erros, 1,25 em média e essa classe se caracteriza por possuir alta coesão conceitual e estrutural (LCOM5 = 0.18 e LCbC = 3). Porém, a classe PhotoViewScreen que apresenta as mesmas características da classe PlayVideoScreen, em termos de níveis de coesão, possui o menor número médio de erro, 0,57 em média.

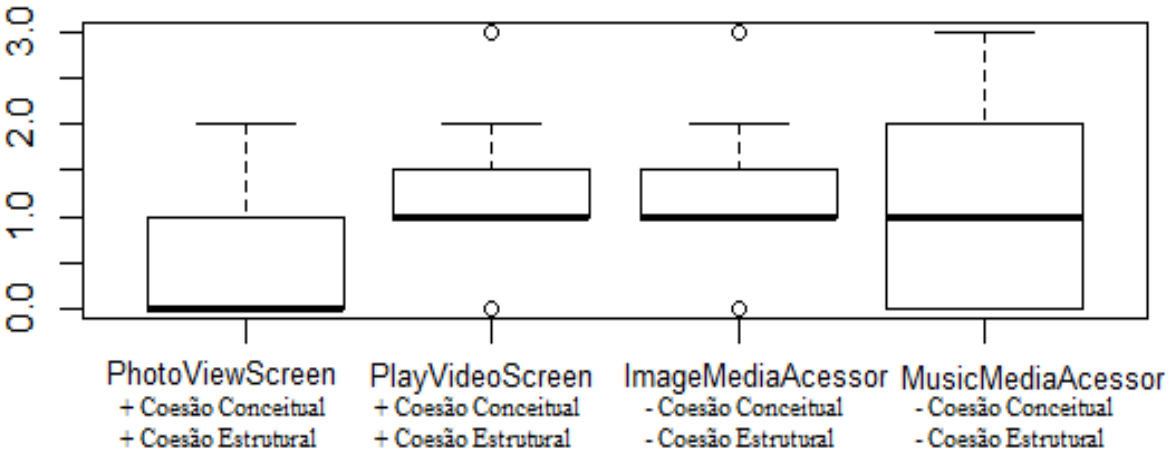


Figura 3.19 Número de erros de cada classe do primeiro cenário.

Por outro lado, as classes que possuem baixa coesão conceitual e estrutural, apresentaram uma média de erros bem parecidas, ImageMediaAcessor obteve uma média de 1,11 erros e MusicMediaAcessor uma média de 1,02 erros.

De acordo com os resultados percebemos que a classe que demanda maior esforço de compreensão nesse primeiro cenário foi a classe PlayVideoScreen e a classe que demandou menor esforço foi PhotoViewScreen e MusicMediaAcessor.

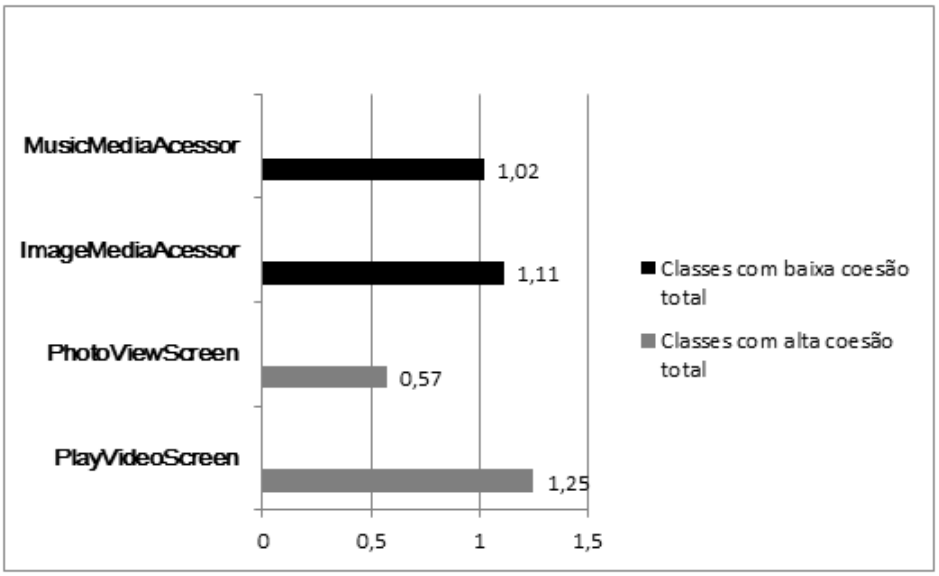


Figura 3.20 Número médio de erros de cada classe do primeiro cenário.

Segundo Cenário: Nesse cenário, as classes tem valores similares de coesão estrutural. Cada par de classes tem valores distintos de coesão conceitual. Os resultados dos testes de variância para as classes do segundo cenário são apresentados nas tabelas 3.16 e 3.17. Não houve diferenças significativas quanto ao tempo médio de análise das classes (Tabela 3.16). Contudo, em relação ao número médio de erros, houve diferenças significativas (p-value $\leq$ 0,05), das classes PlayVideoScreen e PlayVideoController em relação às outras duas (Tabela 3.17).

Tabela 3.16: Teste Kruskal Wallis para o tempo médio de análise das classes do segundo cenário.

Variância do Tempo	PhotoViewScreen	PlayVideoController	MusicPlayController
PlayVideoScreen	0,2298	0,7393	0,2073
PhotoViewScreen		0,4585	0,07672
PlayVideoController			0,6829

Tabela 3.17: T- Test para o número médio de erros das classes do segundo cenário.

Variância do Número de Erros	PhotoViewScreen	PlayVideoController	MusicPlayController
PlayVideoScreen	3,765e-05	0,674	0,0005074
PhotoViewScreen		0,001918	0,5813
PlayVideoController			0,01024

Para avaliar detalhadamente os resultados do estudo, a Figura 3.21 apresenta os boxplots correspondentes ao tempo de análise das classes do segundo cenário e a Figura 3.22 o tempo médio de análise de cada classe. É possível notar que as classes PlayVideoScreen e PhotoViewScreen também pertencem a esse cenário, comparando essas classes com as outras classes do segundo cenário observamos que mais uma vez a classe PlayVideoScreen demanda o maior tempo de análise, em média 12,89 minutos e se caracteriza por possuir alta coesão conceitual (LCOM5 = 0.18 e LCbC = 3).

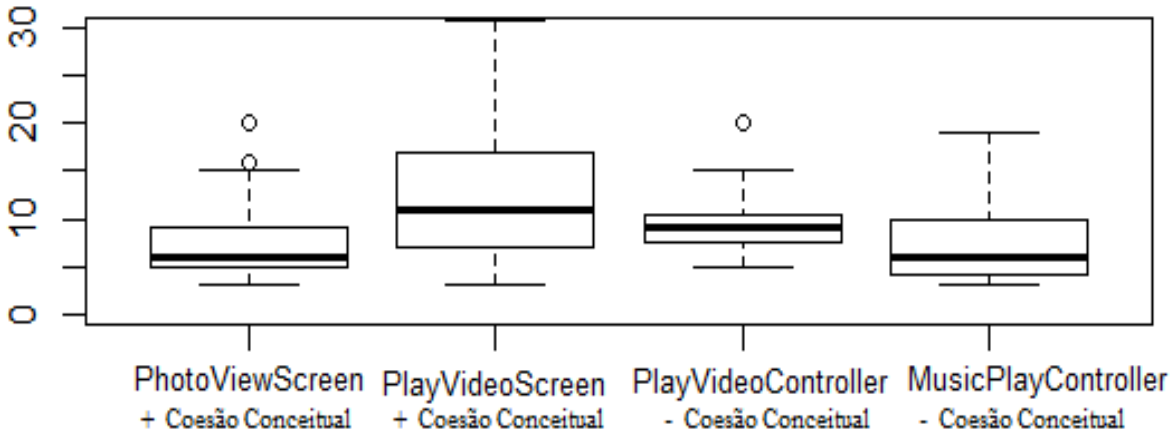


Figura 3.21 Tempo de análise de cada classe do segundo cenário, em minutos.

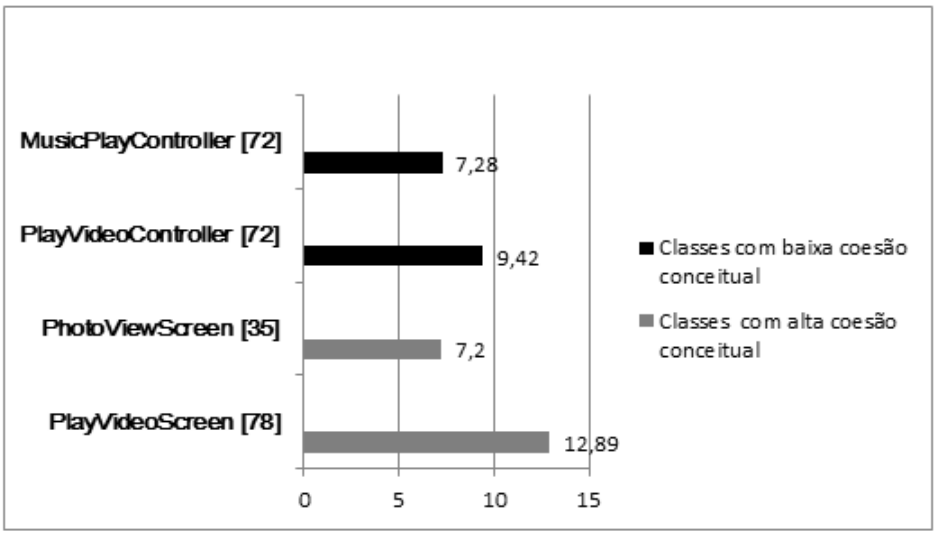


Figura 3.22 Tempo médio de análise de cada classe do segundo cenário, em minutos.

