Universidade Federal da Bahia
Instituto de Matemática

Programa de Pós-Graduação em Ciência da Computação

INTEGRAÇÃO DE PRÁTICAS DO DDM AO PROCESSO ÁGIL AVALIANDO ASPECTOS DE EVOLUÇÃO DE SOFTWARE.

Elton Figueiredo da Silva

DISSERTAÇÃO DE MESTRADO

Salvador
06 de Agosto de 2019

ELTON FIGUEIREDO DA SILVA

INTEGRAÇÃO DE PRÁTICAS DO DDM AO PROCESSO ÁGIL AVALIANDO ASPECTOS DE EVOLUÇÃO DE SOFTWARE.

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.

Orientadora: Prof. Dr. Rita Suzana Pitangueira Maciel
Co-orientadora: Prof. Dr. Ana Patrícia Fontes Magalhães Mascarenhas

Salvador
06 de Agosto de 2019

Sistema de Bibliotecas - UFBA

Silva, Elton Figueiredo (usado em CITACOES).

Integração de práticas do DDM ao processo ágil avaliando aspectos de evolução de software. / Elton Figueiredo da Silva – Salvador, 2019.

124p.: il.

Orientadora: Prof. Dr. Prof. Dr. Rita Suzana Pitangueira Maciel.

Co-orientadora: Prof. Dr. Prof. Dr. Ana Patrícia Fontes Magalhães Mascarenhas.

Dissertação (Mestrado) – Universidade Federal da Bahia, Instituto de Matemática, 2019.

1. Primeira palavra-chave. 2. Segunda palavra-chave. 3. Terceira palavra-chave. I. Maciel, Rita Suzana Pitangueira . II. Mascarenhas, Ana Patrícia Fontes Magalhães . III. Universidade Federal da Bahia. Instituto de Matemática. IV. Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

Dedico esta dissertação à minha mãe Solange, que dignamente me apresentou à importância da família e ao caminho da honestidade e persistência.

A Marcos André pelo apoio incondicional em todos os momentos, principalmente nos de incerteza, muito comuns para quem tenta trilhar novos caminhos.

AGRADECIMENTOS

Agradeço em primeiro lugar à Deus que iluminou o meu caminho durante esta caminhada.

Agradeço a minha professora e orientadora Dra. Rita Suzana Pitangueira Maciel, o meu reconhecimento pela oportunidade de realizar este trabalho ao lado de alguém que transpira sabedoria; meu respeito e admiração pela sua serenidade, capacidade de análise do perfil de seus alunos, e pelo seu dom no ensino, inibindo sempre a vaidade em prol da simplicidade e eficiência.

Agradeço a minha professora co-orientadora Dra. Ana Patricia Magalhães que com bastante paciência me ajudou na conclusão este trabalho.

Agradeço a todos os participantes deste estudo, pela disposição em ajudar no que deles dependesse para a conclusão deste trabalho.

Agradeço aos amigos do Mestrado que compartilharam comigo esses momentos de aprendizado.

Agradeço também aos meus amigos que não ficavam perguntando sobre o prazo de conclusão do mestrado.

"O SENHOR firma os passos de todo aquele cuja conduta lhe agrada!"
Salmos 37:23

—

RESUMO

Em desenvolvimento de software que utiliza algum método ágil, o artefato principal e mais atualizado é o código fonte em detrimento à documentação. Contrapondo-se a esta realidade, a abordagem de Desenvolvimento Dirigido por Modelo (DDM) tem o modelo como principal artefato do processo de desenvolvimento. Objetivando extrair as vantagens de ambas as abordagens, (SALES, 2017) desenvolveu o metaprocesso (MDD ágil) ScrumDDM que adicionou práticas de modelagem e transformação de modelos oriundas do desenvolvimento dirigido por modelo ao framework Scrum. Diferenciando do trabalho de (SALES, 2017), neste trabalho será investigado se a adição de práticas DDM ao framework Scrum antem a agilidade do processo e se foi possível instanciar outro processo com a utilização do metaprocesso ScrumDDM, este novo processo continuou sendo ágil e se foi possível evoluir um software existente através das estórias dos usuários e da documentação provida pelo ScrumDDM, bem como se este metaprocesso é efetivo na instanciação de novos processos que integrando-os ao SCRUM e ao DDM. Para avaliar o metaprocesso quanto a capacidade de evoluir o software a partir das estórias dos usuários e a agilidade no desenvolvimento do software, foi desenvolvido um experimento controlado. Para avaliar a generalização do metaprocesso, por sua vez, um novo processo de software oriundo da literatura acadêmica foi instanciado a partir deste metaprocesso. No experimento controlado desenvolvido neste trabalho, foi possível demostrar que o metaprocesso avaliado apoiou a evolução do software através da documentação oriunda do ScrumDDM. Também demostrou que a agilidade do desenvolvimento foi aperfeiçoada através das transformações e dos modelos da abordagem DDM. Além dos resultados citados, ao final do processo de desenvolvimento, observou-se que, tanto o código do projeto quanto a documentação do software estavam atualizados de acordo com a especificação realizada no inicio do projeto, evidenciando assim, que a inclusão de praticas de DDM no processo ágil pode ser uma alternativa de hibridização viável para aumentar a produtividade sem abrir mão da documentação do projeto.

Palavras-chave: Metodologia ágil, Evolução de Software, Evolução de Estórias dos Usuários, Atributos de Qualidade em Código Fonte, Processo de Software, Desenvolvimento Dirigido por Modelo, Integração DDM e SCRUM, Hibridização de abordagens.

ABSTRACT

In software development that uses some agile method, the source code, rather than documentation, is the main and most up-to-date artifact. In contrast to this reality, the Model Driven Development (DDM) approach has the model as the main artifact of the development process. Intending to get th advantages of both approaches, (SALES, 2017) developed the ScrumDDM meta-process (agile MDD) that added modeling and model transformation practices coming from model driven development to the Scrum framework. Unlike the work of (SALES, 2017), this paper will investigate whether the ScrumDDM metaprocess preserves the characteristic agility of SCRUM, supports software evolution through user stories and documentation provided by DDM using for that aexisting software and developed by ScrumDDM, as well as whether this metaprocess is effective in creating new processes that integrate SCRUM and DDM. To evaluate the metaprocess regarding its ability to evolve user stories and the agility in software development, a controlled experiment was developed. To evaluate the generalization of metaprocess, in turn, a new software process from the academic literature was instantiated from this metaprocess. Through the controlled experiment developed in this work, it was possible to demonstrate that the evaluated metaprocess supported the software evolution through DDM documentation. It has also been shown that development agility has been enhanced through the transformations and models of the DDM approach. In addition to the results cited, at the end of the development process, it was observed that, both the project code and documentation, were up to date.

Keywords: Agile Methodology, Metamodeling in Agile Methods, Evolution of Software, Evolution of User Stories, Quality Attributes in Source Code, Software Process Model, Model Driven Development, Metaprocessing and Metamodeling, DDM and SCRUM Integration.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

AM - Agile Modeling
ASD - Adaptive Software Development
CVDS - Ciclo de Vida do Software
DAS - Desenvolvimento Ágil de Software
DDM - Desenvolvimento Dirigido por Modelos
DDT - Desenvolvimento Dirigido por Teste
DSDM - Dynamic Systems Development Method
EST - Engenharia de Software Tradicional
FDD - Feature Driven Development
JAD - Joint Application Design
LSD - Lean Systems Development
MBT - ModelBased Test (Teste Baseado por Modelos)
MDA - Model Driven Architecture
MDD - Model Driven Development
MDE – Model Development Engineering (Engenharia de Software Dirigida pelos Modelos)
SOA - Arquitetura Orientada a Serviços
SOA - Service-Oriented Architecture
TDD - Test Driven Development
UFBA - Universidade Federal da Bahia
UNEB - Universidade do Estado da Bahia
XP - Extreme Programming
PMBOK - Project Management Body of Knowledge)
GQM - Goal Question-Metric
SPEM - Software & System Process Engineering Metamodel
OMG - Object Management Group
BPM - Business Process Management
EPF - Eclipse Process Framework
CIM - Modelo Independente de Computação
PIM - Modelo Independente de Plataforma
PSM - Modelo Específico de Plataforma
CITM - Modelo de Teste Independente de Computação
PITM - Modelo de Teste Independente de Plataforma
PSTM - Modelo de Teste Específico de Plataforma
UML - Unified Modeling Language
MIN - Modelo de Informação de Negócio
MF - Modelo de Funcionalidade

MN - Modelo Navegacional
ARQSERV - Modelo de Arquitetura
MIS - Modelo de Interação de Serviço
MCS - Modelo de Componente de Serviço
ARQSYS - Modelo de Arquitetura de Sistema

[brazil]babel

Capítulo 1

INTRODUÇÃO

As abordagens de desenvolvimento de software tidas como tradicionais possuem um rigor no planejamento e execução do processo de desenvolvimento do software (SEYFI; PATEL, 2010). Este rigor pode gerar alguns problemas, a exemplo do excesso de documentação produzida e do aumento de tempo entre as etapas de concepção e entrega do software. Com o passar dos anos, novas metodologias para guiar o desenvolvimento foram propostas no intuito de resolver os problemas existentes, como por exemplo: as entregas fora do prazo, sistemas que não atendiam aos desejos do usuário, custos elevados, descobertas de erros tardiamente, dentre outros. Inicialmente as propostas tinham como foco o processo de desenvolvimento e sua documentação (AITKEN; ILANGO, 2013). Há cerca de duas décadas, surgiram os métodos ágeis. Esses métodos trazem como proposta acelerar o processo de desenvolvimento do software através da construção miníma de documentação e da redução entre as etapas de concepção e a entrega do software (BECK et al., 2001), tornando assim o código o artefato principal do processo de desenvolvimento de software(TOMÁS, 2009). Dentre os métodos ágeis propostos pode-se citar o SCRUM (SCHWABER; SUTHERLAND, 2016), O *Feature Driven Development– FDD (LUCIA; QUSEF, 2010), o Extreme Programming – XP (BECK, 2000), o Dynamic Systems Development Method – DSDM(PRESSMAN; MAXIM, 2016), o Adaptive Software Development - ASD (SCHOEPPING et al., 2012), o Crystal Clear* (SCHOEPPING et al., 2012), entre outros.

As abordagens de desenvolvimento de software tidas como ágeis propõem: reduzir os ciclos entre as fases de desenvolvimento e entrega do software, possibilitar maior adaptabilidade e flexibilidade as novas solicitações dos usuários, realizar a entrega do software em ciclos curtos de desenvolvimento e disponibilizar o software funcionando dentro do prazo acordado com o usuário (TOMÁS, 2009). A utilização da metodologia ágil no desenvolvimento do software tem com base o "Manifesto ágil" (BECK et al., 2001), pelo mesmo propor a satisfação do cliente, a possibilidade de mudanças durante o desenvolvimento do software, a possibilidade do software ser entregue ao cliente por partes, a participação do cliente em todas as vazes do processo de desenvolvimento, a possibilidade realizar o processo de desenvolvimento de forma simples e ao final disponibilizar ao cliente software

funcionando. A metodologia ágil busca ser uma metodologia iterativa e incremental, promovendo os princípios e valores definidos no manifesto ágil (SCHOEPPING et al., 2012).

Em meio a desafios, tais como a heterogeneidade dos ambientes de software e a necessidade de aumentar a produtividade sem abrir mão da documentação, surgiram também abordagens para automatizar o próprio processo de desenvolvimento, a exemplo do Desenvolvimento Dirigido por Modelos (DDM). Esta abordagem, busca acelerar o processo de desenvolvimento com a redução de tempo entre as etapas de modelagem e desenvolvimento do código, possibilita o desenvolvimento multiplataforma, além de contribuir para geração da documentação do projeto. Em DDM os modelos são os principais artefatos no processo e a geração do código se tornou uma atividade secundaria neste processo (BEYDEDA et al., 2005).

A diferença entre a metodologia ágil e a abordagem DDM está na forma como elas guiam o desenvolvimento do software; a metodologia ágil com foco no código e a DDM com foco nos modelos; e se assemelham por tentar produzir softwares com maior qualidade, com menor tempo de desenvolvimento, facilidade de realizar manutenções/evoluções em conformidade com as necessidades do usuário, entre outros aspectos (UTSUNOMIYA, 2003).

Definir a forma que o software será construído não é uma tarefa trivial pois fatores relevantes ao cliente precisam ser considerados, como exemplo, o prazo de entrega da primeira versão do software e a documentação detalhada do software (PRESSMAN; MAXIM, 2016). Independente de como o software foi desenvolvido, no decorrer de sua utilização, o mesmo tende a precisar passar por manutenções devido a novas necessidades dos usuários e das organizações. Mudanças nas necessidades dos usuários e das organizações podem ocasionar manutenções classificadas como evolutivas. Quando encontradas inconformidades entre o que foi especificado e o código fonte, tais como a apresentação de falhas/erros no código, podem ocorrer manutenções classificadas como corretivas (YANG et al., 2012).

O software pode ser definido como uma visão computacional das necessidades dos usuários e das organizações e a evolução do mesmo está diretamente ligado a estas necessidades e ou problemas no código do software. Como consequência das constantes atualizações/modificações que o software pode passar ao longo do seu ciclo de vida, o mesmo poderá apresentar alguns riscos (PRESSMAN; MAXIM, 2016) como: documentação desatualizada e ou inexistente e código fonte confuso e/ou desorganizado. Neste cenário, realizar manutenções corretivas ou evolutivas no software pode ser algo crítico e demorado.

Enquanto o software passa por evoluções, o mesmo pode continuar sendo utilizado. Porém, à medida que software tem o seu ciclo de vida prolongado, a sua estrutura tende a ser degradada e as mudanças tendem a ser mais custosas. Depois de alguns anos de uso, fatores como ambiente, hardware e sistemas operacionais podem influenciar para que o software precise passar por constantes evoluções. Chega então, o momento que evoluções significativas se tornam menos rentáveis. Neste momento o software continua sendo utilizado e passando por modificações pontuais para que o mesmo continue funcionando, porém, os defeitos ou problemas apresentados passam a ser contornados pelos usuários,

o que pode indicar que o software precisa ser substituído (PRESSMAN; MAXIM, 2016).

Utilizar práticas de diferentes abordagens na integração/hibridização no processo de desenvolvimento do software pode ser uma alternativa para aumentar o ciclo de vida do software, buscando otimizar alguma estratégia que permita que o software passe por manutenções corretivas e ou evolutivas mais facilmente, utilizando o menor tempo e com menos custos. Um exemplo pode ser a utilização de práticas do desenvolvimento dirigido por modelo integrados ao método ágil SCRUM.

Com o objetivo de integrar o desenvolvimento dirigido por modelo ao SCRUM, (SALES, 2017) em seu trabalhado desenvolveu o metaprocesso ágil, chamado de ScrumDDM, com a finalidade de desenvolver projetos em diferentes tecnologias, gerar documentação com maior conformidade e realizar o desenvolvimento de forma mais rápida. O ScrumDDM integrou as técnicas de metamodelagem, modelagem e transformações da DDM ao conjunto de práticas definidos no framework SCRUM. Uma instância do metaprocesso também foi proposto pela autora como prova de conceito para validação e utilização do ScrumDDM. Adicionalmente foi realizado uma validação para verificar a viabilidade do processo.

1.1 SITUAÇÃO/PROBLEMA

Em desenvolvimento de software que utilizam práticas ágeis, o artefato mais atualizado é o código fonte. Sendo assim, a utilização dos métodos ágeis, por si só, pode não assegurar a evolução do software por existir a possibilidade de não elaborar de forma detalhada a documentação do projeto. Mudanças no software podem e devem ocorrer constantemente. Considerando que na metodologia ágil o artefato mais atualizado é o código do software, entender e realizar manutenções no software, sem a existência de uma documentação formal, pode influenciar no tempo de desenvolvimento das evoluções neste software.

Em (SALES, 2017), foi proposto a integração de práticas do DDM ao Framework de práticas ágeis SCRUM. Esta integração tem por finalidade proporcionar o desenvolvimento de projetos em diferentes tecnologias, gerar documentação com maior conformidade e proporcionar que o desenvolvimento do software ocorra mais rápido e com menor número de inconformidades. Contudo em (SALES, 2017), não ficou evidente se sua utilização preservou a característica de agilidade predominante da metodologia ágil, assim como a possibilidade de evolução do software através dos requisitos representados pelas estórias dos usuários no processo. Também, não ficou evidente a efetividade do metaprocesso na criação de novos processos, ou seja, se outros processos poderiam ser instanciados a partir do metaprocesso ScrumDDM.

1.2 QUESTÕES DE PESQUISA

Diante dos problemas apresentados na seção 1.1, as seguintes questões de pesquisa foram levantadas para nortear este trabalho.

- Q1: A utilização do metaprocesso ScrumDDM na instanciação de novos processos, preserva a agilidade característica da metodologia ágil?
- Q2: A utilização do metaprocesso ScrumDDM apoia a evolução do software através dos artefatos (documentação/modelo e transformações) oriundas do ScrumDDM?

- Q3: A utilização do metaprocesso ScrumDDM apoia a criação/instanciação de novos processos que integram DDM ao Scrum?

1.3 OBJETIVOS

Esta dissertação é uma proposta de avaliar o metaprocesso ScrumDDM, nos seguintes contextos:

- Preservar a agilidade característica do processo ágil.
- Apoiar evolução de software através de novas e/ou modificação de estórias dos usuários existentes.
- Possibilitar a especialização e reuso/adaptação do metaprocesso ScrumDDM.

1.3.1 Objetivos Específicos

Para apoiar a realização do objetivo proposto os seguintes objetivos específicos foram definidos:

- Projetar e realizar estudos para avaliar aspectos de evolução de software desenvolvidos através de processos instanciados ScrumDDM.
- Realizar medição do tempo de desenvolvimento das estórias dos usuários em um software já desenvolvimento com o metaprocesso ScrumDDM.
- Instanciar um novo processo hibrido, a partir do metaprocesso ScrumDDM.

1.3.2 Metodologia a ser seguida

Para obtenção dos objetivos específicos da pesquisa, foi utilizada à metodologia apresentada na figura 1.1.

O trabalho, segundo a metodologia observada na figura 1.1, foi dividido em 4 (quatro) fases contendo atividades desenvolvidas por (SALES, 2017) e atividades que foram desenvolvidas neste trabalho. As atividades executas no ScrumDDM por (SALES, 2017) são: (1) Definir o metaprocesso ScrumDDM, (2) Realizar *Revisão de Literatura* com o objetivo de encontrar um novo processo de software que possa ser instanciado a partir do ScrumDDM e (3) Instanciar o processo adaptado ArqProjProcess (SALES, 2017) a partir ScrumDDM. As atividades realizadas neste trabalho são: (5) Encontrar, através de uma revisão de literatura, outro processo de desenvolvimento de software, (6) Instanciar o processo encontrado com a revisão da literatura acadêmica e (4) Realizar um experimento controlado para avaliar a capacidade de evolução e agilidade do metaprocesso ScrumDDM.

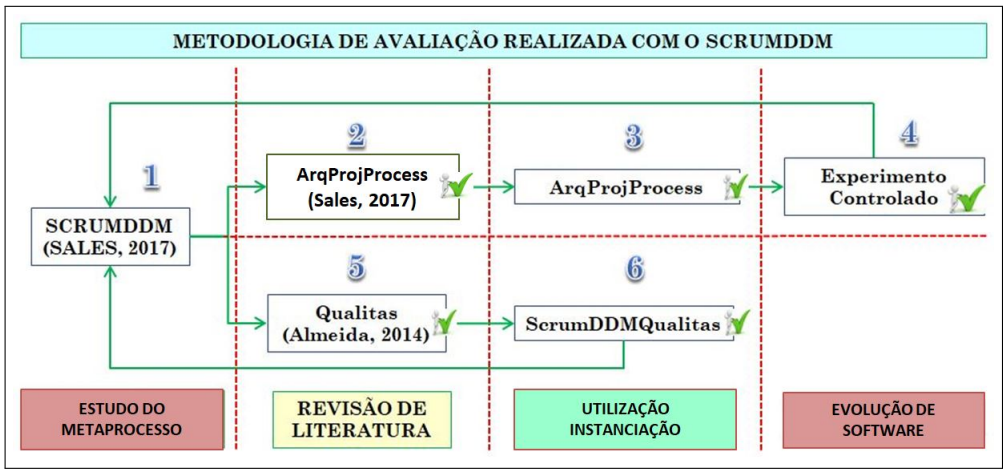


Figura 1.1 Metodologia de pesquisa a ser realizada

1.4 ESTRUTURA DO DOCUMENTO

O Capítulo 2 apresenta uma visão geral sobre a metodologia ágil, desenvolvimento dirigido por modelos, processo de software e qualidade de software, detalhando seus principais elementos.

O Capítulo 3 apresenta o objetivo do metaprocesso ágil ScrumDDM, e as duas instâncias (ScrumDDMQualitas e ArqProjProcess) e seus elementos, a exemplo de suas fases, atividades, artefatos e papeis.

O Capítulo 4 apresenta a avaliação do metaprocesso ScrumDDM quanto sua capacidade de evoluir estórias de usuários e avaliar o tempo de desenvolvimento utilizado nas evolução das estórias. Esta avaliação foi realizada através do experimento controlado.

No Capítulo 5 são apresentados os trabalhos relacionados, considerando as abordagens ágil e do DDM. As seções deste capítulo estão organizadas da seguinte maneira: na seção 5.1 é apresentado o trabalho que integra a metodologia ágil num processo de desenvolvimento dirigido por modelos; na seção 5.2 é apresentando um trabalho prático de integração ágil e DDM; na seção 5.3 é apresentando o metaprocesso ágil ScrumDDM; na seção 5.4 é apresentado o processo Qualitas e na seção 5.5 é apresentado os atributos de evolução de software.

Finalmente, no Capítulo 6 são apresentadas as considerações finais e propostas de trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos: processo de software (Seção 2.1), qualidade de software (Seção 2.2), metodologia ágil (Seção 2.3) e do Desenvolvimento Dirigido a Modelo (DDM) (Seção 2.4), de modo a oferecer suporte a compreensão das bases teóricas deste trabalho.

2.1 PROCESSO DE SOFTWARE

Um processo é um conjunto de *Atividades, Ações* e *Tarefas* realizadas na criação de um determinado produto (PRESSMAN; MAXIM, 2016). Em processo de software, a execução de uma *Atividade* busca atingir um objetivo independente da área de aplicação, do tamanho do projeto, da complexidade de esforços. Uma *Ação* de processo busca executar as tarefas do processo com a finalidade de produzir os artefatos da execução do processo. Por fim, a *Tarefa* de processo busca dividir o problema em partes em relação ao processo geral. Esta divisão orienta os envolvidos a pensar na solução do problema em partes, com o objetivo de alcançar a solução Geral do problema.

Para (PRESSMAN; MAXIM, 2016) e (SOMMERVILLE et al., 2011), um processo no contexto da engenharia de software é uma abordagem adaptável que possibilita a equipe de desenvolvimento realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. O objetivo é sempre entregar o software dentro do prazo e com a qualidade para satisfazer aos patrocinadores e aos utilizadores do software.

Um processo de software pode ser representado esquematicamente conforme observado na figura 2.1. Cada atividade é formada por um conjunto de ações definidas por um conjunto de tarefas que devem ser contempladas com o objetivo de gerar os artefatos do produto. A avaliação sobre o andamento do processo acontece através do uso de marcos, com o objetivo de informar o andamento do processo.

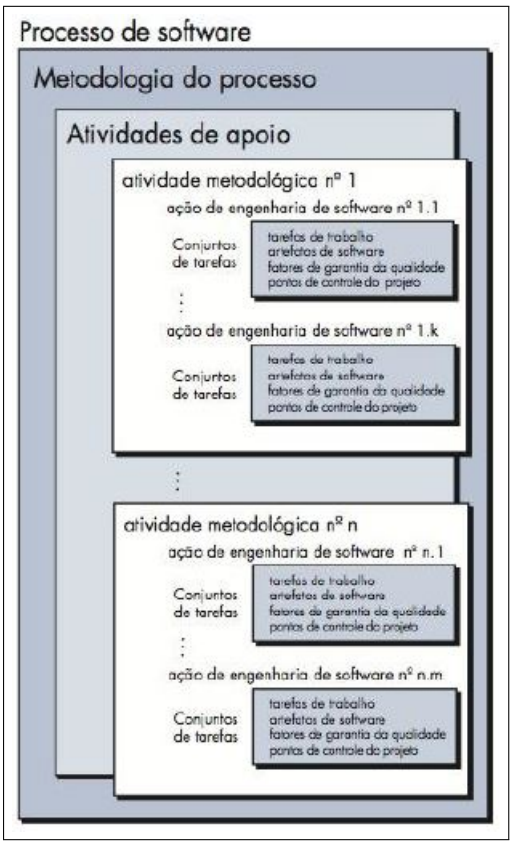


Figura 2.1 Processo de Software (PRESSMAN; MAXIM, 2016)

Genericamente, um processo de desenvolvimento de software compreende cinco atividades genéricas (PRESSMAN; MAXIM, 2016):

Comunicação: o objetivo deste tipo de atividade no processo é estabelecer comunicação com um cliente, buscando retirar o máximo de informações do cliente a fim de compreender seus objetivos, necessidades que ajudarão a definir as funções e as características que o software deve ter;

Planejamento: o objetivo deste tipo de atividade no processo é desenvolver um "mapa"para ajudar a guiar a equipe no desenvolvimento. Este mapa pode ser chamado de plano de projeto de software e contém tarefas técnicas para conduzir os riscos, os recursos necessários, os produtos resultantes e o cronograma de trabalho;

Modelagem: o objetivo deste tipo de atividade no processo é criar um esboço de modo que se possa ter uma ideia do projeto como um todo. Desta maneira, a criação de modelos serve para entender melhor as necessidades do software e o projeto tem que atender a essas necessidades modeladas;

Construção: o objetivo deste tipo de atividade no processo é gerar o código (manual ou automatizada) e testes necessários para revelar erros na codificação;

Entrega: o objetivo deste tipo de atividade no processo é entregar ao cliente o produto de software desenvolvido e fornecer um *feedback* baseado na avaliação.

Estas atividades podem ser utilizadas no desenvolvimento de qualquer software, independente de tamanho e complexidade. Os detalhes de cada processo de software podem ser diferentes, porém, as atividades serão as cinco citadas.

Modelo de Processo de Software: um modelo de processo de software é uma representação genérica e simplificada de um processo de software, ou seja, cada modelo representa uma perspectiva particular do processo, com o objetivo de fornecer informações sobre ele, por exemplo: um modelo de atividade de processo fornece informações sobre as atividades e a sequencia que elas serão executadas (SOMMERVILLE et al., 2011).

A literatura acadêmica apresenta diferentes modelos de processos. Por exemplo o modelo em cascata, desenvolvimento incremental, engenharia de software orientada por reuso, desenvolvimento dirigido por modelo e desenvolvimento dirigido por teste, metodologia ágil, etc. Por ser o modelo de processo usado neste trabalho, a seção 2.2 detalha a abordagem de desenvolvimento ágil.

2.2 METODOLOGIA ÁGIL

Com o objetivo de se contrapor as metodologias de desenvolvimento de software existentes na época, um grupo de 17 desenvolvedores apresentou, em 2001, algumas maneiras de tornar o desenvolvimento de software mais rápido (BECK et al., 2001). Estas novas maneiras formaram um conjunto de práticas conhecidas como "Manifesto para o Desenvolvimento Ágil de Software" (BECK et al., 2001). Pretendia-se com a utilização da metodologia ágil reduzir os ciclos de entrega do software, possibilitar maior adaptabilidade e flexibilidade as novas solicitações, assim como, realizar a entrega dentro do prazo estipulado com o usuário (TOMÁS, 2009) .