As classes MusicPlayController que apresenta baixa coesão conceitual (LCOM5 = 0.17 e LCbC = 5) e a classe PhotoViewScreen que possuía alta coesão conceitual (LCOM5 = 0.09 e LCbC = 3), demandaram tempo de análise bastante similares. MusicPlayController

exigiu em média 7,28 minutos e PhotoViewScreen demandou 7,2 minutos em média. Em ambas as classes metade da amostra necessitou de até seis minutos para analisá-las, ou seja, possuíram o mesmo valor para mediana (seis).

A classe PlayVideoController possui o mesmo nível de coesão conceitual da classe MusicPlayController (baixa coesão conceitual: LCOM5 = 0.17 e LCbC = 5), contudo demandou o segundo maior tempo de análise, em média 9,42 minutos.

Quanto ao tamanho das classes temos que: PlayVideoScreen possui o maior número de linhas (LOC=78) e exigiu o maior tempo de análise (12,89 minutos em média). MusicPlayController é a segunda maior classe (LOC = 72) e PhotoViewScreen é a menor classe (LOC = 35), porém essas classes apresentaram tempo de análise bem semelhantes (7,2 minutos para PhotoViewScreen e 7,28 minutos para MusicPlayController). A classe PlayVideoController possui o mesmo número de linhas da classe MusicPlayController (LOC = 72), mas demandou mais tempo para ser analisada (9,42 minutos em média).

Avaliando o número de erros das classes do segundo cenário, por meio da Figura 3.23 e Figura 3.24, observamos que as classes que demandaram os maiores tempos de análise (PlayVideoScreen e PlayVideoController) também tiveram o maior número de erros médio: PlayVideoScreen com em média 1,25 erros e PlayVideoController com em média 1,14 erros. Apesar de ambas as classes exigirem maior esforço de compreensão, estas apresentam níveis de coesão conceitual distintos: PlayVideoScreen alta coesão conceitual (LCbC = 3) e PlayVideoController baixa coesão conceitual (LCbC = 5).

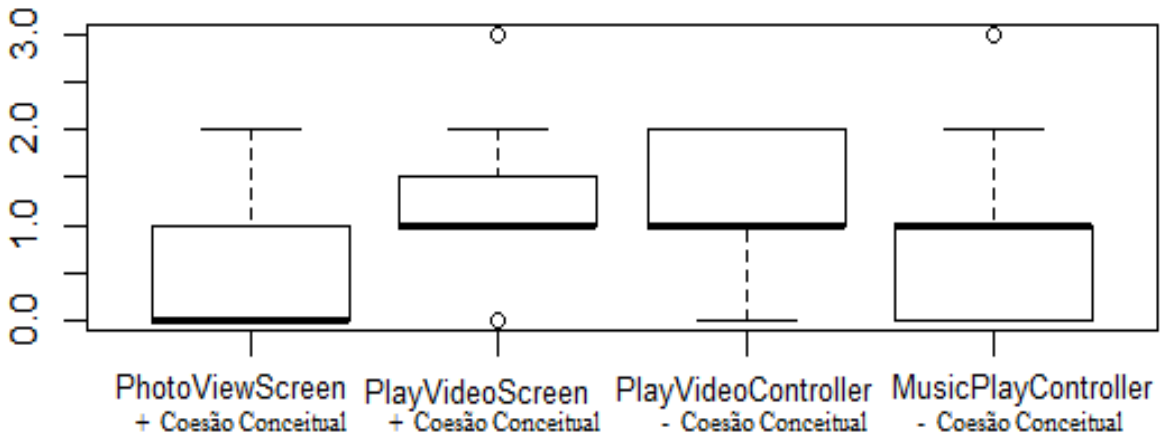


Figura 3.23 Número de erros de cada classe do segundo cenário.

As classes PhotoViewScreen e MusicPlayController obtiveram os menores números médios de erros: PhotoViewScreen com em média 0,57 erros e MusicPlayController com em média 0,71 erros. Contudo, essas classes também possuem valores de coesão e tamanhos distintos: PhotoViewScreen possui alta coesão conceitual (LCbC = 3) e LOC = 35, por outro lado, MusicPlayController possui baixa coesão conceitual (LCbc = 5) e LOC = 72.

Terceiro Cenário: Nesse cenário, as classes tem valores iguais de coesão conceitual, e cada par de classes tem valores distintos de coesão estrutural. Os resultados dos testes

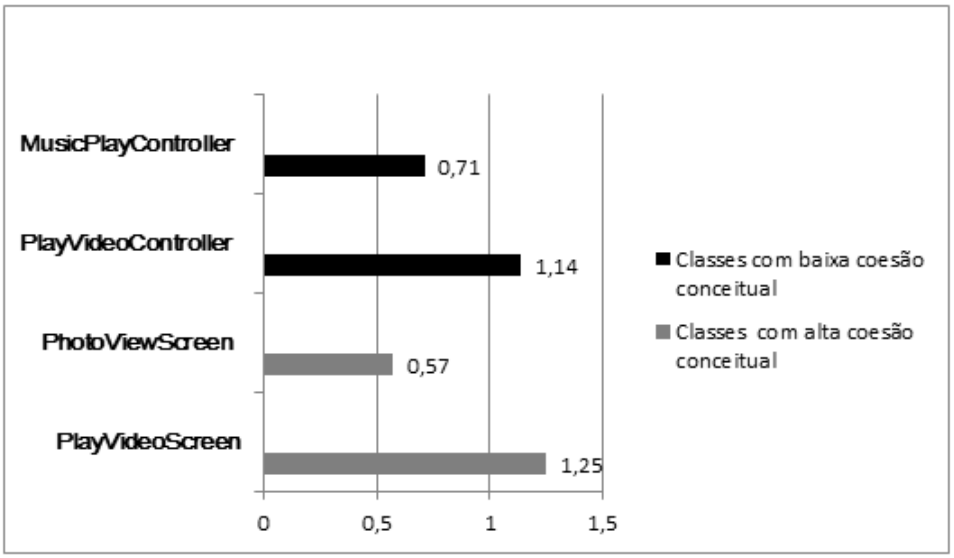


Figura 3.24 Número médio de erros de cada classe do segundo cenário.

estatísticos são mostrados nas tabelas 3.18. e 3.19. Mais uma vez, não houve diferenças significativas nos tempos de análise (Tabela 3.18). Com relação ao número médio de erros, a classe MusicPlayController se difere significantemente das classes ImageMediaAcessor e PlayVideoController (Tabela 3.19).

Tabela 3.18: Teste Kruskal Wallis para o tempo médio de análise das classes do terceiro cenário.

Variância do Tempo	MusicMediaAcessor	PlayVideoController	MusicPlayController
ImageMediaAcessor	0,4376	0,5675	0,1965
MusicMediaAcessor		0,608	0,8166
PlayVideoController			0,6829

Tabela 3.19: T- Test para o número médio de erros das classes do terceiro cenário.

Variância do Número de Erros	MusicMediaAcessor	PlayVideoController	MusicPlayController
ImageMediaAcessor	0,8835	0,5796	0,02579
MusicMediaAcessor		0,7795	0,06909
PlayVideoController			0,01024

Analisando melhor os resultados, podemos observar os dados correspondentes ao tempo de análise das classes do terceiro cenário na Figura 3.25 e na Figura 3.26 o tempo médio de análise de cada classe. Percebemos que as classes desse cenário também fazem parte dos cenários anteriores.

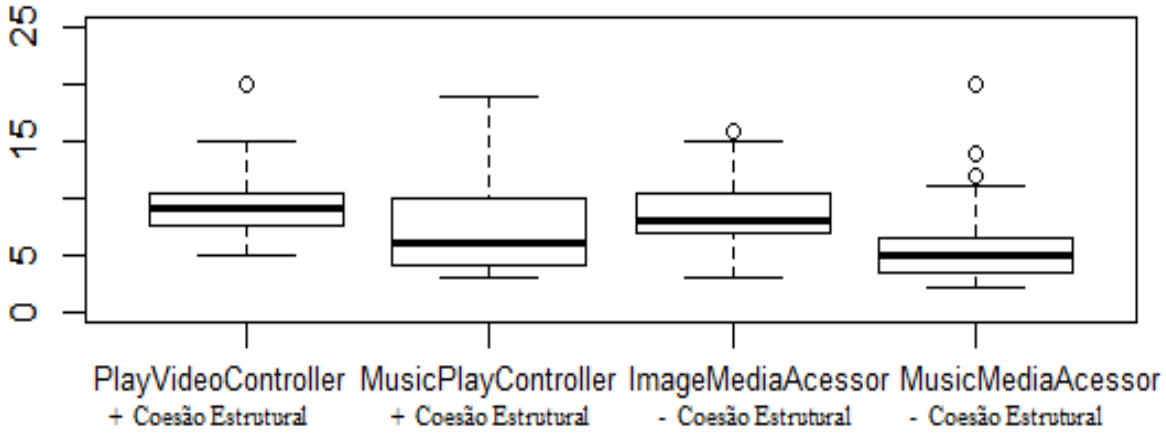


Figura 3.25 Tempo de análise de cada classe do terceiro cenário, em minutos.