As práticas apresentadas no "Manifesto para o Desenvolvimento Ágil de Software" (BECK et al., 2001)), ficou conhecido como os princípios da utilização dos métodos ágeis. Estes princípios são:

- **satisfazer o cliente,** esta satisfação é possível através da entrega contínua e adiantada do software;
- **possibilitar que mudanças nos requisitos sejam bem vindas, mesmo que tardiamente,** pois a possibilidade de modificação dos requisitos ao longo do processo de desenvolvimento do software é o que aumenta o grau de competitividade para o cliente;
- **entregar frequentemente software funcionando,** uma vez que estas entregas devem acontecer após poucas semanas do início do projeto ou a poucos meses e com preferência no menor intervalo de tempo;
- **buscar que o cliente esteja sempre presente,** pois ele é o conhecedor do negócio, deve trabalhar em conjunto com os desenvolvedores, reduzindo assim possíveis falhas de concepção ao longo do projeto;

- **manter os envolvidos no projeto motivados,** está relacionado a atenção, ao ambiente e a confiança demonstrada ao time;
- **melhorar a comunicação entre o time,** a forma mais eficiente de transmitir informações entre e para uma equipe de desenvolvimento é através de conversas cara a cara;
- **entregar o software funcionando,** a evolução do projeto é medida através de cada entrega do software funcionando;
- **promover ambiente sustentável,** a utilização dos processos ágeis faz com que os patrocinadores, desenvolvedores e usuários sejam capazes de manter um ritmo indefinidamente constante;
- **promover excelência técnica,** continua atenção ao projeto pode assegurar um software bem desenvolvido e o aumento da agilidade;
- **tornar o desenvolvimento do projeto algo simples,** somente desenvolver o que é essencial ao negócio e ao cliente;
- **manter Times auto-organizáveis,** a auto-organização pode produzir melhores arquiteturas, requisitos e designs;
- **incentivar mudança de comportamento,** em intervalos regulares, o time reflete sobre seu comportamento com o objetivo de se tornarem mais eficientes.

O mesmo grupo de 17 desenvolvedores identificou como valores mais relevantes nesta nova metodologia de desenvolvimento, os seguintes:

- *a comunicação entre as pessoas e as interações* se mostra mais importante que o processo e as ferramentas;
- *a software funcionando* é mais importante que uma documentação detalhada do software;
- *o cliente presente e colaborando* como um aspecto mais importante que a burocracia das relações no que tange as negociações de contrato;
- *a possibilidade de modificação no projeto* é esta como algo mais importante que seguir um plano.

Com base no manifesto ágil, diversos métodos ágeis foram propostos, com características iterativa e incremental, com o objetivo de promover os princípios e os valores do manifesto (SCHOEPPING et al., 2012), e enfatizar a diminuição do prazo da entrega da primeira versão do software ao usuário (BRAGA; LEAL, 2013). Entre os diversos métodos propostos podemos citar o Scrum (SCHWABER; SUTHERLAND, 2016), *Feature Driven Development (FDD) (LUCIA; QUSEF, 2010), Extreme Programming (XP) (BECK, 2000), Dynamic Systems Development Method (DSDM) (PRESSMAN;*

MAXIM, 2016), Adaptive Software Development (ASD) (SCHOEPPING et al., 2012), Crystal Clear (SCHOEPPING et al., 2012), entre outros.

Apesar de não ser prioridade, alguns métodos ágeis como o Scrum, o XP e o *Crystal Clear* podem utilizar na etapa da entrega do software a prática da documentação breve.

A possibilidade de gerar uma documentação breve é motivada pela utilização dos artefatos: diagrama de caso de uso, estórias dos usuários, e *product backlog* considerados nestes métodos. Os objetivos desses artefatos são descritos a seguir.

- **Utilização de diagrama de caso de uso:** tem como objetivo documentar o ponto de vista do usuário, descrevendo as funcionalidades que o software pode conter. Esta descrição não considera aspectos técnicos do software (PRESSMAN; MAXIM, 2016).

- **Arquivamento das estórias dos usuários:** as estórias dos usuários tem por objetivo descrever as características e as funcionalidades necessárias para construção do software, descrições estas realizadas pelo cliente com o apoio da equipe de desenvolvimento. Neste momento o cliente deve definir quais serão as funcionalidades prioritárias (PRESSMAN; MAXIM, 2016).

- **Criação do *product backlog:*** lista de funcionalidades priorizadas pelo cliente. Esta lista de funcionalidade é o ponto de partida no *framework* Scrum, podendo ser modificada ao longo do desenvolvimento do software (PRESSMAN; MAXIM, 2016).

Alguns métodos como o XP, FDD, *Crystal Clear* e ASD podem utilizar, na etapa de desenvolvimento e validação do incremento, práticas associadas a qualidade de código a exemplo de: (i) refatoração de código, (ii) integração continua e (iii) Inspeção de código (SCHOEPPING et al., 2012), ou seja, estas práticas associadas buscam melhorar aspectos da manutenção, estabilidade e a confiabilidade do código produzido.

- **Refatoração de código:** tem como objetivo melhorar a estrutura do código sem modificar o seu comportamento, a fim de tornar o código mais simples, compreensível e reutilizável. Esta é uma pratica do XP que pode ser utilizada sempre que necessária, porém a sua utilização não é obrigatória (SCHOEPPING et al., 2012).

- **Integração contínua:** o código produzido na iteração atual é integrado ao código do incremento anterior, buscando-se com esta prática a diminuição de conflitos e erros no código fonte. Esta é uma prática dos métodos XP e Crystal Clear (SCHOEPPING et al., 2012).

- **Inspeção de código:** tem como objetivo encontrar defeitos no código fonte e verificar se o mesmo está simples para ser entendido. Recomenda-se que um desenvolvedor inspecione o código de outro desenvolvedor. Esta recomendação é necessária para que a inspeção ocorra de forma imparcial e sem vícios. A inspeção de código fonte é uma prática de validação do incremento, presente nos métodos FDD e ASD (SCHOEPPING et al., 2012).

Na Figura 2.2 podemos observar um modelo de ciclos iterativos para um processo ágil simplificado (adaptado de (BRAGA; LEAL, 2013)).

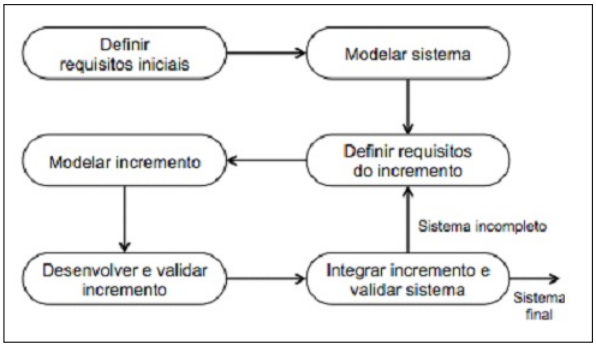


Figura 2.2 Modelo Simplificado do Processo Ágil (adaptado de (BRAGA; LEAL, 2013))

Este modelo de ciclos iterativos é composto pelas seguintes etapas: 1 - Definição dos requisitos iniciais, 2 - Modelagem do sistema, 3 - Definição dos requisitos do incremento, 4 - Modelagem do incremento, 5 - Desenvolvimento e validação do incremento e 6 - Integra incremento e valida sistema. Conforme a Figura 2.1, as etapas 3, 4, 5 e 6 formam uma iteração do processo ágil. Sendo assim, cada etapa é definida da seguinte maneira:

- **definição dos requisitos iniciais,** misto são realizados os levantamentos de requisitos, funcionalidades, definição das entradas e saídas esperadas do software, possíveis restrições. Todas estas informações são utilizadas no processo de modelagem do software.
- **modelagem do sistema,** realizada de forma simples para que o software funcione. Esta simplicidade deve acontecer porque, a cada novo ciclo iterativo, os requisitos iniciais podem ser modificados. Nesta etapa também são definidos quais requisitos devem ser priorizados no decorrer do projeto.
- **definição dos requisitos do incremento:** que irão compor cada incremento, quais as funcionalidades serão entregue e quando isso vai ocorrer.
- **modelagem do incremento,** de forma simples e objetiva, estimando maneiras de programar cada funcionalidade e como estas funcionalidades serão integradas ao software.
- **desenvolvimento e validação do incremento,** considerando a modelagem do incremento realizada na etapa 4 - Modelagem do incremento, este incremento é desenvolvido e testado. Os testes podem ser escritos antes ou depois da codificação do incremento.
- **integração do incremento e validação do sistema,** onde o incremento é integrado ao sistema e passa por testes de integração e aceitação. O ciclo só é finalizando quando o incremento realizar tudo o que foi especificado pelo usuário e

o software desenvolvido estiver funcionando de forma correta. Caso contrário, o incremento deve voltar para as etapas 3 - Definição dos requisitos do incremento ou 5 - Desenvolvimento e validação do incremento.

Apesar da diminuição do tempo de entrega, dentre outras vantagens, a utilização da metodologia ágil pode apresentar algumas desvantagens: (i) falta de conhecimento dos desenvolvedores sobre a metodologia, (ii) cliente satisfeito com o protótipo e (iii) falta de um marco final do projeto (SEYFI; PATEL, 2010). As desvantagens citadas são detalhadas abaixo.

- Pode ocorrer dos desenvolvedores não compreenderem como criar funcionalidades cooperativas (funções reutilizáveis do projeto), dificultando assim que outros envolvidos tenham uma visão geral do projeto.

- No desenvolvimento ágil é comum que exista prototipação do software em desenvolvimento. Em alguns casos o protótipo foi tão bem desenvolvido que o cliente demonstra satisfação em usá-lo e começa a reclamar de iterações posteriores. Isso pode ocorrer porque o cliente pode modificar seu processo de trabalho para utilizar o protótipo do software não finalizado.

- O custo do projeto pode ser elevado, devido ao mesmo não possuir um marco formal para o final do desenvolvimento do software. O marco que norteia o final do projeto é formalizado por um acordo para que novas solicitações sejam desenvolvidas e entregue na próxima versão do software.

Sendo assim, os métodos ágeis podem ser adequados a determinadas situações, enquanto os métodos tradicionais a outras.

A escolha do processo a ser utilizado no desenvolvimento de um software envolve critérios como: (i) o prazo de entrega, (ii) o tamanho e localidade da equipe, (iii) o envolvimento do cliente no projeto, (iv) a geração de documentação e (v) a possibilidade de mudanças corretivas ou evolutivas no projeto, dentre outros. Independente da metodologia adotada no desenvolvimento do software, os critérios citados devem ser considerados na produção do software (SCHOEPPING et al., 2012).

Diferenciando das metodologias tradicionais que possuem um rigor na execução de cada fase do processo de desenvolvimento, a metodologia ágil considera mais relevante as relações entre os envolvidos no processo, a possibilidade de mudança e o software funcionando, em relação a seguir um plano pré-estabelecido no início do projeto (BECK et al., 2001).

O fato das metodologias tradicionais possuírem um rigor na execução de cada fase do processo pode possibilitar que o software passe por manutenções corretivas ou evolutivas mais facilmente (SEYFI; PATEL, 2010), pois leva à produção de uma documentação detalhada do software que, em geral, auxilia as manutenções corretivas ou evolutivas.

Na Figura 2.3 a seguir é possível observar, uma comparação entre a metodologia tradicional de desenvolvimento de software e a metodologia ágil de desenvolvimento de software (AITKEN; ILANGO, 2013).

Metodologia Tradicional de Desenvolvimento de Software	Metodologia Ágil de Desenvolvimento de Software
Incentiva uso rígido do processo e das ferramentas.	Incentiva a colaboração entre os envolvidos no projeto.
Toda atividade é medida por uma intensa documentação.	O software de trabalho é utilizado como métrica de avaliação para a evolução do projeto, tornando assim a concepção do código a atividade mais relevante no desenvolvimento do software.
Não envolve os usuários continuamente.	Envolve os usuários continuamente.
Mudanças não são bem vindas. Dificilmente pode haver modificações no projeto uma vez que o contrato foi assinado.	Mudanças são bem vindas, mesmo durante o desenvolvimento do projeto.
Realiza uma entrega ao final do projeto.	O software é entregue por partes até que todas as solicitações dos usuários sejam atendidas.
Geralmente não incentiva uma entrega dentro de duas semanas a dois meses, ou seja, as iterações são mais longas.	Incentiva que a entrega do software ocorra por partes através de iterações curtas, que podem durar entre duas semanas e dois meses (máximo).
Geralmente não incentiva equipes auto-organizadas.	O software pode ser entregue no melhor das equipes co-localizadas.
Acompanha formalmente as fases do ciclo de vida do software, por incentivar à análise formal no planejamento do projeto.	Não seguem de forma rígida as fases. Por exemplo, pode não realizar uma análise formal no planejamento do projeto.
As solicitações são documentadas na fase de coleta e devem ser entregue ao usuário ao final do desenvolvimento do software.	Novas solicitações dos usuários podem ser adicionadas durante do desenvolvimento do projeto.
Não enfatiza o ritmo constante entre desenvolvedores e o usuário, pois os usuários não são envolvidos durante o desenvolvimento do projeto.	Com o objetivo de promover o desenvolvimento, é incentivada a integração entre os desenvolvedores e os usuários do software durante todas as fases do projeto.

Figura 2.3 Metodologia tradicional de desenvolvimento de Software X Metodologia ágil de desenvolvimento de software (adaptado de (AITKEN; ILANGO, 2013))

Na Figura 2.3 podemos observar que a característica de uma contrapõe-se a da outra metodologia, por exemplo: a metodologia tradicional incentiva seguir de forma rígida o processo e a utilização de ferramentas, a possibilidade de gerar documentação detalhada, e não envolver constantemente o cliente; já as metodologias ágeis procuram promover a colaboração dos envolvidos no processo de desenvolvimento, a geração do código fonte é enfatizada como a mais relevante e a participação do cliente ao longo do desenvolvimento do software é incentivada.

2.3 DESENVOLVIMENTO DIRIGIDO POR MODELOS (DDM)

O Desenvolvimento Dirigido por Modelos (DDM) é uma abordagem que coloca o modelo como o elemento principal no desenvolvimento do software (MACIEL et al., 2013). Esta abordagem tende a facilitar o desenvolvimento do software por elevar o nível de abstração do problema, por exemplo, com a criação do diagrama de caso de uso para documentar o problema sob a perspectiva do usuário (PRESSMAN; MAXIM, 2016). Desta forma, a atenção dos envolvidos no processo de desenvolvimento do software será na construção dos modelos ao invés do código. Isso é possível por esta abordagem possibilitar geração de código (semi)automático a partir dos modelos e das transformações destes (BEYDEDA et al., 2005). DDM apresenta como benefícios aumentar a produtividade, reduzir o custo do desenvolvimento e da manutenção em sistemas de software, pois o problema esta descrito no modelo e estes podem passar por uma cadeia de transformação até a geração do código fonte.

A realização mais conhecida da DDM é a MDA (*Model Driven Archtecture*) padronizada pela OMG (ALVES; MACHADO; RAMALHO, 2008). A MDA considera o desenvolvimento do software em três níveis de abstração além do código: o *Computational Independent Model* (CIM)/Modelo Independente de Computação, o *Platform Independent Model* (PIM)/ Modelo Independente de Plataforma e o *Platform Specific Model* (PSM)/ Modelo Especifico de Plataforma (MDA, 2014).