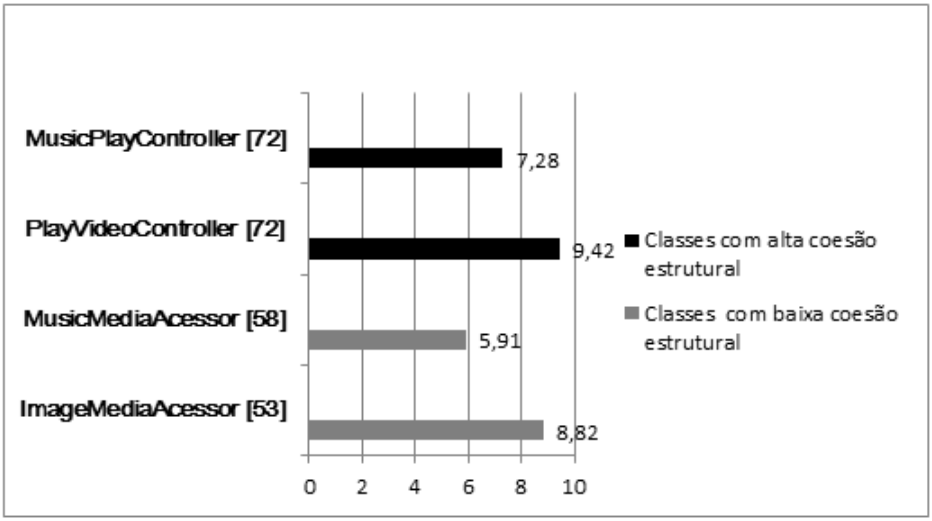


Figura 3.26 Tempo médio de análise de cada classe do terceiro cenário, em minutos.

De acordo com os dados a classe PlayVideoController exigiu o maior tempo de análise, em média 9,42 minutos, essa classe apresenta alta coesão estrutural (LCOM5 = 0.17 e LCbC = 5).

A classe MusicPlayController possui as mesmas características da classe PlayVideoController, ou seja, alta coesão estrutural (LCOM5 = 0.17 e LCbC = 5), porém demandou o segundo menor tempo médio de análise, 7,28 minutos.

A classe ImageMediaAcessor possui baixa coesão estrutural (LCOM5 = 0.67 e LCbC = 5) e essa classe demandou o segundo maior tempo de análise, em média 8,82 minutos. A classe MusicMediaAcessor também possui baixa coesão estrutural, mas demandou o menor tempo de análise de em média 5,91 minutos.

Em termos de número de linhas das classes do terceiro cenário, notamos que: PlayVideoController e MusicPlayController são as maiores classes com LOC = 72, e possuem características iguais quanto a coesão, porém demandaram tempo de análise distintos (em média 9,42 minutos para PlayVideoController e 7,28 minutos em média para MusicPlayController). A menor classe em termos de número de linhas foi ImageMediaAcessor (LOC = 53), porém exigiu 8,82 minutos em média de análise, não sendo o menor tempo. A classe MusicMediaAcessor com LOC = 58 possuiu o menor tempo de análise (5,91 minutos em média).

Quanto ao número de erros podemos observar os resultados na Figura 3.27 e a média de número de erros de cada classe na Figura 3.28. De acordo com as figuras notamos que o maior número médio de erros foi da classe PlayVideoController com 1,14 erros, essa classe exigiu o maior tempo de análise e apresenta o maior número de linhas (LOC = 72), quanto a coesão essa classe possui alta coesão estrutural (LCOM5 = 0.17).

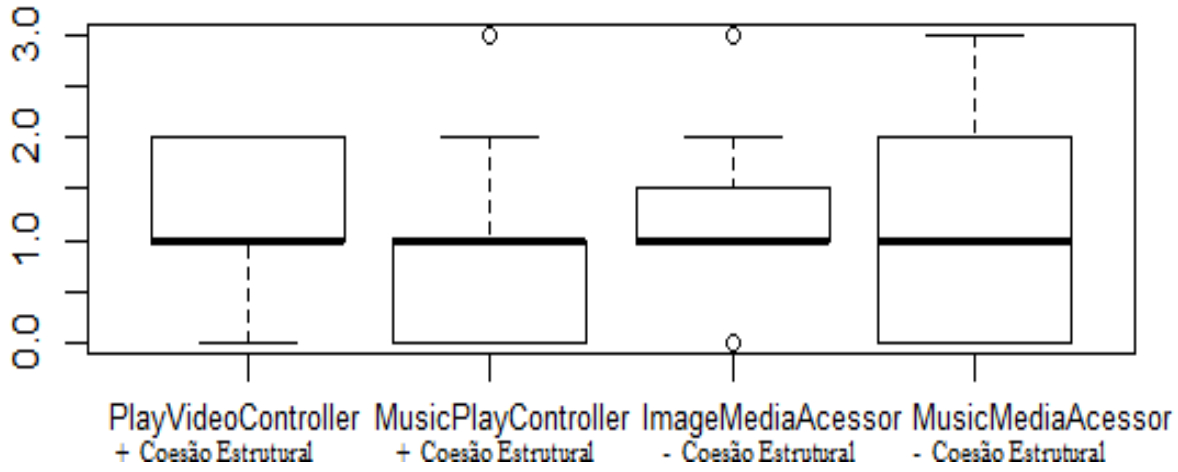


Figura 3.27 Número de erros de cada classe do terceiro cenário.

A classe ImageMediaAcessor também apresentou uma alto número médio de erros, 1,11. Contudo, essa classe possui característica contrárias à classe PlayVideoController. ImageMediaAcessor apresenta baixa coesão estrutural (LCOM5 = 0,67) e LOC = 53.

A classe MusicMediaAcessor também possui baixa coesão estrutural (LCOM5 = 0.60), LOC = 58 e obteve um número médio de erros de 1,02, características semelhantes a da

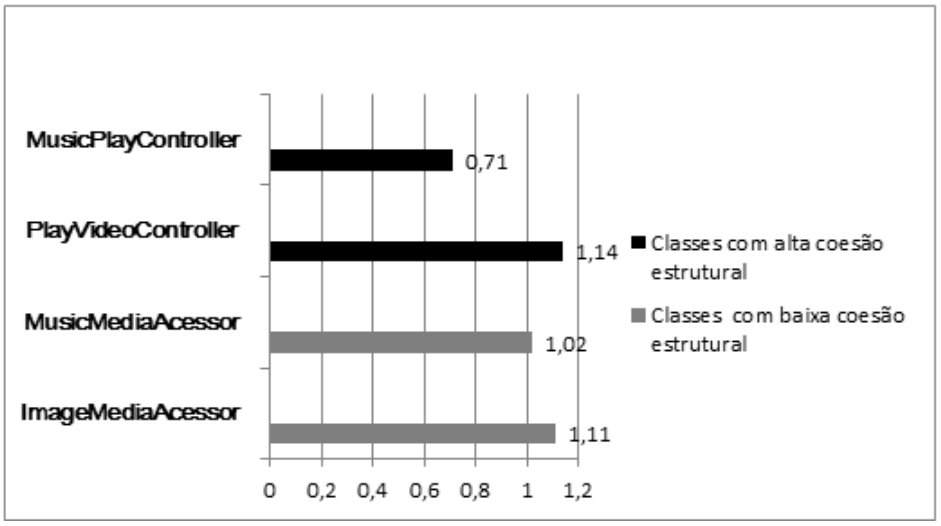


Figura 3.28 Número médio de erros de cada classe do terceiro cenário.

classe ImageMediaAcessor.

O menor número médio de erros foi da classe MusicPlayController, 0,71 erros, a qual possuiu também o menor tempo médio de análise, o maior número de linhas (LOC = 72) e uma alta coesão estrutural (LCOM5 = 0.17). Devido ao menor número de erros essa classe apresentou diferenças significativas em relação às demais no teste de variabilidade.