A MDA pretende aumentar a portabilidade, interoperabilidade e a produtividade de software, partindo do princípio de que os conceitos são mais estáveis que a tecnologia. Desta forma, o domínio existe de forma independente da tecnologia, possibilitando realizar a modelagem e depois definir qual a tecnologia que será utilizada no desenvolvimento do software (MAGALHãES, 2016). A seguir, serão apresentados alguns conceitos do domínio do desenvolvimento dirigido por modelos.

- **Metamodelo:** Um modelo da linguagem de modelagem onde é definida a semântica, a estrutura e as restrições para geração de modelos. Um metamodelo descreve aspectos comportamentais e estruturais dos modelos UML, e é uma instância do MOF (OMG; NOTATION, 2008).

- **Modelo**: Conjunto de elementos com representações abstratas de alguma realidade física, simplificada e dentro de algum contexto. Em DDM os modelos são artefatos que representam o software nos diversos níveis de abstração.Estes precisam ser escritos utilizando linguagem de modelagem com semântica bem definida como,

Unified Modeling Language (UML) e *Domain-Specific Language* (DSL) (MELLOR et al., 2005).

- **Plataforma:** É a especificação de um ambiente de execução para um conjunto de modelos. Ela deve ter uma implementação da especificação a qual representa, ou seja, uma realização dela. Desta forma, ela é um conjunto de subsistemas e tecnologias que provê um conjugado de funções coerentes que podem ser utilizadas a partir de interfaces e padrões específicos, onde qualquer aplicação suportada pela plataforma pode utilizar sem se preocupar como a funcionalidade foi implementada (MELLOR et al., 2005).

- **Transformação de Modelo:** Atividade de transformar os elementos existentes de um modelo (origem) em elementos de um modelo (destino). Segundo a arquitetura MDA, uma transformação é a execução de um conjunto de regras e técnicas utilizadas para modificar, refinar um modelo e obter outro modelo. (RUTLE et al., 2018).

  Existem dois tipos de transformações: (i) *Model to Model* (M2M) modelo gerando modelo e (ii) *Model to Text* (M2T) modelo gerando texto/código. Estes dois tipos de transformações possibilitam diferentes níveis de abstração, ou seja, um processo com (n) modelos de origem que gera (n) modelos de destino e código fonte em (n) linguagens. As regras para transformação M2M e M2T podem ser desenvolvidas nas linguagens de transformação *Atlas Transformation Language* (ATL) e ou *Query/View/Transformation* (QVT), considerando a semântica, a estrutura e as restrições definidas no metamodelo conceitual definido.

2.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo apresentamos os conceitos norteadores para o desenvolvimento deste trabalho, a exemplo da metodologia ágil, DDM, processo de software e qualidade de software.

Conceitualmente o Scrum é um método ágil voltado para gestão de projetos ágeis. Desta maneira, o mesmo pode agregar práticas de outros métodos ágeis e práticas do desenvolvimento dirigido por modelo e desenvolvimento dirigido por teste. Assim como outros métodos ágeis, o Scrum mantém o objetivo de diminuir o tempo entre as fases de *PreGame, Game* e *PostGame*, mantendo enfase na produção do código.

Contrapondo-se aos métodos ágeis, o desenvolvimento dirigido por modelo (DDM), tem no processo de desenvolvimento a enfase no modelo. Desta maneira, ao final do desenvolvimento, pode ser entregue ao cliente um conjunto de artefatos formado pela documentação do projeto e código fonte, ambos atualizados.

As necessidades dos usuários e das organização tendem a mudar constantemente. Considerando isso, o Scrum e o DDM contribuem para esta mudança com a possibilidade de desenvolver projetos adaptáveis e com documentação para geração/atualização do código.

O metaprocesso desenhado ao decorrer deste trabalho vai possibilitar a visualização da integração de práticas do desenvolvimento dirigido por modelos, práticas de outras métodos ágeis ao Scrum.

Capítulo

3

METAPROCESSO SCRUMDDM

Este capítulo apresenta o metaprocesso ScrumDDM considerando seu objetivo (Seção 3.1), papéis e responsabilidades da equipe do metaprocesso ScrumDDM (Seção 3.2), suas Fases, Atividades, Subatividades, Tarefas e Artefatos (Seção 3.3), as instâncias desenvolvidas a partir do ScrumDDM(Seções 3.4 e 3.5). Por fim, um resumo sobre o ScrumDDM nas Considerações sobre o capítulo (3.6).

3.1 O SCRUMDDM

O ScrumDDM é um metaprocesso ágil proposto por (SALES, 2017), que introduz e integra um conjunto de práticas oriundas do Desenvolvimento Dirigido a Modelos (DDM) ao *framework* Scrum.

O ScrumDDM (SALES, 2017) foi desenvolvido buscando não desvirtuar os valores e princípios da metodologia ágil. Desta forma, o ScrumDDM tem por finalidade proporcionar o desenvolvimento de projetos em diferentes tecnologias, gerar documentação em conformidade entre o que foi solicitado pelo usuário e o que foi desenvolvido e, por fim, possibilitar que o desenvolvimento do software ocorra mais rápido e com menor número de inconformidades. Para selecionar as práticas ágeis que pudessem ser integradas com práticas do DDM foi utilizado o framework de práticas ágeis (VILAIN; FAGUNDES; MACHADO, 2007). Este framework possibilitou que as práticas atividades: *Requisitos, Arquitetura, Desenvolvimento, Integração* e *Teste*, e as tarefas *Retrospectiva Sprint, Reunião de Planejamento, Revisão do Sprint, Reunião diária* pudessem ser utilizadas no metaprocesso ScrumDDM desenvolvido.

O ScrumDDM foi especificado em *SPEM - Software Process Engineering Metamodel* (OMG; NOTATION, 2008), e desenvolvido em EPF [1]. Sendo assim (SALES, 2017), buscou desenvolver um metaprocesso que possibilitasse um desenvolvimento e gestão simples, que pudesse ser hibridizado e atendesse a vários domínios.

[1] *Eclipse Process Framework Composer* (EPF, 2014)

3.1.1 Apresentação

A Figura 3.1 mostra uma visão geral das fases do metaprocesso ágil ScrumDDM:*Pregame, Game* e *Postgame*, oriundas do Scrum.

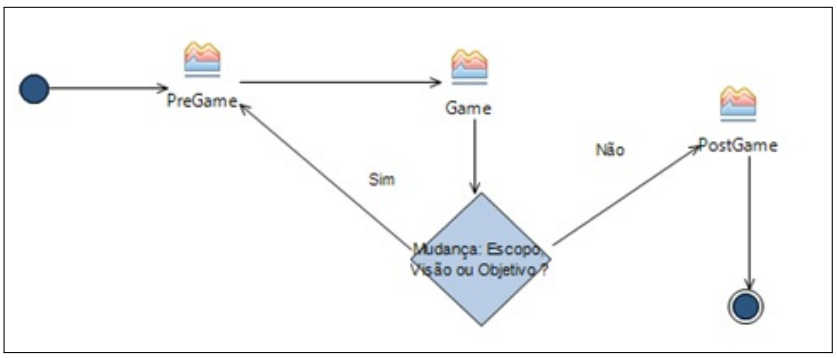


Figura 3.1 Visão Geral do Metaprocesso ScrumDDM (adaptado de (SALES, 2017))

Nas fases de *Pregame* e *Game*, foram introduzidas práticas oriundas do DDM e de outros métodos ágeis, bem como, subatividades de modelagem (Metamodelos, Modelos e Perfis UML) e transformação de modelos em 2 níveis M2M (*Model to Model*) e M2T (*Model to Text*). Na fase *PostGame*, a mesma não recebe práticas de DDM, uma vez que esta fase é destina para entrega do software ao usuário final.

O ScrumDDM é composto pela definição de papéis (responsáveis), artefatos (documentação e código) e atividades (disciplinas) que visam auxiliar no desenvolvimento do projeto nas 2 fases macro do Scrum: (i) *Pregame* e (ii)*Game* (Beedle et al., 2002; Koscianski, 2006). Estas fases são realizadas ao longo da execução do projeto e têm propósitos e marcos bem específicos.

3.1.2 Ciclo de Vida do ScrumDDM

Os detalhes do ciclo de vida do metaprocesso ScrumDDM (SALES, 2017), podem ser observado na figura 3.2. A fase de *Pregame* é realizada no início do projeto e engloba a atividade Visão, contendo as subatividades de Arquitetura e Requisito e as Tarefas da Visão. Na fase de*Game* são executadas as atividades da *Sprints*, que é composta pelas subatividades *Desenvolvimento, Teste* e *Integração*, além das tarefas da *Sprint*. No início de cada sprint, a tarefa *Reunião de Planejamento* é reexecutada com objetivo de rever o planejamento macro do projeto e de planejar a nova sprint, considerando os resultados alcançados e as alterações de escopo solicitadas. O retorno à fase *Pregame* pode ocorrer em alguns momentos especiais, como por exemplo, quando são identificadas mudanças muito significativas no escopo, visão ou objetivo do projeto. Uma vez obtido o produto final, a fase de *Encerramento* é realizada.

O metaprocesso ScrumDDM segue o ciclo de vida do Scrum e adiciona a ele características DDM. As fases *PreGame, Game* do Scrum, recebe as práticas, *Modelagem, Transformação (M2M)* e *Transformação (M2T)* do DDM que são executadas e seus respectivos artefatos são construídos. Com base nestas definições, diversos processos de

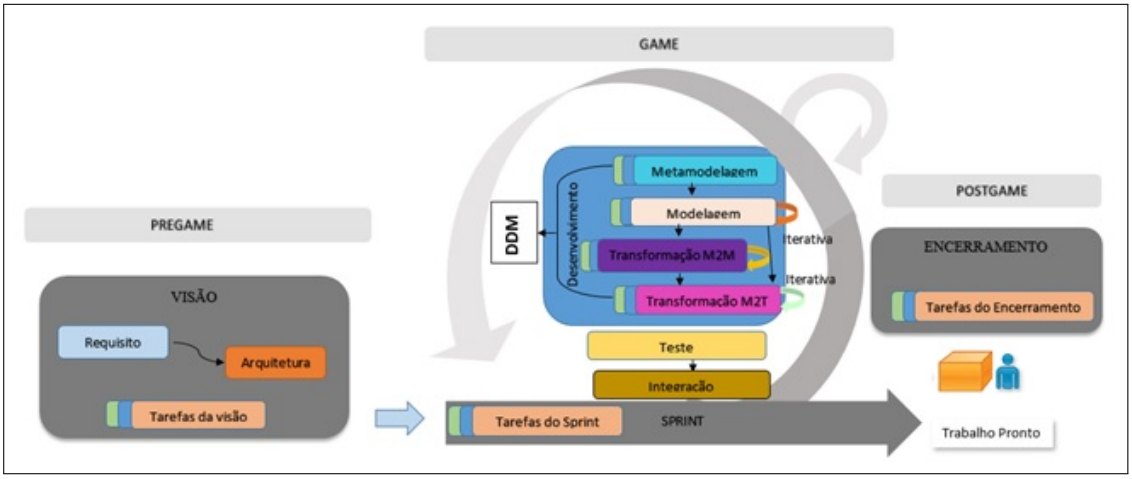


Figura 3.2 Ciclo de Vida do ScrumDDM (adaptado de (SALES, 2017))

desenvolvimento podem ser instanciados a partir do ScrumDDM selecionando as características que melhor se adéquem ao contexto em que será utilizado.

3.2 PAPEIS E RESPONSABILIDADES DA EQUIPE NO SCRUMDDM

Os papéis e responsabilidades da equipe do metaprocesso são o ScrumDDMMaster, o ScrumDDMTeam e o *Product Owner*, conforme descritos abaixo (SALES, 2017):

- **ScrumDDMMaster,** responsável pela aplicação do Scrum e do DDM ao projeto. Para isso será indispensável que o mesmo tenha o conhecimento no método ágil Scrum e no desenvolvimento dirigido por modelo.

- **ScrumDDMTeam,** responsável por desenvolver o produto em conformidade com o que foi especificado pelo cliente, além de definir os metamodelos, os modelos e as transformações para o projeto, ou seja, o *ScrumDDMTeam* é o time de desenvolvedores do projeto.

- ***Product Owner,*** responsável em possuir a visão do produto como se fosse o dono do produto.

Com o objetivo de desenvolver o metamodelo e as transformações M2M e M2T, é necessário a existência de um arquiteto de software, para que o ScrumDDMMaster e o ScrumDDMTeam, possam desenvolver o software a partir dos modelos e transformações já desenvolvidas por este profissional (SALES, 2017).

Desta forma, os envolvidos possuem papeis e responsabilidades especificas a cada fase do metaprocesso ágil.

3.3 FASES, ATIVIDADES, SUBATIVIDADES, TAREFAS E ARTEFATOS DO SCRUMDDM

O ScrumDDM está organizado em fases. Cada fase compreende atividades e subatividades que devem ser seguidas pelos participantes do projeto. Para isso são especificadas diversas tarefas para cada subatividade, bem como os artefatos envolvidos para cada tarefa. A Tabela 3.1 resume a especificação do ScrumDDM. Cada uma das colunas corresponde a uma fase do metaprocesso (*Pregame, Game* e *PostGame*). Em seguida são definidos os elementos (atividades, subatividades e tarefas) que compõem cada fase. Note que as tarefas estão classificadas entre tarefas oriundas do Scrum e tarefas oriundas do DDM. Adicionalmente nas ultimas linhas da figura estão descritos também os artefatos envolvidos em cada fase. As seções a seguir detalham cada uma das fases e seus elementos.