De acordo com as respostas dadas pelos participantes no questionário de feedback, que solicitava do participante uma análise da aplicação do experimento e uma análise acerca da facilidade de compreensão das classes. Para os participantes, o estudo foi de dificuldade média e a maioria selecionou a classe PlayVideoScreen como a mais difícil de compreender. A opinião dos participantes a respeito da classe que exigiu maior esforço de compreensão foi semelhante aos resultados quantitativos, de fato, a classe PlayVideoScreen demandou o maior tempo de análise, em média 12,89 minutos e também obteve o maior número médio de erros, 1,25 erros. Quanto à coesão essa classe possui alta coesão estrutural e conceitual (LCOM5 = 0.17 e LCbC = 3). Quanto à classe mais fácil de compreender, a maior parte dos participantes selecionou a classe MusicMediaAcessor. Mais uma vez a opinião dos participantes foi semelhante aos resultados quantitativos, a classe MusicMediaAcessor foi a que demandou o menor tempo de análise, em média 5,91 minutos, contudo o número de erros foi 1,02, o terceiro menor número de erros. Quanto à coesão a classe MusicMediaAcessor possui baixa coesão estrutural e baixa coesão conceitual (LCOM5 = 0.67 e LCbC = 5).

Para justificar a escolha da classe mais difícil e mais fácil de compreender os participantes em sua maioria citaram: o domínio da classe, a simplicidade e legibilidade do código, o tamanho da classe, a lógica do código, a clareza dos métodos e os comandos utilizados pelas classes. Além desses fatores a escolha dessas classes pode ter sido influenciada pela ordem de aplicação das classes, uma vez que a classe considerada a mais difícil foi a primeira classe aplicada e a classe considerada a mais fácil foi a última classe aplicada no experimento. Sendo assim, com a análise das classes o participante pode ter

adquirido experiência e uma maior facilidade em compreender o código, pois ele já estava habituado com o domínio do código, sendo assim a última classe foi mais simples para os participantes e a primeira mais complexa. Esse pode ter sido um problema de termos selecionado classes de um mesmo sistema. Em replicações futuras desse experimento, pretendemos alternar a ordem de análise das classes por parte dos participantes. Contudo, de acordo com os dados quantitativos, mesmo possuindo menor tempo de análise a última classe (ImageMediaAcessor) ainda assim apresentou um número de erros alto. Demostrando que ela demandou menos tempo para análise provavelmente por ter sido a última classe, porém não foi tão bem compreendida.

Os participantes tiveram acesso a todo o código do MobileMedia, apesar de poderem acessar outras classe, além das classes que faziam parte do experimento, a maioria dos participantes não consultou nenhuma outra classe além das classes selecionadas para o estudo. Somente 12 participantes consultaram outras classes do sistema e justificaram a consulta como um busca por maiores informações, ainda sim, a maioria desses participantes acessaram somente uma única classe diferente das classes do estudo, a classe MediaAcessor.

Diante dos resultados, discutimos as questões de pesquisa da seguinte forma:

Questão 1: A coesão de uma classe apresenta alguma influência sobre o esforço de compreensão do seu código? De acordo com os resultados do primeiro cenário do experimento, onde comparamos classes de alta coesão estrutural e conceitual com classes de baixa coesão estrutural e conceitual, notamos que **os valores de coesão não estão influenciando significativamente no esforço de compreensão**. Notamos que entre as classes que exigiram o maior esforço de compreensão esta justamente a classe de alta coesão estrutural e conceitual (PlayVideoScreen). Por outro lado classes de baixa coesão estrutural e conceitual demandaram menos esforço de compreensão (MusicMediaAcessor), possuindo menos tempo de análise e menos número médio de erros.

Observando a opinião dos participantes, percebemos que eles também selecionaram a classe PlayVideoScreen como a mais difícil de compreender, sendo está a mais coesa. Os participantes consideraram outros fatores, diferentes da coesão, para realizar essa escolha, esses fatores representam características físicas e estruturais do código. Fatores esses, também considerados pelos participantes dos estudos anteriores.

Questão 2: Há diferenças no esforço de compreensão de classes com diferentes valores de coesão conceitual e coesão estrutural? Observando os resultados correspondentes ao segundo e terceiro cenário, onde comparamos classes com valores de coesão estrutural e conceitual variando temos que:

Quanto à coesão conceitual não observamos a influencia desse tipo de coesão no esforço de compreensão, isso é exemplificado quando observamos classes com valores distintos de coesão conceitual (PhotoViewScreen e MusicPlayController) apresentando esforços de compreensão semelhantes. Analisando esse segundo cenário notamos que a classe que exigiu o maior esforço de compreensão possuía alta coesão conceitual, e as classes que exigiram os menores esforços de compreensão possuíam: alta coesão conceitual (PhotoViewScreen) e baixa coesão conceitual (MusicPlayController).

Analisando os resultados do terceiro cenário, onde avaliamos a influência da coesão estrutural sobre o esforço de compreensão notamos também que a **coesão estrutural**

não influencia determinadamente no esforço de compreensão das classes. Percebemos nesse cenário as mesmas características do cenário anterior, existem classes com valores distintos de coesão, porém com esforços de compreensão similares (PlayVideoController e ImageMediaAcessor).

3.3.4 Conclusão

Assim como os estudos anteriores, buscamos analisar a influência da coesão sobre o esforço de compreensão. No caso desse terceiro experimento, utilizamos classes de um mesmo sistema e que, por isso, apresentavam a forma de organização do seu código fonte semelhante entre si. Ainda assim, percebemos a influência mais forte de outros fatores ligados à organização e à estrutura do código fonte sobre o esforço de compreensão.**Os resultados quantitativos e qualitativos nos levam a concluir que, mais uma vez, a coesão não influenciou determinantemente o esforço de compreensão, como acredita a literatura**.

Capítulo

4

CONCLUSÃO

Coesão é um conceito bem conhecido e usado como atributo de qualidade interna de design do software. A literatura acredita que o grau de coesão dos módulos de um sistema pode influenciar atributos de qualidade externa importantes, como manutenibilidade, facilidade de compreensão e facilidade de reutilização (Pflegger e Atlee, 2010). Contudo, quantificar a coesão não é algo trivial e, portanto, várias métricas estruturais e conceituais têm sido propostas (Chidamber e Kemerer, 1991; Chidamber e Kemerer, 1994; Bieman e Kang, 1995; Hitz e Montazeri, 1995; Henderson-Sellers et al., 1996; Martin, 1997; Briand et al., 1998; Marcus e Poshyvanyk, 2005; Újházi et al., 2010; Silva et al., 2011).

As métricas estruturais são mais conhecidas e existem em maior quantidade (Briand et al., 1998), pois determinam a coesão baseada na dependência estrutural entre os métodos, como por exemplo, o acesso à atributos em comum. Dentre as métricas de coesão estruturais podemos destacar a *Lack of Cohesion on Methods 5* - LCOM5 (Henderson-Sellers et al., 1996). Essa métrica é normalizada, ou seja, os valores variam entre [0,1] e deriva da tradicional métrica de coesão *Lack of Cohesion on Methods* – LCOM (Chidamber e Kemerer, 1994), detalhada no capítulo 2.

Por outro lado, as métricas de coesão conceituais procuram levar em consideração aspectos semânticos do código fonte, para calcular a coesão da classe (Counsell, et al., 2005; Újházi et al., 2010; Silva et al., 2012), como por exemplo, a quantidade de interesses que uma classe implementa. Podemos destacar a métrica *Lack of Concern-based Cohesion* - LCbC (Sant'Anna, 2008), também apresentada no capítulo 2.

Segundo a literatura, quanto maior for coesão, menor pode ser o esforço para se compreender um programa e vice-versa (Pflegger e Atlee, 2010). Contudo, não encontramos na literatura evidências empíricas dessa relação entre coesão e esforço de compreensão. Portanto, para avaliar em que nível o grau de coesão de módulos de sistemas de software estão relacionados ao esforço para compreender o código fonte desses módulos, essa pesquisa realizou três quase-experimentos. Nesses estudos comparamos os valores de coesão conceitual e estrutural de uma classe com o esforço de compreensão dos participantes.

Como resultado dos três experimentos, podemos dizer que não percebemos influência da coesão sobre o esforço de compreensão do código fonte. Avaliamos também a influência da coesão estrutural e conceitual individualmente e também não notamos interferências desses tipos de coesão sobre o esforço de compreensão.

Com a análise dos resultados percebemos que outros fatores apresentaram influências sobre o esforço para compreender o código fonte da classe. Esses fatores foram denominados na pesquisa como fatores de confusão (Capítulo 3) e representam características das classes, como por exemplo: domínio, nomenclatura de identificadores, tamanho, uso de bibliotecas java, dentre outros. Como esses fatores sempre estarão presentes de forma descontrolada no código fonte de sistemas reais, os resultados desses experimentos nos dão uma impressão preliminar que a coesão não tem uma influência isolada no esforço de compreensão de programas. No desenvolvimento dos estudos selecionamos as classes levando em consideração todos esses fatores, de forma que pudéssemos minimizar a influência deles sobre o esforço de compreensão. Contudo, mesmo tentando controlá-los, o esforço de compreensão de código fonte se mostrou mais sensível a variações desses fatores do que a variações nos valores das métricas de coesão.

4.1 TRABALHOS FUTUROS

A partir da pesquisa podem surgir as seguintes opções de trabalhos futuros:

• **Realizar novos estudos avaliando a relação entre o acoplamento e esforço de compreensão**: O objetivo da nossa pesquisa foi avaliar a relação entre coesão e esforço de compreensão do código fonte. Porém além da coesão outro importante atributo de *design* de software é o acoplamento. Segundo a literatura esse atributo também pode influenciar alguns atributos externos do software, como facilidade de compreensão (Pfleger and Atlee, 2010). Além disso, em nosso primeiro estudo, onde não controlamos o fator acoplamento, notamos uma forte influência desse atributo sobre o esforço de compreensão do código das classes (Capítulo 3). Portanto, sugerimos a execução de estudos seguindo uma mesma configuração dos experimentos realizados na pesquisa, contudo, variando os valores do acoplamento e controlando os valores de coesão conceitual e estrutural.

• **Utilizar outras métricas de coesão**: Como mencionado no capítulo 2 quantificar a coesão não é algo trivial e para isso métricas de coesão têm sido propostas. Essas métricas são classificadas em dois tipos: estruturais e conceituais. Na pesquisa utilizamos como métrica estrutural a LCOM5 e como métrica conceitual LCbC. Contudo, há uma diversidade de métricas presente na literatura e essas métricas calculam a coesão de diferentes maneiras. Portanto, sugerimos uma replicação dos experimentos realizados levando em consideração outras métricas de coesão e assim analisar se os resultados são semelhantes ou se há alguma métrica de coesão que possa refletir o esforço de compreensão.

• **Avaliar coesão com outros atributos externos**: Na pesquisa seguimos avaliando a relação entre a coesão e o esforço de compreensão de programas, porém a literatura acredita que a coesão pode influenciar uma diversidade de atributos externos, como: manutenibilidade, reusabilidade, propensão a mudanças, propensão a defeitos, dentre outros.

Como observamos, por meio dos resultados da pesquisa, a coesão aparentemente não influencia o esforço de compreensão como a literatura afirma, mas isso não a impede de interferir nos demais atributos externos. Portanto, sugerimos e realização de novos estudos buscando avaliar a relação entre a coesão e outro atributo externo, como os já citados anteriormente.

• **Avaliar a relação entre outros fatores e o esforço de compreensão**: Avaliamos nessa pesquisa a relação entre a coesão e o esforço de compreensão de programa, contudo, observamos que outros fatores, diferente da coesão, influenciaram o esforço de compreensão. Esses fatores são ligados à estrutura do código fonte e os denominamos de fatores de confusão. Durante os estudos tentamos controlar esses fatores, porém, ainda sim, não conseguimos minimizar totalmente a interferência deles e assim, continuaram influenciando a compreensão do código. Portanto, sugerimos novos estudos que possam avaliar a relação desses fatores de confusão com o esforço de compreensão. Como por exemplo, avaliar a relação entre o domínio (ou tamanho, ou uso de bibliotecas java, ou o uso de muitas sentenças condicionais) do software e o esforço de compreensão do código.

REFERÊNCIAS BIBLIOGRÁFICAS

Bieman, J. M. e Kang, B. –K. Cohesion e Reuse in an Object-Oriented System. *Pro. Int'l Symp. Software Reusabiliy*, v.20, pp. 259-262, 1995.

Bois, B. D.; Demeyer, S.; Verelst, J.; Mens, T.; Temmerman, M. Does God Class Decomposition Affect Comprehensibility?. *International Conference on Software Engineering – IASTED*, 2006.

Briand, L. C.; Daly, J. W.; Wust, J. K. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3(1), pp. 65-117, 1998.

Chidamber, S. e Kemerer, C. Towards a Metrics Suite for Object Oriented Design. *Object-oriented Programming Systems, Languages e Applications – OOPSLA'91*, ACM, 26(11), pp. 197-211, 1991.

Chidamber, S. e Kemerer, C. A Metric Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, 1994.

Counsell, S.; Swift, S.; Tucker, A.; Mendes, E.; Object-oriented cohesion as a surrogate of software comprehension: an empirical study. *5th IEEE International Workshop on Source Code Analysis e Manipulation (SCAM'05)*, Hungary, pp. 161-169, 2005.

Dunsmore, A. e Roper, M. A Comparative Evaluation of Program Comprehension Measures. *EFoCS SEG Technical Reports*, 2000.

Eaddy, M.; Sherwood, K. D.; Garg, V.; Murphy, G. C.; Nagappan, N. Do Crosscutting Concerns Cause Defects?. *IEEE Transactions on Software Engineering*, 2008.

Eder, J.; Kappel, G.; Schrefl, M. Coupling e Cohesion in Object-Oriented Systems. *Technical Report. University of Klagenfurt*, 1994.

Eisenbarth, T.; Koschke, R.; Simon, D.; Aiding Program Comprehension by Static e Dynamic Feature Analysis. *IEEE International Conference on Software Maintenance*, pp.602-611, 2001.

Feigenspan, J.; Apel, S.; Liebig, J.; Kastner, C. Exploring Software Measures to Assess Program Comprehension. *IEEE International Symposium on Empirical Software Engi-*

neering and Measurement (ESEM), 2011.

Fenton, N. e Peleeger, S. Software Metrics: A Rigorous e Practical Approach. 2.ed. London: PWS, 1997.

Ferreira, K. A. M.; Bigonha, M. A. S.; Bigonha, R. S.; Almeida, H. C.; Neves, R. C.das. Métrica de Coesão de Responsabilidade – A Utilidade de Métricas de Coesão na Identificação de Classes com Problemas Estruturais. *Simpósio Brasileiro de Qualidade de Software – SBQS*, 2011. Curitiba - Paraná, 2011.

Henderson-Sellers, B.; Constantine, L. L.; Graham, I. M. Coupling e Cohesion (Towards a valid metrics suite for object-oriented analysis e design). Object Oriented Systems, v.3, pp. 143-158, 1996.

Hitz, M. e Montazeri, B. Measuring Coupling e Cohesion in Object-Oriented System. *Proc. 3rd Intl. Symp. Applied Corporate Computing.* Oct. 1995.

Li, W. e Henry, S. Object-Oriented Metrics that Predict Maintainability. J. Systems e Software, vol. 23, 1993.

Liu, Y.; Poshyvanyk, D.; Ferenc, R.; Gyimothy, T.; Chrisochoides, N. Modeling class cohesion as mixtures of latent topics. *International Conference on Software Maintenance (ICSM2009)*, September, Edmonton, Canada. p. 233–242, 2009.

Marcus, A. e Poshyvanyk, D. The Conceptual Cohesion of Classes. *IEEE International Conference on Software Maintenance (ICSM '05).* Washington - DC, pp. 133-142, 2005.

Martin, R. Stability. C++ Report, 1997.

Nishizono, K.; Morisaki, S.; Vivanco, R.; Matsumoto, K. Source code comprehension strategies and metrics to predict comprehension effort in software maintenance and evolution tasks – an empirical study with industry practitioners. *27th IEEE International Conference on Software Maintenance – ICSM*, Williamsburg, pp. 473-481, 2011.

Ostermann, K.; Giarrusso, P. G.; Kastner, C.; Rendel, T. Revisiting information hiding: reflections on classical and nonclassical modularity. *Proceedings of the 25th European conference on Object-oriented programming - ECOOP' 11*, pp. 155-178, 2011.

Parnas, D.L. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, Vol. 15, No. 12, pp. 1053 – 1058, 1972.

Pfleeger, S. e Atlee, J. Software Engineering: theory and practice. 4th ed, Prentice Hall, 2010.

Robillard, M. P. e Murphy, G. C. Representing concerns in source code. *ACM Transactions on Software Engineering e Methodology – TOSEM*, 2007.

Robillard, M. e Weigand-Warr, F. ConcernMapper: simple view-based separation of scattered concerns. *OOPSLA Workshop on Eclipse technology eXchange*. ACM, 2005.

Rugaber, S. Program Comprehension. *Georgia Institute of Technology*. May 4, 1995.

Sant' Anna, C. N. (Tese de Doutorado em Informática). On the Modularity of Aspect-Oriented Design: A concern-drive measurement approach. Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2008.

Siegmund, J.; Kastner, C.; Apel, S.; Brechmann, A.; Saake, G. Experience from Measuring Program Comprehension – Toward a General Framework. *Multikonferenz Software Engineering*. Aachen, pp. 239-257, 2013.

Silva, B.; Sant' Anna, C.; Chavez, C. Concern-Based Cohesion as Change Proneness Indicator: An Initial Empirical Study. *WETSoM'11 ACM*, 2011.

Silva, B.; Sant' Anna, C.; Chavez, C.; Garcia, A. Concern-Based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement. *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC 2012)*, Passau, Germany, pp. 103-112, 2012.

Silva, B.; Sant' Anna, C.; Chavez, C. An Empirical Study on How Developers Reason about Module Cohesion. *13th International Conference on Modularity (Modularity 2014)*, Lugano, pp. 121-132, 2014.

Sommerville, I. Software Engineering, 6th ed: Addison-Wesley Longman, 2001.

Stevens, W.; Myers, G.; Constantine, L. Structured Design. IBM Systems, vol.13, n.2, 1974.