Tabela 3.1 Fases, Atividades, Subatividades, Tarefas e artefatos do ScrumDDM

Metaprocesso ScrumDDM		
PréGame	**Game**	**PostGame**
Atividades		
Visão	**Sprint**	**Encerramento**
SubAtividades		
Requisito Arquitetura	Desenvolvimento Teste Integração	Aprovação dos Entregáveis Conclusão do projeto
Tarefas Scrum:		
Reunião de planejamento da Sprint Criar Produto backlog Priorizar backlog Planejamento de versão Escrever Estorias dos usuarios Decompor estorias dos usuarios Estimar estorias dos usuarios Escrever teste de aceitação Executar teste de aceitação Escrever caso de uso Atualizar caso de uso	Reunião de planejamento da Sprint Reunião diária Reunião de revisão do sprint Retrospectiva do sprint	
Tarefas DDM:		
Definir/utilizar Metamodelo Definir/utilizar Modelos	Metamodelagem Modelagem Tranformação M2M Tranformação M2T	
Artefatos do Conjunto:		
Metamodelos de Domínio Modelos Propostos Estorias dos Usuários Casos de uso	Modelos Mecanismo de extensão UML Perfil Expressões OCL Código	Trabalho Concluido
Artefatos das práticas DDM		
Artefatos das práticas ágeis		

Na Tabela 3.1 podemos observar um resumo sobre o ScrumDDM quanto as suas fa-

ses, atividades, subatividades, tarefas e artefatos, bem como, a forma como cada fase foi definida. Podemos observar também que as tarefas, *Metamodelagem*, *Modelagem*, *Transformação M2M* e *Transformação M2T* são tarefas executadas/utilizadas na subatividade *Desenvolvimento* da atividade *Sprint* da fase *Game*.

3.3.1 Fase de Pregame

A Figura 3.3, mostra o fluxo da atividade *Visão* da fase *PreGame*. O processo de desenvolvimento é iniciado com a execução da tarefa de *Reunião de Planejamento* do Sprint_visão para apresentar as informações e funcionalidades do produto e possibilitar que o *Product Owner* e o *ScrumDDMMaster* consigam criar o plano de desenvolvimento em alto nível e o *backlog do produto*. É nesta fase que o CIM (*Computation Independent Model*) e as configurações da plataforma PSM (*Platform Specific Model*) são definidas.

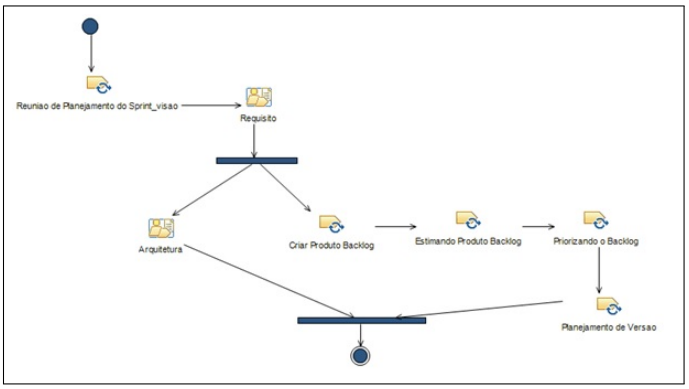


Figura 3.3 Fluxo da atividade *Visão* da fase *Pregame* (adaptado de (SALES, 2017))

A fase de *PreGame* do ScrumDDM (SALES, 2017), é iniciada com a realização da reunião de planejamento da sprint visão que tem como resultado a lista dos requisitos a serem desenvolvidos. Com base nesses requisitos é definida a arquitetura do Sistema e criado o *backlog* do produto. Este *backlog* é utilizado para estimar o prazo de desenvolvimento, priorizar o que será desenvolvido e planejar as diversas versões.

A adição das práticas DDM começa na fase *Pregame* com o objetivo de atender o CIM (*Computation Independent Model*) e dar suporte ao PSM (*Platform Specific Model*) correspondente ao processo DDM.

Requisito: Esta subatividade tem o objetivo de capturar dos *stakeholders* os requisitos do domínio e a visão geral das funcionalidades desejadas para o sistema. Em ambiente ágil, os requisitos são definidos ao longo do desenvolvimento, o que permite a equipe buscar conhecer apenas o suficiente para identificar como o sistema funcionará e suas restrições, possibilitando desta maneira o mapeamento das estórias de usuário e dos modelos.

O *ScrumDDMTeam* e o *Product Owner* definem um objetivo para o sprint, através

de uma breve descrição textual sobre qual o objetivo da sprint.

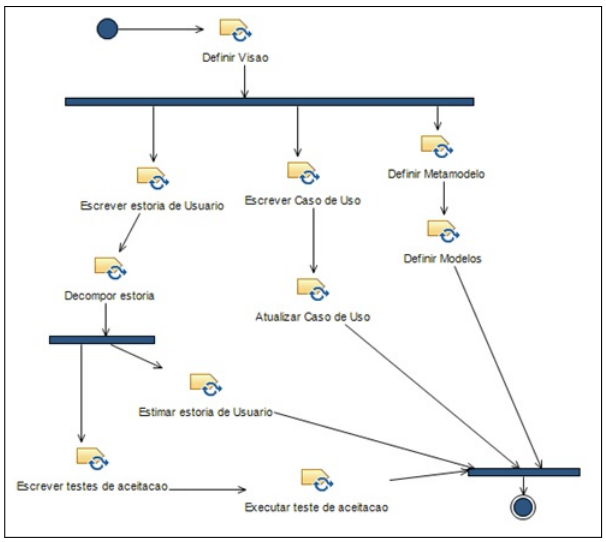


Figura 3.4 Fluxo da Subatividade *Requisito* da fase *Pregame* (adaptado de (SALES, 2017))

Podemos visualizar na Figura 3.4 que a subatividade *Requisito* pode ser composta por três tarefas com execução concorrentes: *Definir (utilizar) Metamodelo, Escrever estórias dos usuários e Escrever Caso de Uso.* A tarefa *Definir (utilizar) Metamodelo* possui como subtarefa a *Definição (utilização) de Modelos.* A tarefa *Escrever Caso de Uso* possui a subtarefa *Atualizar o Caso de Uso.* Por fim, a tarefa *Escrever estórias dos usuários* possui o maior número de subtarefas da subatividade *Requisito*, a exemplo de: *Decompor estórias*, *Escrever teste de aceitação* e *Executar os testes de aceitação* escritos.

Definir/utilizar Metamodelo: o objetivo desta tarefa é definir quais os metamodelos existentes podem ser utilizados nas transformações e no desenvolvimento do software. Podem ser utilizados metamodelos já existentes ou definição para o desenvolvimento de um novo metamodelo, desde que atendam ao propósito do projeto.

Definir Modelos: o objetivo desta tarefa é identificar e descrever os modelos do domínio de aplicação considerando o objetivo proposto. Os modelos definidos devem estar em conformidade com o metamodelo já definido.

Arquitetura: o objetivo desta subatividade é definir a arquitetura para o software que será desenvolvido a partir do metaprocesso ScrumDDM. Para isso deve-se fazer uso da visão do produto e das estórias dos usuários já definidas, com a finalidade de guiar as decisões técnicas que poderão ser tomadas. A definição da subatividade de *Arquitetura* pode ser considerada um fator crítico ao projeto. Isto porque um bom desenho da arquitetura pode proporcionar manutenções ou evoluções no projeto mais facilmente.

Conforme descrito na subatividade de *Requisito*, os requisitos/ estórias dos usuários tendem a evoluir ao longo do desenvolvimento, o que permite à equipe buscar conhecer

apenas o suficiente para identificar como o sistema funcionará e suas restrições. Sendo assim, não existe no ScrumDDM uma lista de requisitos/ estórias dos usuários completa nas iterações iniciais, sendo de responsabilidade do time capturar o máximo de informações possíveis para definir, descrever e refinar a arquitetura. Desta forma, vemos nesta subatividade arquitetural uma grande oportunidade de aplicar práticas do DDM (uso de modelos e transformação M2M) para obter uma documentação adequada.

Em resumo, após a execução da atividade *Visão* da fase *PreGame*, o projeto deve estar definido e com as macros de todas as atividades, subatividades e tarefas planejados e o time alocado. É possível que no decorrer do processo possa acontecer o retorno à fase de *PreGame*, quando identificadas mudanças muito significativas no escopo, visão ou objetivo do projeto. Quando mudanças não forem identificadas é possível seguir para a fase *Game*.

3.3.2 Fase de Game

Assim como ocorre na fase *PreGame*, esta fase do ScrumDDM (SALES, 2017) é composta por subatividades e tarefas relacionadas ao desenvolvimento dirigido por modelo e pelo método ágil Scrum. A finalidade desta fase é desenvolver o software através da utilização do ScrumDDM, afim de possibilitar que a documentação e software desenvolvido estejam em conformidade com o que foi especificado pelo usuário.

Compõem a atividade *Sprint* as subatividades: *Desenvolvimento, Teste e Integração* e pelas tarefas *Reunião de planejamento da Sprint, Reunião diária, Reunião de revisão do sprint*, *Retrospectiva do sprint*, *Metamodelagem, Modelagem, Transformação M2M* e *Transformação M2T*.

O conjunto formado pelas tarefas *Reunião de planejamento, Reunião diária, Reunião de Revisão do Sprint e a Retrospectiva do Sprint*, estão relacionadas ao Scrum e as tarefas *Utilização de Metamodelagem, Utilização de Modelagem, Execução da Transformação M2M* e *Execução da Transformação M2T* estão relacionadas ao desenvolvimento dirigido por modelos.

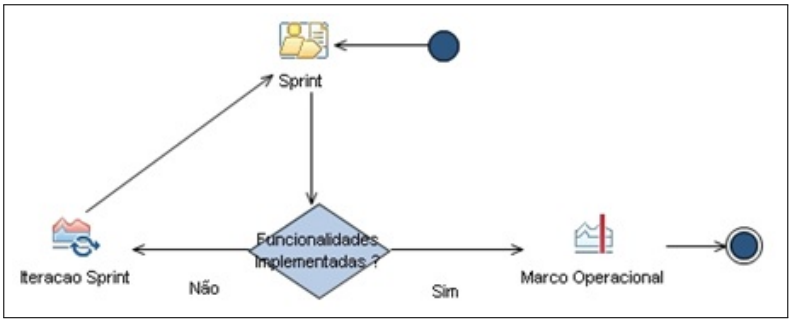


Figura 3.5 Fluxo da fase*Game* (adaptado de (SALES, 2017))

Podemos observar na Figura 3.5 que na fase de *Game* é aonde ocorre a execução das subatividades e tarefas, que compõe uma ou múltiplas *Sprints*.

No início de cada *sprint* é executada a tarefa de *Reunião de planejamento* com objetivo de rever o planejamento macro do projeto e planejar a nova *sprint*, considerando os resultados alcançados e as alterações de escopo solicitadas.

Desenvolvimento: Esta subatividade é responsável pela solução técnica do projeto em conformidade com as subatividades de *Arquitetura* e os *Requisitos* especificados na atividade de *Visão* na fase de *Pregame*. O foco desta subatividade são os modelos propostos, porém estes modelos podem passar por codificações, com o objetivo de complementar o código gerado a partir das transformações originalmente definidas.

Compõe o *Desenvolvimento* as seguintes tarefas: *Metamodelagem, Modelagem, Executar Transformação M2M e Transformação M2T, Analisar Cadeia de Transformação, Gerar Código, Completar Código, Refatorar Código e Versionamento.*

Os detalhes e conceitos de cada tarefa que tem a finalidade direcionar a subatividade de *Desenvolvimento* foram descritas abaixo:

Metamodelagem: nesta tarefa será utilizado o metamodelo definido na subatividade de *Requisito* da atividade *Visão* na fase de *Pregame*, porém se o mesmo já estiver construído esta tarefa não será necessária. Isso porque a estrutura, semântica e restrições do metamodelo já estão desenvolvidas. Esta tarefa é executa uma única vez, caso seja necessário o sua atualização no processo de desenvolvimento.

Modelagem: nesta tarefa serão construídos os modelos com o objetivo de atender o domínio de aplicação, sendo estes modelos os artefatos de entrada para as Transformações. Tendo como artefato de entrada as estórias dos usuários e/ou as descrições de caso de uso, ambos os artefatos devem estar em conformidade com o Metamodelo e o Modelo Proposto.

Transformação Model to Model M2M: nesta tarefa do *Desenvolvimento* serão construídas ou selecionadas as regras de transformação entre modelos, segundo objetivo/necessidades do *ScrumDDMTeam*. Os artefatos de entrada para Transformação M2M são: metamodelos, modelos e as expressões em OCL. Os artefatos de saída são: as regras de Transformação M2M desenvolvidas na linguagem ATL.

Transformação Model to Text-M2T: nesta tarefa do *Desenvolvimento* serão selecionadas e utilizadas as transformação de modelo para texto (M2T) previamente definidas e construídas, segundo o objetivo/necessidades do *ScrumDDMTeam*. Os artefatos de entrada para Transformação M2T são:modelos. Os artefatos de saída são: novos modelos resultados da execução de transformação M2T desenvolvidas na linguagem ACCELEO.

Teste: nesta subatividade são medidos os progressos do projeto em pequenos incrementos com a finalidade de garantir a não ocorrência de defeitos, aplicando a cada iteração do *Sprint* um *feedback* frequente para verificar se o sistema está em conformidade com os requisitos.

Compõe a subatividade *Teste* as seguintes tarefas: *Definir Testes, Implantar Ambiente, Executar Testes de (cliente e unidade) e Inspecionar Código.* Após a execução de todas as tarefas pelo *ScrumDDMTeam*, o mesmo reporta os resultados ao usuário e ou corrige as inconformidades encontradas antes de disponibilizar a versão ao usuário.

Integração: nesta subatividade são integradas as partes já desenvolvidas e testadas pelo *ScrumDDMTeam*, além de realizar testes mínimos no incremento a fim de validar a integração (SALES, 2017).

3.3.3 Fase de Postgame

Nesta fase práticas de DDM não são utilizadas, isso porque a finalidade da fase é entregar o produto de software desenvolvido ao usuário, através da execução da atividade *Encerramento.*

A atividade *Encerramento* é composta pelas seguintes tarefas: *Aprovação dos Entregáveis* e *Conclusão do projeto.* Estão envolvidos nesta fase o *ScrumDDMTeam*, os *Stakeholders* e o usuário. Em conjunto, os envolvidos citados realizam a avaliação final do produto. Sendo assim, nesta fase o projeto é encerrado, o software e a documentação são entregues ao usuário.

3.3.4 Artefatos do ScrumDDM

No ScrumDDM os artefatos são gerados tanto pelas práticas do Scrum quanto pelas práticas do DDM. As atividades macros (Visão, Sprint e Enceramento) são responsáveis por produzir os artefatos relacionados ao Scrum. Já os artefatos advindos das práticas DDM são gerados das subatividades que serão usados como entrada e sáida nas atividades do processo, representando assim a documentação mínima e necessária para a gestão ágil do projeto aderente a‘s práticas do DDM.