Újházi, B.; Ferenc, R.; Poshyvanyk, D.; Gyimóthy, T. New Conceptual Coupling e Cohesion Metrics for Object-Oriented Systems. *Working Conference on Source Code Analysis e Manipulation, 2010*. IEEE Computer Society, pp. 33-42, 2010.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A. Experimentation in Software Engineering. Springer Publishing Company, Berlin, 2012.

Young, T. (Dissertação de Mestrado) Using AspectJ to Build a Software Product Line for Mobile Devices. Univ. of British Columbia, 2005.

APÊNDICE A

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Título do Estudo: Estudo experimental da relação entre o grau de coesão e a compreensão do código fonte de uma classe.

Pesquisadores responsáveis: Elienai Bitencourt Batista e Cláudio Nogueira Sant'Anna.

Prezado (a) senhor (a),

Eu sou Elienai Bitencourt Batista, estudante do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia e estou realizando, um experimento, como parte das atividades da disciplina Tópicos em Engenharia de Software IV sob supervisão do professor Cláudio Nogueira Sant'Anna. O objetivo geral deste experimento é avaliar de que forma a relação entre o grau de coesão de uma classe está relacionado com a compreensão dessa classe.

Gostaria de convidar o(a) senhor(a) para colaborar como um participante desse experimento controlado, para que seja possível alcançar os objetivos estabelecidos. Sua participação é voluntária e consistirá no preenchimento de um questionário de caracterização, anexado a esse termo, que visa coletar informações sobre sua experiência profissional e informações técnicas. Além disso, será também solicitado sua participação no dia da execução do experimento e o preenchimento de um questionário de feedback.

Se alguma pergunta dos questionários lhe causar desconforto, o(a) senhor(a) poderá alertar os responsáveis do experimento. Ao participar desta pesquisa, o(a) senhor(a), não será identificado(a), permanecendo em anonimato e poderá retirar seu consentimento em qualquer momento, sem sofrer nenhum dano.

O experimento também não traz nenhum prejuízo financeiro para o senhor(a), nem qualquer forma de ressarcimento ou indenização financeira por sua participação. Caso não deseje participar sua vontade será respeitada.

O(A) senhor(a) pode solicitar esclarecimento antes, durante e depois da participação no experimento. Tais esclarecimentos podem ser obtidos com Elienai Bitencourt Batista através do e-mail elienaibittencourt@gmail.com ou com o professor Cláudio Nogueira Sant'Anna através do e-mail santanna@dcc.ufba.br. Se o(a) senhor(a) aceitar participar desta pesquisa, precisará assinar o Termo de Consentimento Livre e Esclarecido – TCLE e entregar aos pesquisadores responsáveis.

Desde já agradeço a sua atenção e colaboração!

Salvador – BA.

Assinatura do Participante

APÊNDICE B

QUESTIONÁRIO DE CARACTERIZAÇÃO

QUESTIONÁRIO DE CARACTERIZAÇÃO

Informações Pessoais e Informações Técnicas

1. Qual seu nome?————————————————

2. Qual a sua idade? ————- anos.

3. Email: ————————————————-

4. Qual seu grau de instrução?

a. Superior incompleto
b. Superior completo
c. Mestrado incompleto
d. Mestrado completo
e. Doutorado incompleto
f. Doutorado completo

Definir curso: ————————————————-

Experiência Profissional

1. Você trabalha ou já trabalhou com desenvolvimento de software (exceto nas disciplinas do seu curso)?

a. Sim
b. Não

Se você respondeu SIM na questão anterior, responda:
Por quanto tempo? ————————————
Qual era a sua função no processo de desenvolvimento?
———————————————————————————————————

2. Classifique seu conhecimento sobre Programação Orientada a Objeto?

1. Expert
2. Bom
3. Razoável
4. Pouco

5. Nenhum

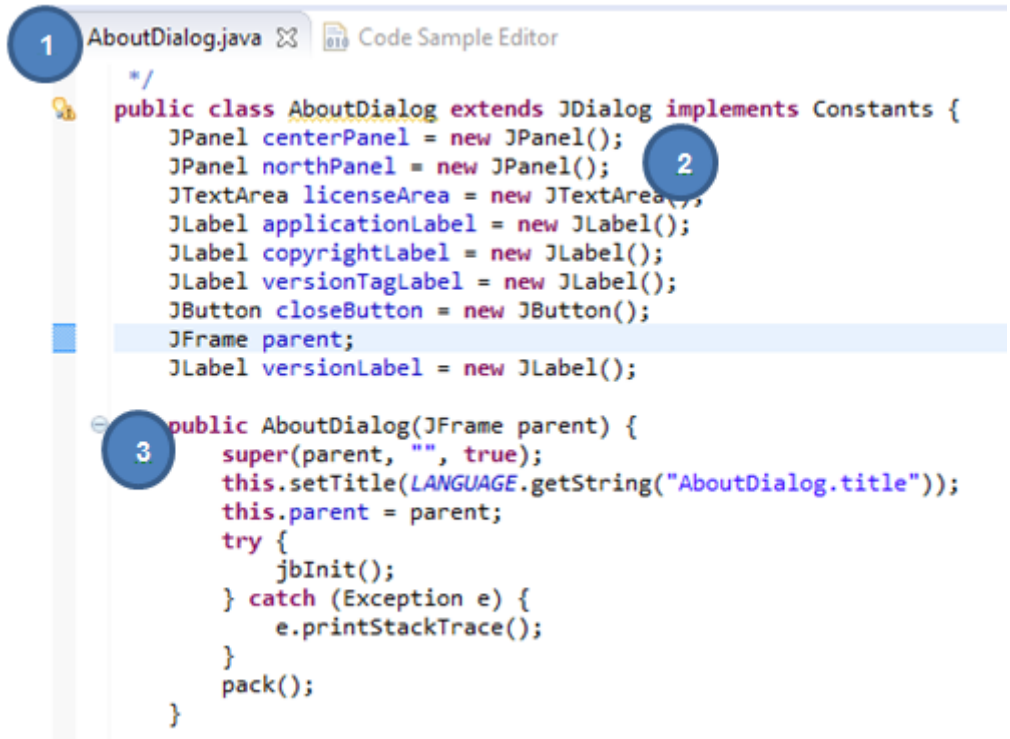
3. Há quanto tempo você programa (profissionalmente ou não):

1. Menos de 2 anos
2. De 2 a 3 anos
3. De 3 a 4 anos
4. Mais de 5 anos

4. Você tem dificuldade para identificar alguns componentes como: classes, métodos e atributos em um código?

1. Não
2. Às vezes
3. Tenho
4. Não sei o que são classes, métodos e atributos.

Se você respondeu a questão anterior com: 1, 2 ou 3, identifique o que os números abaixo representam (classe, método ou atributo):



APÊNDICE C

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO I

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO I

Hora de Início: ———— Hora de Término: ————

CLASSE: **BibliotecaUI2**

1. Complete as linhas em branco do código (Linha 59 e 152):

a. Linha 59, método consultarLivros():

```
if (_____) {
    System.out.println("Nao ha' nenhum livro cadastrado.");
}
```

b. Linha 152, método ConsultarUsuarios():

```
if (_____) {
    System.out.println("Nao ha' nenhum usuario cadastrado.");
}
```

2. No método excluirUsuario() o que acontecerá se o idUsuario digitado for -5?

_____

_____

3. No método concultarEmprestimo(), se o valor do ISBN do livro digitado pelo usuário for: "não sei", e caso, não exista nenhum empréstimo cujo livro apresente esse valor de ISBN. Qual será o valor da variável empréstimo e qual será a mensagem apresentada na tela para o usuário?

_____

_____

_____

4. Encontre qual linha do código corresponde à adição de um novo livro no sistema. Escreva abaixo a linha de código correspondente.

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Person2**

1. Complete a linha em branco do código (Linha 97):

```
private void removeRelationFromVector(Relation relation)
{
        ____________________________________________
}
```

2. No método addSpouse() considera a adição de:
a. Somente esposa
b. Somente esposo
c. Esposa e Esposo
d. Nenhuma das Anteriores

3. No método illeggalRelation() existem 4 condições que caracterizam uma relação como ilegal. Quais são essas condições?

4. Encontre a linha do código que adiciona a relação "RelationParentChild" no vetor correspondente. Escreva a linha de código correspondente abaixo:

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Locadora2**

1. Complete as linhas em branco do código (Linha 121 e 137):

a. Linha 121, método CadastrarCliente():

```
if (_____)
 {
    Cliente c = new Cliente();
    System.out.println("Informe o nome do cliente:");
    Scanner scan = new Scanner(System.in);
    c.setNome(scan.nextLine());
    c.setCod(GerarCodCliente());
    clientes[i] = c;
    System.out.println("Cliente cadastrado com    sucesso!!!");
    ImprimirCliente(clientes[i]);
 }
```

b. Linha 137, método CadastrarFilme():

```
if (_____)
 {
        Filme f = new Filme();
        System.out.println("Informe o nome do filme:");
        Scanner scan = new Scanner(System.in);
        f.setNome(scan.nextLine());
        f.setCod(GerarCodFilme());
        filmes[i] = f;
        System.out.println("Filme cadastrado com    sucesso!!!");
        ImprimirFilme(filmes[i]);
 }
```

2. Qual mensagem será retornar para usuário que seguir o seguinte fluxo de entrada para os menus?
Digita 1, depois digita 2, depois digita -1:

_____

_____

3. Semelhante à questão anterior, determine as entradas necessárias para que o usuário cadastre um filme e depois imprima todos os filmes cadastrados:

_____

_____

4. Localize a linha de código responsável pela exclusão de um filme. Escreva abaixo a linha de código correspondente:

_____

_____

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Contatos**

1. Complete a linha em branco do código (Linha 34):

```
do {
        op = Integer.parseInt(JOptionPane.showInputDialog(null,"Deseja
    efetuar outro  cadastro?\n\n1. Sim\n2. Não\n\nDigite sua opção:"));
} while(_____);
```

2. Se no momento de alterar um contato, o usuário digitar o mesmo número de telefone que o contato já possuía, alguma mensagem aparecerá para o usuário? Se sim, qual será a mensagem?

_____
_____

3. No momento do cadastro se o usuário digitar um número de telefone que já existe entre os contatos, o que acontecerá?

_____
_____
_____

4. Localize as linhas de código que são responsáveis por:
a. Exclusão do contato: ————————————————————————-
b. Adição dos dados alterados: ————————————————————————-

Observação: Não se esqueça de preencher o tempo inicial e final

APÊNDICE D

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO II

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO II

Hora de Início: ———— Hora de Término: ————

CLASSE: **RepositorioCliente**

1. Complete a linha em branco do código (Linha 115):

```
public Cliente buscarPorCpf(String cpf) throws Exception {

        Cliente cliente = null;

        if (_____________________________) {

              PreparedStatement pstm = null;
              ResultSet rs = null;
        .
        .
        .
 }
```

2. Implemente um método pata listar os clientes pelo nome, baseado nos métodos existentes.

3. De acordo com o código, qual ou quais atributos do cliente podem ser atualizado(s)?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **ContatoDAO**

1. Complete a linha em branco do código (Linha 64):

```
public Contato listaNome(Contato contato){
      try {
            PreparedStatement          preparedStatement          =
            conexao.prepareStatement(PorNome);

            preparedStatement.setString(_____________________);
            ResultSet resultSet = preparedStatement.executeQuery();
            Contato contato = new Contato();
            resultSet.next();
            contato = dadosBanco(resultSet);
            resultSet.close();
            preparedStatement.close();
            return contato;

      } catch (SQLException e) {
            throw new RuntimeException(e);
      }
}
```

2. Implemente um método para pesquisar contato pelo nome, baseado nos método existentes.

3. De acordo com o código, ao atualizar um contato é também possível atualizar o id do contato?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **PassagemControle**

1. Complete a linha em branco do código (Linha 47):

```
public void preencherVoo(){
        if (!this.validaDataSalvar()){
                return;
        }

        this.voosDisponiveis                                                    =
        vooService.consultarVooDisponiveis(this.passagem.getDataIda(),
        this.passagem.getOrigem(), this.passagem.getDestino());

        if (_____________________________________________){
                FacesContextUtil.addMessage(Severity.WARN,    "Não    há    voos
                disponíveis de Ida para esta data.", null, true);
        }
        .
        .
        .
}
```

2. Implemente mais uma condição no método validaDataSalvar(), baseado nos existentes, verificando se foi preenchido o campo "tipo passagem" e retornando uma mensagem para o usuário , caso o campo não tenha sido preenchido.

_____
_____
_____
_____
_____

3. De acordo com o código, existe diferença nos valores de passagem de criança e de adulto? Se sim, qual é a diferença?

_____
_____
_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Locadora**

1. Complete a linha em branco do código (Linha 78):

```
public static int BuscaClientePorCodigo(int cod){
        boolean achou = false;
        int i = 0;
        while(achou == false && i<20){
            if (clientes[i]!= null){
                if (_____________________________){
                    achou = true;
                }else{
                    i++;
                }
            }else{
                i++;
            }
        }
        if(achou){
            return i;
        }else{
            return -1;
        }
    }
```

2. Implemente o método BuscarFilmePorCodigo(int cod) baseado nos métodos existentes.

3. De acordo com o código, qual mensagem será retornada para usuário que seguir o seguinte fluxo de entrada para os menus? Digita 1, depois digita 2, depois digita -1:

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Compromisso**

1. Complete a linha em branco do código (Linha 113):

```
if (naoHoje) {

   if (_____________________________) {

       enviarEmailCompromisso(Principal.dados.getEmail(),"Voce   perdeu
       um    compromisso...    -Equipe    Nortev    Solutions","Caro(a)
       "+Principal.dados.getNome()+",\nNo                          dia:
       "+sdfBR.format(dataCompromisso)+",      voce      tinha      um
       compromisso.\n\nLocal:         "+notificacao.getLocal()+"\nHora:
       "+notificacao.getHora()+"\nAssunto:
       "+notificacao.getDetalhes());

       notificacao.setFuiNotificadoDia(true);
       try {
              compromissoDao.atualizar(notificacao);
       } catch (Exception e) {
       System.out.println("erro");
       e.printStackTrace();
       }
       compromissosRemovidos.add(notificacao);
   }
}
```

2. Implemente baseado nas condições existente, uma nova condição que envie um email para avisar de um compromisso que acontecerá na data de hoje.

3. Qual mensagem o usuário receberá se existir a seguinte situação?
Data de hoje = 25/02/2015 Horário: 15:00hs
Data do compromisso = 24/02/2015 Horário: 15:00hs

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **BombaDeInsulina**

1. Complete a linha em branco do código (Linha 22):

```
for (_____________________________________)
    System.out.println (" leitura = " + i + " " + Leituras [i] );

DoseAtual = CalcularInsulina();

System.out.println ("Calculo de Insulina retorna " + DoseAtual) ;
```

2. Implemente dentro do método main() uma alteração, que conte quantas vezes a DoseAtual foi maior que DoseMaxima e imprima essa quantidade no final.

3. Se existir as seguintes leituras do nível de açucar:
Leituras[0] = 6
Leituras[1] =7
Leituras [2] = 10
Qual será o valor da dose de insulina atual?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **RepositorioFuncionario**

1. Complete a linha em branco do código (Linha 84):

```
public void removerFuncionario(String cpf) throws Exception {

      if (_____________________________) {

            PreparedStatement preparedStatement= null;
            try {
             try {
                   String sql;
                   sql = "DELETE FROM Funcionario WHERE fun_cpf=?";
                         preparedStatement=
                         this.conexao.prepareStatement(sql);
                         preparedStatement.setString(1, cpf);
                         preparedStatement.executeUpdate();
             } finally {
                   if (preparedStatement!= null) {
                   pstm.close();
             }
           }
           } catch (SQLException e) {
             e.printStackTrace();
             throw new Exception(e);
           }
      }
}
```

2. Implemente um método para atualizar o CTPS dos funcionários que tenham salários maiores que 1000.

_____

_____

_____

_____

3. No método listarFuniconarioPorNome(String nome) a lista dos funcionários apresentada será somente com os nomes dos funcionários ou além dos nomes essa lista também apresentará os demais atributos do funcionário? Justifique

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Droga**

1. Complete a linha em branco do código (Linha 53):

```
public void deletarDroga(int IdDroga) {
      try {
            PreparedStatement          preparedStatement          =
            conexao.prepareStatement("DELETE    FROM    drug    WHERE
            drugId=?");
            ps._____________________________;
            ps.executeUpdate();
            ps.close();
      } catch (SQLException ex) {

      Logger.getLogger(Droga.class.getName()).log(Level.SEVERE,   null,
      ex);
      }
}
```

2. Implemente um método para pesquisar a Droga somente pelo nome, baseado no método de busca existente.

_____
_____
_____
_____
_____
_____

3. Quais são os atributos chaves para realizar a busca de Droga nos métodos: EditarDroga, AdicionarDroga, AtualizarDroga e DeletarDroga?