3.4 O SCRUMDDM E SUAS INSTÂNCIAS

Por se tratar de um metaprocesso, o ScrumDDM precisa ser instanciado para que esta instância possa ser utilizada em determinado projeto de desenvolvimento de software (SALES, 2017). As instâncias podem ter objetivos distintos para atender domínios específicos do desenvolvimento de software, como por exemplo: WEB, SOA, Desenvolvimento Dirigido por testes (DDT), DDM, aplicações móveis, etc. O ScrumDDM foi utilizado para instanciar dois processos de Software com o objetivo de integrar/hibridizar as abordagem de desenvolvimento dirigido por modelos e desenvolvimento dirigido por teste ao Scrum. Inicialmente um processo hibrido desenvolvido por (BRAGA, 2011) foi adaptado, instanciado e definido como ArqProjProcess (SALES, 2017), Posteriormente o Qualitas (ALMEIDA et al., 2014) foi instanciado neste trabalho. Além de promover a integração entre as abordagens e métodos ágeis, o ScrumDDM busca promover uma documentação mais consistente sem comprometer as características de ambas as abordagens, por exemplo a agilidade e a comunicação entre os envolvidos no processo de desenvolvimento.

O ArqProjProcess (SALES, 2017) foi utilizado através de um experimento controlado para verificar se o ScrumDDM era capaz de promover evolução de software sem comprometer a agilidade no desenvolvimento. O ScrumDDMQualitas é uma nova instancia a partir do ScrumDDM este foi adaptado do processo Qualitas (ALMEIDA et al., 2014) original, seu foco é o teste de software com o objetivo é identificar e corrigir os erros o quanto antes no desenvolvimento, contribuindo assim com a melhora da qualidade do software.

3.4.1 O processo Qualitas

O Qualitas é um processo de desenvolvimento de software, iterativo e incremental com o objetivo de desenvolver projetos orientados a modelos, utilizando as abordagens de desenvolvimento dirigido por modelos (DDM) e desenvolvimento dirigido por testes (DDT) (ALMEIDA et al., 2014) .

O processo Qualitas foi desenvolvido, considerando que o mesmo pudesse ser integrado a metodologia ágil e ao PMBOK (*Project Management Body of Knowledge*). O foco do Qualitas é o desenvolvimento de pequenos e médios projetos software (ALMEIDA et al., 2014).

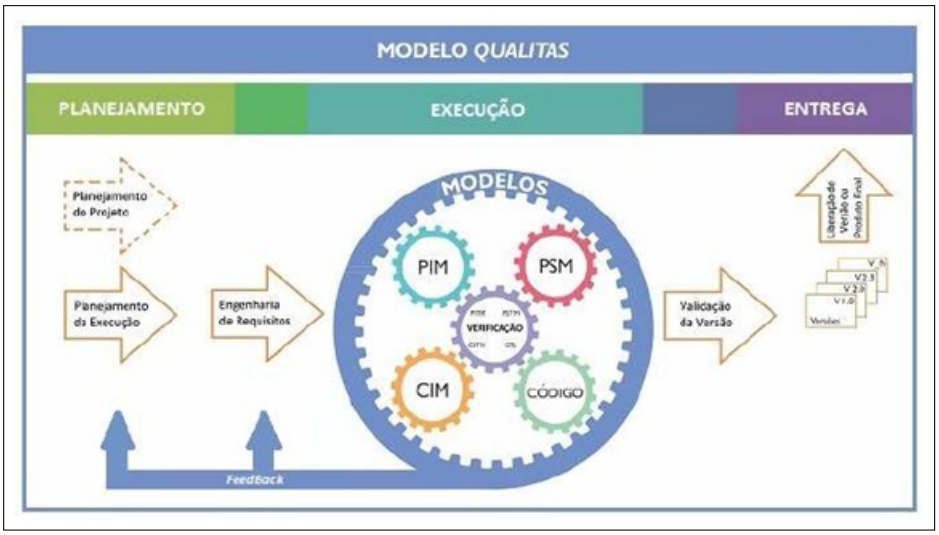


Figura 3.6 Processo Qualitas (adaptado de (ALMEIDA et al., 2014))

Na Figura 3.6 é possível observar o desenho da execução do processo Qualitas através das fases, etapas e atividades que o processo Qualitas possui. A composição do Qualitas pode ser observada na Tabela 3.2, por três fases: (i) *Planejamento*, (ii) *Execução* e (iii) *Entrega*. A fase (i) *Planejamento* contém as atividades de *Planejamento do Projeto* e *Planejamento da Execução*, e os artefatos: *Plano de Gestão do Projeto* e *Plano de Execução*. A fase (ii) *Execução* contém as atividades *Engenharia de Requisitos, Modelo Independente de Computação (CIM), Modelo de Teste Independente de Computação (CITM), Modelo Independente de Plataforma (PIM), Modelo de Teste Independente de*

Plataforma (PITM), Modelo Especifico de Plataforma (PSM), Modelo de Teste Especifico de Plataforma (PSTM), Código, Geração de Testes (GTs) e *Validação da Versão* e dos artefatos: *CIM, CITM, PIM, PITM, PSM, PSTM, Codigo Fonte* e *Código Teste*. Estas atividades podem ocorrer de forma incremental a depender da quantidade de versões definidas no escopo do projeto. Por fim, a fase *Entrega* que contém a atividade: *Liberação de Versão* ou *Produto Final*. Nesta fase é disponibilizado ao usuário a versão final do produto desenvolvido, testado e validado.

Tabela 3.2 Resumo das fases, atividades com seus objetivos no processo Qualitas (adaptado de (ALMEIDA et al., 2014))

Fases	Atividades	Objetivos
Planejamento	Planejamento do Projeto	Construir o Plano de Gestão do Projeto
	Planejamento da Execução	Produzir o Plano de Execução
Execução	Engenharia de Requisitos	Produzir o Documento de Requisitos
	CIM	Construir os Modelos Independentes de Computação
	CITM	Gerar os Modelos de Testes Independentes de Computação
		Testar os Modelos Independentes de Computação
		Analisar os Resultados dos Testes
	PIM	Gerar o Modelo Independente de Plataforma
	PITM	Gerar os Modelos de Testes Independentes de Plataforma
		Testar os Modelos Independentes de Plataforma
		Analisar os Resultados dos Testes
	PSM	Gerar os Modelos Específicos de Plataforma
	PSTM	Gerar os Modelos de Testes Específicos de Plataforma
		Testar os Modelos Específicos de Plataforma
		Analisar os Resultados dos Testes
	Geração de Código	Gerar o Código Fonte a partir do PSM
		Complementar, se necessário, o Código Fonte
	Geração de Testes	Gerar os Testes a partir do código
	Validação da Versão	Validar se a versão está atendendo ao que foi especificado.
		Avaliar os Resultados da Execução dos Testes
Entrega	Liberação de Versão ou Produto Final	Disponibilizar cada versão validada, ou ao término do projeto, o produto final para o cliente

3.4.2 Instância do processo Qualitas

Nesta seção, uma instância do processo Qualitas a partir do metaprocesso ScrumDDM será apresentada. Esta instância será chamada de **ScrumDDMQualitas**.

A adaptação ao processo original de (ALMEIDA et al., 2014) foi necessária, pois o *Qualitas* não utilizava os conceitos e aplicação para metodologia ágil presentes no ScrumDDM. As seguintes adaptações foram realizadas; as fases do Qualitas passaram a

ser atividades das fases do ScrumDDM, as atividades do Qualitas passaram a ser subatividades; uma fase teve que ser excluída e algumas subatividades mudaram de fase. Os detalhes sobre a composição do ScrumDDMQualitas serão explicados a seguir.

Com o objetivo de instanciar o processo Qualitas (ALMEIDA et al., 2014) a partir do ScrumDDM, o ScrumDDMQualitas preservou atividades, subatividades e tarefas do processo Qualitas, além de fases, atividades, subatividades e tarefas do ScrumDDM. As seguintes adaptações foram realizadas: As fases *Pregame, Game* e *PostGame* do ScrumDDM foram preservadas. As fases *Planejamento, Execução* e *Entrega* do Qualitas passaram a ser atividades no ScrumDDMQualitas. A atividade *Planejamento de Projeto* deixou de existir, pois no processo Qualitas ela já era opcional e no ScrumDDM o processo começa com a tarefa *Reunião de Planejamento*. As subatividades *Engenharia de Requisitos, CIM e CITM* passaram para atividade de *Planejamento*, com o objetivo de agrupar as atividades relacionadas a concepção do código. Na fase *Game*, a tarefa *Retrospectiva da Execução* substituiu a subatividade *Validação da Versão*, que conceitualmente eram equivalentes; Na fase *PostGame*, a subatividade *Liberação da Versão* continuou na atividade *Entrega*. Quanto aos papeis e responsabilidades, a instância preservou os papeis e responsabilidades do ScrumDDM, a exemplo de *Product Owner*, ScrumDDMMaster e ScrumDDMTeam.

Podemos observar na figura: 3.7 a seguir, o ciclo de vida da instância ScrumDDM-Qualitas.

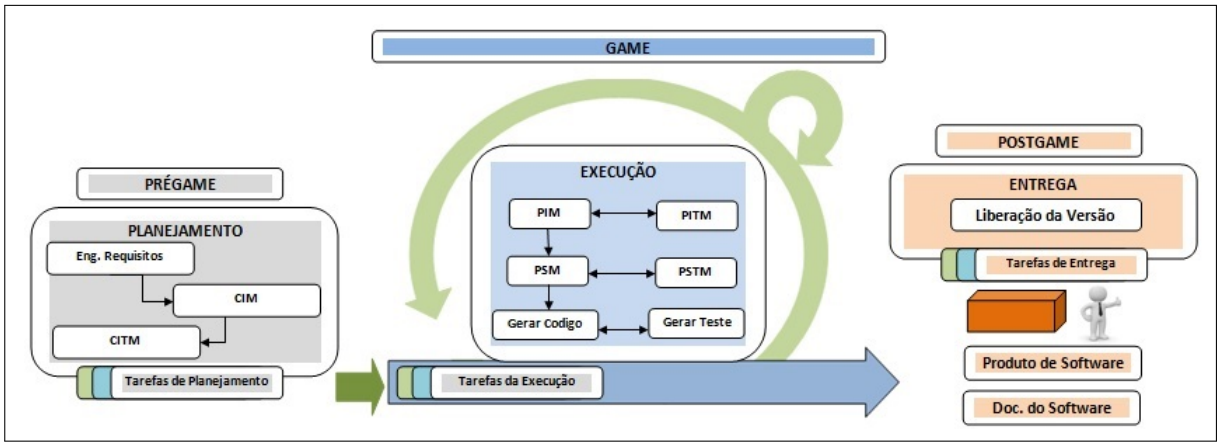


Figura 3.7 Ciclo de Vida ScrumDDMQualitas

Fase de *Pregame*: fase inicial do processo, sendo de responsabilidade do *ScrumDDM-Master* e do *Product Owner* obter junto ao cliente o máximo de informações sobre o produto de software a ser construído e apresentar estas informações ao *ScrumDDM-Team*. As tarefas da subatividade *Engenharia de Requisitos* do processo Qualitas: *Identificar, Levantar, Analisar* e *Especificar o Problema* foram substituídas pelas tarefas do ScrumDDM: *Criar* e *Priorizar* o *Product Backlog*, Escrever, Decompor e Estimar as Estórias dos usuários, assim com as subatividades CIM e CITM. Esta fase tem como objetivo a criação do *Product Backlog* e do Plano de Execução do Desenvolvimento.

Fase de*Game*: na fase de*Game* é executado o desenvolvimento do projeto, sendo de responsabilidade do *ScrumDDMMaster* e do *ScrumDDMTeam* desenvolver a atividade de execução e suas subatividades e tarefas. Foi adicionada nesta fase as tarefas relacionadas ao SCRUM a exemplo de *Reunião de planejamento*, *Reunião diária*, *Reunião de Revisão da Execução* e a *Retrospectiva da Execução*.

PostGame: a fase de *Postgame* é a ultima fase do processo, composta pela atividade de *Entrega*, pela subatividade *Liberação da Versão* e, por fim, pela tarefa de *Disponibilizar a Versão*, sendo de responsabilidade do *ScrumDDMMaster*, *Product Owner* executar a atividade de Entrega, sua subatividade e sua tarefa com o objetivo de entregar ao cliente o *Produto de Software* e a *Documentação do Projeto*.

3.4.3 Fases, Atividades, Subatividades e Tarefas do ScrumDDMQualitas

Esta subseção detalha as fases do **ScrumDDMQualitas**.

3.4.3.1 Fase de *Pregame*: conforme a figura 3.8, temos o fluxo da fase *PreGame*, que se inicia com a tarefa *Reunião de Planejamento de Execução*, para atender a subatividade *Engenharia de Requisito*, desenvolvendo assim, o *Product Backlog* e posteriormente o artefato *Plano de Execução do Desenvolvimento*. Após a criação do *Plano de Execução do Desenvolvimento* o *ScrumDDMTeam* começa a construção dos artefatos: Modelos Independente de Computação (CIM) e dos Modelos de Testes Independestes de Computação (CITM). Existem nesta fase, tarefas relacionadas ao DDM (*Gerar CIM*), DDT (*Gerar CITM*) e o Scrum (*Criar Product Backlog*).

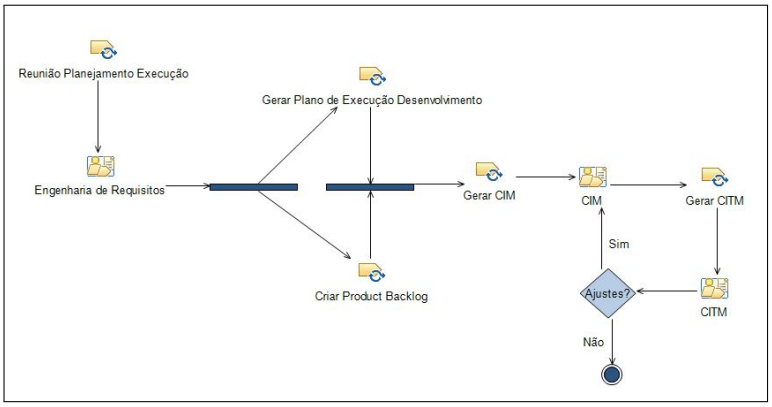


Figura 3.8 Fluxo da fase *PreGame*

As subatividades que compõe a fase de *Pregame* são:

Subatividade Engenharia de Requisitos: nesta subatividade será construído o artefato *Plano de Execução do desenvolvimento*, preservando as tarefas do ScrumDDM para criação do *Product Backlog*. O *Plano de Execução do desenvolvimento*, será o artefato de entrada da tarefa *Gerar CIM* na subatividade *(CIM)*.