_____
_____
_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Jogo**

1. Complete a linha em branco do código (Linha 62):

```
public boolean jogoAcabou () {
	if ((this.erros == 5) || (this.acertos == _____)
	{
		return true;
	}
	else {
		return false;
	}
}
```

2. Implemente um método para imprimir cada caractere da string segredo, juntamente com a posição do caractere.

3. Se o usuário cometer 3 erros, o jogo acaba? Qual jogo está sendo implementado?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **Sala**

1. Complete a linha em branco do código (Linha 117):

```
public Double getTemperaturaInterna(){

       Double tempExterna = 0.0;

       for(int k=0;_____________________ ;k++)
        tempExterna = tempExterna + sensoresInternos.get(k).getValor();
       return tempExterna/sensoresInternos.size();
}
```

2. Implemente um método que imprima todos os valores do sensor externo e a posição correspondente de cada valor.

3. De acordo com o método getTemperaturaExterna(), qual será o valor retornado se existir um sensor externo com 3 valores, a saber: 22, 18, 25?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **RepositorioEndereco**

1. Complete a linha em branco do código (Linha 117):

```
public Endereco buscarPorIdCliente(Integer idCliente) throws Exception
{
	Endereco endereco = null;
	if (_____________________________) {

		String sql = "SELECT * FROM Endereco WHERE id_cli_fk=?";
		PreparedStatement preparedStatement  = null;
		ResultSet resultSet  = null;
	.
	.
	.
	.
}
```

2. Implemente um método que retorne os endereços cujo CEP seja igual a 45345000.

3. De acordo com o código, para cadastrar um endereço é necessário existir um cliente cadastrado? Justifique ?

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **ControleConvenio**

1. Complete a linha em branco do código (Linha 16):

```
public boolean existeConvenio(Convenio convenio) {
	boolean existe = false;
	Collection ConvenioColecao;
	try {
		ConvenioColecao = obterConvenio(convenio);
		if (_____________________________________)
			if (ConvenioColecao.size()>0) {
				existe=true;
			}
	} catch (Exception e) {
		e.printStackTrace();
	}
	return existe;
}
```

2. Implemente um método que retorne os convênios que tenha como médico Rogério Costa ou número 7.

3. Ao modificar convenio quais atributos podem ser alterados?

Observação: Não se esqueça de preencher o tempo inicial e final

APÊNDICE E

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO III

QUESTIONÁRIOS DE ATIVIDADES EXPERIMENTO III

Hora de Início: ———— Hora de Término: ————

CLASSE: **PlayVideoScreen**

1. Complete a linha em branco do código (Linha 89):

```
public void stopVideo() {
        try {
        if(_____________________________)
        {
                player.stop();
        }
        } catch(Exception e) {e.printStackTrace(); }
}
```

2. Quais são os comandos disponíveis para controle do vídeo?

_____

_____

3. De acordo com o código é possível realizar o "pause" no vídeo? Justifique

_____

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **PlayVideoController**

1. Complete a linha em branco do código (Linha 56):

```
else if (label.equals(_____________________)) {

AddMediaToAlbum copyPhotoToAlbum = new AddMediaToAlbum("Copy
Media to Album");

copyVideoToAlbum.setItemName(mediaName);
copyVideoToAlbum.setLabePath("Copy to Album:");
copyVideoToAlbum.setCommandListener(this);
Display.getDisplay(midlet).setCurrent(copyVideoToAlbum);

return true;
```

2. Existem algumas condições que impedem de salvar o vídeo no álbum, quais são essas condições?

_____
_____
_____
_____

3. Quais são os atributos exigidos ao adicionar o vídeo ao álbum?

_____
_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **ImageMediaAccessor**

1. Complete a linha em branco do código (Linha 69):

```
public void addImageData(String photoname, byte[] imgdata, String    albumname)
                throws InvalidImageDataException, PersistenceMechanismException {
                try {
                addMediaArrayOfBytes(_____________________________________);
                        } catch (RecordStoreException e) {
                        throw new PersistenceMechanismException();
                }
}
```

2. O que acontece quando a linha de código "removeRecords()" é executada? (linha 23)

_____

_____

3. Ao adicionar uma imagem no álbum quais atributos são exigidos?

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **PhotoViewScreen**

1. Complete a linha em branco do código (Linha 32):

```
public PhotoViewScreen(Image img) {

        image = img;
        this.addCommand(backCommand);
        this.addCommand(copyCommand);
        this.addCommand(_____________________________);
}
```

2. De acordo com o método "paint(Graphics g)" o desenho da imagem é iniciado em qual parte da tela? Justifique.

_____

_____

3. Se o objeto imagem for "null" o que será desenhado na tela? Justifique.

_____

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **MusicPlayController**

1. Complete a linha em branco do código (Linha 56):

```
    else if (label.equals(_____________________)) {
    AddMediaToAlbum copyMusicAlbum = new AddMediaToAlbum("Copy Media to
Album");
          copyMusicToAlbum.setItemName(mediaName);
          copyMusicToAlbum.setLabePath("Copy to Album:");
          copyMusicToAlbum.setCommandListener(this);
```

2. Existem algumas condições que impedem de salvar a música no álbum, quais são essas condições?

_____
_____
_____
_____

3. De acordo com o código, quais são os comandos existentes para controle da música?

_____
_____

Observação: Não se esqueça de preencher o tempo inicial e final

QUESTIONÁRIO DE ATIVIDADES

Hora de Início: ———— Hora de Término: ————

CLASSE: **MusicMediaAccessor**

1. Complete a linha em branco do código (Linha 36):

```
protected MediaData getMediaFromBytes(byte[] data) throws InvalidArrayFormatException {
        MediaData iiObject converter.getMultiMediaInfoFromBytes(_____);
                return iiObject;
}
```

2. Quais atributos são exigidos para adicionar uma música no álbum?

_____

_____

3. Antes das músicas de "teste" serem adicionadas no álbum o que é feito?

_____

_____

Observação: Não se esqueça de preencher o tempo inicial e final

APÊNDICE F

QUESTIONÁRIO DE FEEDBACK EXPERIMENTO I

QUESTIONÁRIO DE FEEDBACK

1. Qual foi o seu nível de dificuldade em responder as questões do experimento?

1. Baixo
2. Médio
3. Razoavelmente alto
4. Alto
5. Não tive dificuldade

2. Qual das classes apresentada você teve mais dificuldade de compreender?

1. Person2
2. Locadora2
3. BibliotecaUI2
4. Contatos

3. As possíveis limitações, que você encontrou na realização do experimento, são causadas por (Pode marcar mais de uma alternativa):

1. Falta de comentários nas classes
2. Classes com domínio desconhecido
3. Classes longas demais
4. Nomenclatura dos identificadores desapropriada
5. Número muito extenso de perguntas
6. Outros. Quais: ———————————————————

4. Em sua opinião as perguntas realizadas no experimento foram (Pode marcar mais de uma alternativa):

1. Claras e de fácil entendimento
2. Difíceis de compreender
3. Possuíam interpretação ambígua
4. Objetivas
5. Subjetivas
6. Possuíam fácil interpretação

5. Você conseguiu responder todas as questões?

1. Sim
2. Não. Por quê?———————————————————

APÊNDICE G

QUESTIONÁRIO DE FEEDBACK EXPERIMENTO II

QUESTIONÁRIO DE FEEDBACK - EXPERIMENTO II

1. Qual foi o seu nível de dificuldade em responder as questões do experimento?

1. Baixo
2. Médio
3. Razoavelmente alto
4. Alto
5. Não tive dificuldade

2. De acordo com sua análise ordene as classes da mais fácil para a mais difícil (1° classe = a mais fácil de compreender/ última classe = a mais difícil de compreender).

_____
_____
_____
_____
_____
_____

3. Cite quais critérios você utilizou para ordenar as classes:

_____
_____
_____
_____

4. As possíveis limitações, que você encontrou na realização do experimento, são causadas por (Pode marcar mais de uma alternativa):

1. Falta de comentários nas classes
2. Classes com domínio desconhecido
3. Classes longas demais
4. Nomenclatura dos identificadores desapropriada
5. Número muito extenso de perguntas
6. Outros. Quais: _____

5. Em sua opinião as perguntas realizadas no experimento foram (Pode marcar mais de uma alternativa):

1. Claras e de fácil entendimento
2. Difíceis de compreender
3. Possuíam interpretação ambígua

4. Objetivas
5. Subjetivas
6. Possuíam fácil interpretação

6. Você conseguiu responder todas as questões?

1. Sim
2. Não. Por quê?———————————————————

APÊNDICE H

QUESTIONÁRIO DE FEEDBACK EXPERIMENTO III

QUESTIONÁRIO DE FEEDBACK - EXPERIMENTO III

1. Qual foi o seu nível de dificuldade em responder as questões do experimento?

1. Baixo
2. Médio
3. Razoavelmente alto
4. Alto
5. Não tive dificuldade

2. De acordo com sua análise determine a classe mais faz de compreender e a mais difícil de ser compreendida, de acordo com sua opinião (Se quiser pode citar mais de uma).

_____ = MAIS FÁCIL
_____ = MAIS DIFÍCIL

3. Cite quais critérios você utilizou para determinar a classe mais fácil e mais difícil de compreender:

_____
_____
_____
_____

4. Para responder as questões você consultou somente o código da classe correspondente ou fez consulta ao código de outras classes? Cite o nome das classes consultadas.

_____
_____
_____

5. As possíveis limitações, que você encontrou na realização do experimento, são causadas por (Pode marcar mais de uma alternativa):

1. Falta de comentários nas classes
2. Classes com domínio desconhecido
3. Classes longas demais
4. Nomenclatura dos identificadores desapropriada
5. Número muito extenso de perguntas
6. Outros. Quais: _____

6. Em sua opinião as perguntas realizadas no experimento foram (Pode marcar mais de uma alternativa):

1. Claras e de fácil entendimento
2. Difíceis de compreender
3. Possuíam interpretação ambígua
4. Objetivas
5. Subjetivas
6. Possuíam fácil interpretação

7. Você conseguiu responder todas as questões?

1. Sim
2. Não. Por quê?_____