Subatividade (CIM): nesta subatividade com a execução da tarefa *Gerar CIM* que os Modelos de Domínio ou os Modelos de Negócio serão construídos. Estes modelos

podem ser representados através de, por exemplo, Notação de Modelagem de Processo de Negócio, Diagramas de Fluxo de Dados ou UML nos Diagramas de Atividade.

Subatividade (CITM): nesta subatividade são construídos os Modelos Teste Independente de Computação (CITM) com a tarefa *Gerar CITM*. Seu o objetivo é testar, analisar e validar os resultados da aplicação do (CITM) no (CIM). O artefato de entrada para esta subatividade é o (CIM) e o artefato de saída será o resultado dos testes. Com o resultado dos testes, pode ser necessário realizar algum ajuste no (CIM) ou no (CITM) e, após a conclusão dos ajustes, é iniciada a fase *Game* com a tarefa *Gerar PIM* através da transformação (M2M) do (CIM) para (PIM).

3.4.3.2 Fase de *Game*: conforme a figura 3.9, temos o fluxo da fase *PreGame*, ele se inicia com a tarefa *Reunião de Planejamento* com o objetivo de construir os modelos independente de plataforma, os modelos de teste independente de plataforma, os modelos especifico de plataforma, os modelo de testes específicos de plataforma. Por fim, o código fonte e o código de teste ambos gerados através de uma cadeia de transformação entre modelos desenvolvidos.

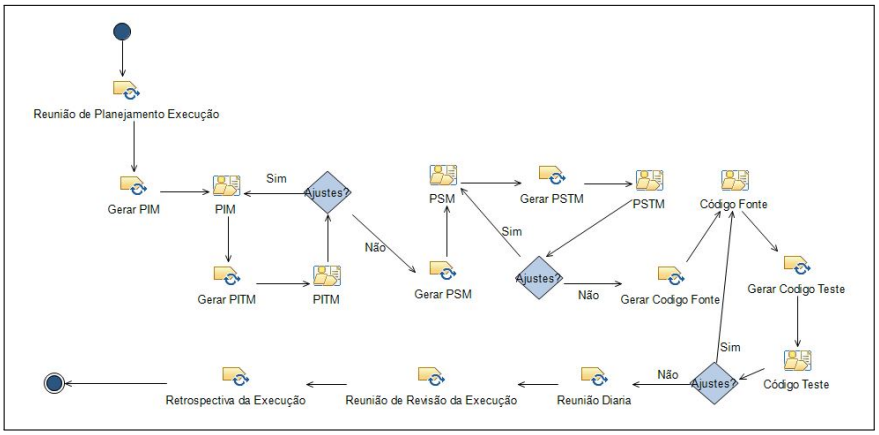


Figura 3.9 Fluxo da fase *Game*

Existem nesta fase tarefas relacionadas à DDM (*Gerar PIM, Gerar PSM, Gerar Código Fonte, Transformar M2M /M2T*), à DDT (*Gerar PITM, Gerar PSTM, , Gerar Código Teste*) e ao Scrum (*Planejar, Revisar e Realizar Retrospectiva da Sprint*).

Subatividade PIM: nesta subatividade são construídos os *PIM - (Modelos Independentes de Plataforma)*, que tem o objetivo de descrever o comportamento e sua estrutura do software. Estes Modelos podem ser representados através de diagrama de classes, diagrama de caso de uso em *Unified Modeling Language* (UML). O artefato de entrada da subatividade (PIM) é o Modelo Independente de Computação (CIM) e o artefato de saída é o modelo independente de plataforma (PIM).

Subatividade (PITM): nesta subatividade são construídos os Modelos Teste Independente de Plataforma (PITM) com a tarefa *Gerar PITM*, seu objetivo é testar, analisar

e validar os resultados da aplicação do (PITM) no (PIM). O artefato de entrada para esta subatividade é o (PIM) e o artefato de saída será o resultado dos testes. Com o resultado dos testes, pode ser necessário realizar algum ajuste no (PIM) ou no (PITM). Após a conclusão dos ajustes, é iniciada a tarefa *Gerar PSM* através da transformação (M2M) do (PIM) para (PSM).

Subatividade PSM: nesta subatividade são construídos os Modelos Especifico de Plataforma (PSM), com objetivo de descrever a solução concreta do objetivo do Software em uma plataforma especifica. Cada (PSM) possui elementos, caraterísticas e restrições da plataforma adotada. O artefato de entrada do (PSM) é o (PIM) e o artefato de saída é o (PSM).

PSTM: nesta subatividade são construídos Modelos Teste Especifico de Plataforma (PSTM) com a tarefa *Gerar PSTM*, seu objetivo é testar, analisar e validar os resultados da aplicação do (PSTM) no (PSM) e o artefato de entrada da subatividade é o (PSM) e o artefato de saída será o resultado dos testes. Com o resultado dos testes pode ser necessário a realização de algum ajuste no (PSM) ou no o (PSTM). Após a conclusão dos ajustes, é iniciada a tarefa *Gerar Código Fonte* através da transformação (M2T) do (PSM) para (Código Fonte).

Subatividade Geração de Código: nesta subatividade é construído/completado o código fonte especifico por plataforma através da execução da tarefa *Gerar Código Fonte* que utiliza para isso a execução da transformação M2T, além de alguma ferramenta e/ou *Plugins* que aopie transformar o artefato de entrada (PSM) no código fonte do software, que é o artefato de saída da subatividade.

Subatividade Geração de Teste: nesta subatividade são construídos os Modelos de Testes, Casos de Testes e/ou Código de Testes através da tarefa *Gerar Testes*. O artefato de entrada da tarefa (*Gerar Teste*) É o código fonte do software e o artefato de saída é o resultados dos testes. Com o resultado dos testes pode ser necessário a realização de algum ajuste no software ou nos Modelos de Testes, Casos de Testes e/ou Código de Testes. Após a conclusão dos ajustes, é iniciada a fase *PostGame* com a atividade *Entrada* através da tarefa *Liberar Versão*.

3.4.3.3 Fase de *PostGame*: esta fase corresponde a finalização das atividades do processo de desenvolvimento, com a realização de entrega do produto e da documentação construída ao cliente, além da transferência dos aprendizados chave e celebração da finalização. Como pode ser visto na figura 3.10, a fase *Postgame* conta com: Um marco e uma atividade *Liberação de Versão* e duas tarefas *Liberar Versão do Software* e *Disponibilizar Documentação do Projeto*. Seu marco compreende a entrega do produto final, a entrega da documentação e o fechamento do projeto.

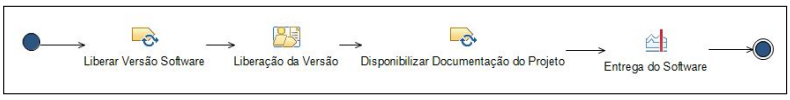


Figura 3.10 Fluxo da fase *PostGame*

3.4.4 Considerações da Seção

Em resumo o ScrumDDMQualitas é um processo de desenvolvimento de software instanciado a partir do ScruDDM que integra: desenvolvimento dirigido por modelos, desenvolvimento dirigido por teste e o Scrum. Considerando a existência do processo Qualitas(ALMEIDA et al., 2014)), foi necessário realizar uma adaptação para que o mesmo pudesse ser utilizado como instância do metaprocesso ScrumDDM. Para isso foi adicionados na adaptação ScrumDDMQualitas fases, subatividades e tarefas do ScrumDDM, a exemplo das estórias dos usuários e das tarefas do *Sprints*. Buscou-se não desvirtuar características do ScrumDDM nem do Qualitas, a fim de otimizar a aplicação de seus conceitos.

3.5 INSTÂNCIA DO ARQPROJPROCESS A PARTIR SCRUMDDM

O ArqProjProcess é uma adaptação realizada por (SALES, 2017) ao processo de (BRAGA, 2011). Esta adaptação ao processo existente foi realizada para que o processo pudesse ser hibridizando com o framework Scrum e automatizado para ser utilizado a partir do ScrumDDM (SALES, 2017).

A automação/adaptação do processo de (BRAGA, 2011) acontece da seguinte forma: o conjunto de tarefas que compõe a atividade *Especificar Modelo de Negócio* da fase *(CIM)* foi reduzido à tarefa de *Modelar Conceitos de Negócio* uma vez que esta é responsável em captar o máximo de informações sobre o software que será desenvolvido. Com relação à atividade *Analisar Serviço* da fase *PIM*, todas as tarefas foram mantidas, porém, as mesmas foram adaptadas. Por fim, a atividade *Projetar Serviço* da fase *PSM*, foi reduzida às tarefas *Definir Arquitetura* e *Projetar Back-End*, ficando esta última tarefa com a responsabilidade de Gerar o Código do projeto.

Com a finalidade de relacionar o ArqProjProcess ao ScrumDDM, será descrito a relação entre as fases, atividades e tarefas de ambas as propostas. As fases do ScrumDDM como: *Pregame, Game* e *PostGame* permanecem iguais, logo o ArqProjProcess contempla essas três fases também. No entanto, estas fases devem passar por adaptações para atender as atividades *Especificar Modelo de Negócio, Analisar Serviços e Projetar Serviços.*

Fase *Pregame*: nesta fase são extraídas dos usuários o máximo de informações sobre o projeto que será desenvolvido, sendo de responsabilidade do *ScrumDDMMaster* e do *Product Owner* obter estas informações e repassa-las ao *ScrumDDMTeam* para que estes, por sua vez, executem as tarefas *Escrever, Decompor e Estimar as Estórias dos*

usuários da subatividade de *Requisito*. Os artefato de entrada desta fase são as estórias dos usuários. Os artefato de saída podem ser os metamodelos de domínio e modelos propostos.

Fase *Game*: nesta fase é construído/desenvolvido o projeto do software. Para isso é inicializada a tarefa *Reunião de Planejamento* e, logo depois, a subatividade *Desenvolvimento* contendo três tarefas: (i) *Especificar Modelo de Negócio* para desenvolver os modelos de analise; (ii) *Analisar Serviço* para desenvolver o modelos de projeto e (iii) *Projetar Serviço* para desenvolver os modelos de implementação e a geração do código (SALES, 2017).

Fase *PostGame*: nesta fase é realizada a conclusão do projeto e a a disponibilização do software pronto ao usuário. O *ScrumDDMMaster* e o *Product Owner* são os responsáveis em executar a atividade de *Encerramento*.

As fases, atividades, tarefas e artefatos da instancia ArqProjProcess foi composta considerando as atividades, subatividades e tarefas em relação ao ScrumDDM e suas necessidades, conforme demonstrado na Tabela 3.3 a seguir.

Tabela 3.3 Resumo das fases, atividades da instância Adaptada ArqProjProcesss

PréGame	Game	PostGame
Atividades		
Visão	**Sprint/Desenvolvimento**	**Encerramento**
SubAtividades		
Requisito	Especificar modelo de negócio (ES) Analisar Serviços (AS) Projetar Serviços (PS)	
Tarefas		
Escrever, Decopor e Estimar Estorias dos usuarios, Escrever, Executar teste de aceitação, Escrever, Atualizar Caso de Uso, Definir Metamodelo e Modelos	Reunião diaria, Reunião de revisão da sprint e Retrospectiva do sprint Transformação M2M (AS) e Transformação M2T (PS)	Aprovar Entregaveis Consluir Projeto
Artefatos do Conjunto:		
	Modelo de Funcionalidade (ES) Modelo de Informação de Negocio (ES) Modelo de Arquitetura de Serviço (AS) Modelo de Interação dos Serviços (AS) Modelo de Componente dos Serviços (AS) Modelo de Arquitetura do Sistema (PS) Classe em uma linguagem especifica JAVA(PS)	Projeto Arquitetural do Software Documentação

O ArqProjProcess manteve as três fases do ScrumDDM: *Pregame, Game* e *PostGame*. As fases *Visão, Sprint* e *Encerramento*, a subatividade *Arquitetura* na fase *Pregame* do ScrumDDM foram retiradas da instância do ArqProjProcess, pois, o processo ArqProjProcess envolve a criação de um projeto arquitetural em Arquitetura Orientada a Serviços (SOA). Na fase *Game* a remoção das subatividades *Teste* e *Integração*, por não serem

necessárias no projeto arquitetural (SALES, 2017). Na mesma fase a subatividade *Desenvolvimento* passa a conter as tarefas de *Especificar Modelo de Negócio*, *Analisar Serviços* e *Projetar Serviços*.

3.5.1 Fases, Atividades, Subatividades, Tarefas do ArqProjProcess

O ArqProjProcess (SALES, 2017) é composto pelas fases *Pregame, Game* e *PostGame* e por três atividades: *Planejamento, Sprint* e *Encerramento*, assim como no ScrumDDM.

3.5.1.1 Fase *Pregame*: a atividade *Visão* da fase de *PreGame* é inicializada, com a tarefa de *Reunião de Planejamento* com o objetivo de construir o *Product Backlog* através das tarefas *Criar, Estimar e Priorizar o Product Backlog* planejando com o *Product Backlog*, a Versão do produto.

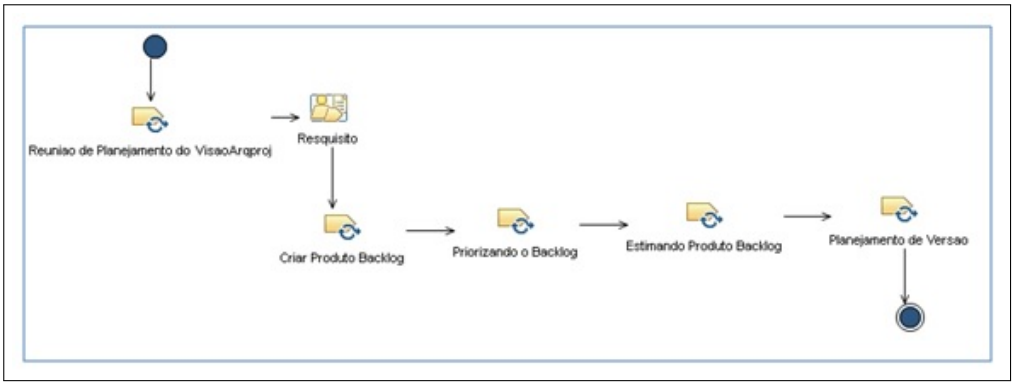


Figura 3.11 Fluxo da Atividade Visão (adaptado de (SALES, 2017))

Portanto, a fase *Pregame* da instância Arqprojprocess, é composta por: Uma atividade de *Visão*, uma subatividade de *Requisito* e cinco tarefas a exemplo de: *Reunião de planejamento, Criar, Estimar e Priorizar Product Backlog* e *Planejar versão.*

Requisito: Esta subatividade preserva as mesmas características e fluxo do ScrumDDM. Porém, o seu foco esta em guiar os envolvidos no projeto, definindo para isso estratégias, requisitos e restrições não funcionais e o estilo da arquitetura. Esta subatividade é composta pelas tarefas: *Definir Visão, Escrever, Decompor e Estimar Estórias dos Usuários, Escrever e Executar Teste de Aceitação, Atualizar e Escrever Caso de Uso* e *Definir Metamodelos e Modelos.*

As estórias dos usuários e os casos de uso desenvolvidos são os artefatos de entrada para construção dos modelos, a exemplo do modelo de funcionalidade e o modelo de informação de negócio. Mesmo existindo a possibilidade de utilizar as tarefas *Atualizar e Escrever Caso de Uso*, o ArqProjProcess foi desenvolvido dando preferência a utilização das estórias dos usuários.

Existem nesta fase tarefas que tem relação com DDM *Definir Metamodelos e Definir Modelos* e tarefas relacionadas ao Scrum *Reunião de planejamento, Criar, Estimar e Priorizar Product Backlog*.

Definir Metamodelo: Esta tarefa tem o objetivo de detalhar a arquitetura e atender o estilo arquitetural com o metamodelo em UML e SOAML. Este metamodelo definido deve ser capaz de conter informações detalhada da arquitetura.

Definir Modelos: O objetivo desta tarefa é expor as visões arquiteturais a fim de atender as necessidades dos envolvidos no projeto. Sendo assim, os modelos gerados constituem uma documentação do processo de desenvolvimento e busca aumentar a conformidade entre as estórias dos usuários e o projeto arquitetural desenvolvido. Os modelos definidos no ArqProjProcess são: modelo de funcionalidade, modelo de informação de negócio, arquitetura de serviço, modelo de interação dos serviço, modelo de componentes dos serviço e arquitetura do sistema.

3.5.1.2 Fase *Game*: nesta é inicializada com a tarefa *Reunião de Planejamento* da atividade *Sprint*, seguindo para subatividade de *Desenvolvimento* composta pelas tarefas: *Especificar Modelo de Negócio, Analisar Serviço* e *Projetar Serviço*, cada tarefa citada atinge um nível de detalhamento da arquitetura sendo possível obter ao final o código fonte. Após a execução da subatividade de *Desenvolvimento*, a tarefa de *Reunião Diária* e acionada que posteriormente será executada a tarefa *Reunião e Revisão* da atividade *Sprint*. Por fim, a tarefa *Retrospectiva do Sprint* será executada. Podemos observar a fase de *Game* representada na figura 3.12.

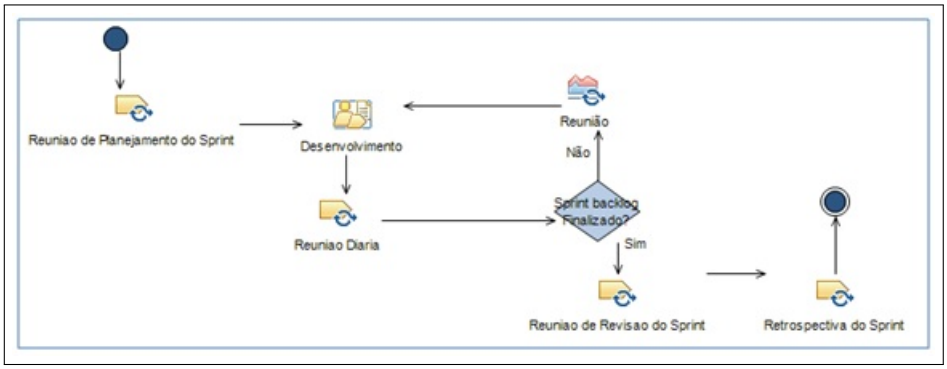


Figura 3.12 Fluxo da Atividade *Sprint* (adaptado de (SALES, 2017))

Desenvolvimento: nesta subatividade ocorre o desenvolvimento do projeto através das tarefas **Especificar Modelo de Negócio**, **Analisar Serviço** e a **Projetar Serviço**.

Esta subatividade contém as tarefas do conceito de DDM (*Modelagem, Transformações M2M e Transformações M2T*) e tarefas relacionadas SOA (*Analisar Serviço*). Desta forma, a tarefa *Modelagem* será realizada através da tarefa *Especificar Modelo de Negócio*. A tarefa *Transformação M2M* será realizada através da tarefa, *Analisar Serviço* e a ta-

refa *Transformação M2T* será realizada pela tarefa *Projetar Serviço*.

Especificar Modelos de Negócio: o objetivo desta tarefa é construir os modelos de analise através da subtarefa de *Modelar Negócio*.

Artefato Modelo de Funcionalidade (MF): o objetivo deste artefato é descrever o modelo de domínio de aplicação, representando as funcionalidades (casos de uso) de domínio e suas relações com os usuários (atores) do sistema. Este modelo é construído com base no conhecimento do domínio da aplicação, e a representação deste artefato é um *Diagrama de Caso de Uso* em UML.

Artefato Modelo de Informação de Negócio (MIN): o objetivo deste artefato é descrever o modelo o domínio de aplicação através dos conceitos de (entidades) e associações (relacionamentos) entre pares dessas entidades. A representação deste artefato é um *Diagrama de Classes* em UML, possibilitando, assim, a identificação da colaboração entre as classes.

Analisar Serviço: esta tarefa da subatividade da *Desenvolvimento* tem o objetivo de gerar os primeiros modelos da arquitetura do sistema.

Identificar Serviços: nesta subtarefa o Modelo de Funcionalidade (MF) e o Modelo de Informação de Negócio (MIN) são construídos/atualizados.

Refinar Serviço: nesta subtarefa tem o objetivo de gerar o Modelo de Iteração de Serviço (MIS), com a finalidade de identificar as capacidades dos serviços.

Identificar Componentes: esta subtarefa tem o objetivo de gerar o Modelo de Componentes dos Serviços (MCS).

Projetar Serviço: o objetivo desta tarefa é construir a arquitetura detalhada do sistema, possibilitando com esta arquitetura visualizar oportunidades de reuso e a geração de código com base nas informações do negócio.

Definir Arquitetura do Sistema: Esta subtarefa tem o objetivo gerar o modelo de arquitetura do sistema *ARQSYS*. Observe-se que quando inconformidades forem encontradas ou quando necessário, pode ser possível retornar a qualquer fase anterior. Portanto, quando a modificação for realizada nos modelos gerados anteriormente, os mesmos precisam ser re-gerados, ou seja, repetindo todo o processo.

Back-End: Esta subtarefa tem o objetivo de gerar o código fonte através de um modelo para uma ou varias plataformas, conforme definido na fase de *Pregame*. Ao final da execução desta subtarefa o software será entregue ao cliente em conformidade com as especificações realizadas no inicio do projeto.

3.5.1.3 Fase de *PostGame* O objetivo desta fase é entregar o software e sua documentação ao cliente. Estes devem esta em conformidade com as necessidades especificadas pelo cliente na fase de *Pregame* e desenvolvida ao decorrer da fase *Game*. Sendo assim, nesta fase o projeto é finalizado e é celebrado o encerramento do mesmo junto ao cliente.

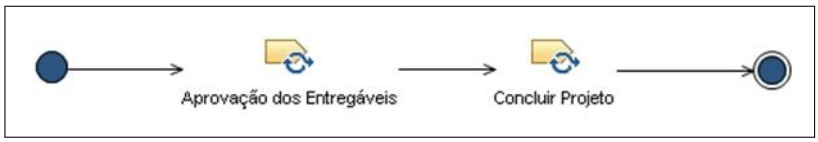


Figura 3.13 Fluxo da Atividade *Encerramento* (adaptado de (SALES, 2017))

Na figura 3.13 podemos observar o fluxo de execução da atividade *Encerramento* da fase *PostGame*.

3.5.2 Considerações da Seção

Em resumo, um processo hibrido proposto por (BRAGA, 2011) foi adaptado e chamado de ArqProjProcess(SALES, 2017) para que o mesmo pudesse ser uma prova de conceito à utilização/aplicação ao metaprocesso ScrumDDM (SALES, 2017). Para isso, (SALES, 2017) realizou uma nova hibridização agora com adição de práticas ágeis ao processo.

Na tabela 3.4, podemos observar um resumo do ArqProjProcess como processo adaptado por (SALES, 2017). Este resumo contém as fases, etapas, atividades,responsabilidades, artefatos de entrada e saída da instância do ScrumDDM.

Tabela 3.4 Resumo da instância adaptada ArqProjProcess

Atividades	SubAtividades	Tarefas	Responsável	Entradas	Saidas
Fase PréGame					
Visão VisãoArqProj	Requisitos	Reunião de planejamento da Sprint	ScrumDDMMaster Product Owner e Cliente	Estórias dos usuários e ou Caso de Uso	Metamodelos de dominio e Modelos Propostos
		Criar e Priorizar Produto backlog			
		Planejamento de versão			
		Escrever, Decompor e Estimar Estorias dos usuarios			
		Escrever, Executar teste de aceitação			
		Escrever e Atualizar Caso de Uso			
		Definir Metamodelo e Modelos			
Fase Game					
Sprint	Desenvolvimento	Especificar modelo de negócio, Analisar Serviços e Projetar Serviços	ScrumDDMTeam ScrumDDMMaster	Modelos, Mecanismo de extensão UML Perfil e Expressões OCL	Modelos, Documentação e Código Projeto Arquitetual
Fase PostGame					
Encerramento		Aprovar Entregáveis Concluir Projeto	ScrumDDMMaster Product Owner Cliente		Projeto Arquitetural do Software
					Documentação

3.6 CONSIDERAÇÕES SOBRE O CAPÍTULO

Em resumo, ScrumDDM (SALES, 2017), introduziu práticas oriundas DDM ao Scrum, buscando agregar as vantagens das duas abordagens sem desvirtuar os princípios e valores das mesmas.

Na Tabela 3.5, podemos observar um resumo do metaprocesso ScrumDDM (SALES, 2017). Este resumo contém as fases, etapas, atividades,responsabilidades, artefatos de entrada e saída do metaprocesso.

A instancia ArqprojProcess, foi utilizada neste trabalho com o objetivo de avaliar questões relacionadas ao desenvolvimento da evolução de software a partir de um software existente e desenvolvido com o metaprocesso ScrumDDM (SALES, 2017).

A nova instancia ScrumDDMQualitas, foi desenvolvida neste trabalho com o objetivo de avaliar a capacidade do metaprocesso ScrumDDM em instanciar novos processos de desenvolvimento de software.

Observamos nas instancias que as fases, atividades, subatividades e tarefas puderam ser integradas, assim como os papeis e responsabilidade. As adaptações necessárias em ambas as instancias não descaracterizaram os processos originais. Ao final da execução das instancias, o software e a documentação pode ser entregue ao cliente.

Tabela 3.5 Resumo do sobre o metaprocesso ScrumDDM

Atividades	SubAtividades	Tarefas	Responsável	Entradas	Saidas
Fase PréGame					
Visão	Requisitos Arquitetura	Reunião de Planejamento Sprint	ScrumDDMMaster *Product Owner* e Cliente	Estórias dos Usuários e ou Caso de Uso	Metamodelos de Dominio e Modelos Propostos
		Criar, Priorizar Produto backlog			
		Planejamento de versão			
		Escrever, Decompor, Estimar Estorias dos Usuários			
		Escrever, Executar Teste Aceitação			
		Escrever, Atualizar Caso de Uso			
		Definir Metamodelo, Modelos			
Fase Game					
Sprint	Desenvolvimento	Metamodelagem Modelagem	ScrumDDMTeam ScrumDDMMaster	Modelos	Modelos
		Transformação M2M, M2T		Mecanismo de extensão UML	
		Reunião Diária		Perfil	Código
		Reunião de Planejamento, Revisão, Restrospectiva da Sprint		Expressões OCL	
	Teste	Definir Testes Implantar Ambiente Executar Testes Inspecionar Código	ScrumDDMMaster, ScrumDDMTeam	Estória de Usuário Script de Teste Teste Aceitação Código	Teste Cliente Teste Ambiente Log de Teste
	Integração	Integrar e Construir	ScrumDDMTeam	Código	Checkout List
Fase PostGame					
Encerramento		Aprovar Entregáveis Concluir Projeto	ScrumDDMMaster ScrumDDMTeam Product Owner Cliente		Produto de Software Documentação

Capítulo

4

AVALIAÇÃO DO METAPROCESSO SCRUMDDM

O elemento central do ScrumDDM é a introdução de práticas de DDM ao método ágil Scrum que, apoiado em modelos e transformações, conduz o desenvolvimento de projetos ágeis em diferentes tecnologias. Objetiva assim que a documentação e o código fonte do software sejam gerados em conformidade ao que foi especificado pelo usuário, sem comprometer a velocidade no processo de desenvolvimento.

Processos de software são essencialmente difíceis de serem avaliados, por envolver: (i) pessoas com diferentes experiências, (ii) metodologias e (iii) ferramentas. Para avaliar o metaprocesso ScrumDDM utilizamos a instância ArqprojProcess, através da realização de um experimento controlado com o objetivo de avaliar sua capacidade de possibilitar a evolução de um software e a agilidade (esforço e velocidade) do desenvolvimento desta evolução. A avaliação realizada por (SALES, 2017) foi importante para verificarmos a viabilidade da utilização do metaprocesso na construção de um software, ou seja, se foi possível seguir o processo, gerar os artefatos intermediários e o código fonte do software. No entanto, a eficácia da documentação gerada pelo ScrumDDM no apoio a evolução de software e a agilidade (esforço e velocidade) no desenvolvimento não foram avaliados. Por isso, uma segunda avaliação foi planejada para o ScrumDDM contemplando um segundo experimento controlado.

Neste capitulo será descrito o experimento controlado (Seção 4.1) e as considerações finais do capítulo na (Seção 4.2).

4.1 EXPERIMENTO CONTROLADO

Esta seção apresenta o experimento controlado desenvolvido para avaliar ScrumDDM quanto: (i) a eficácia da documentação gerada pelo ScrumDDM no apoio a evolução de software e (ii) se sua utilização influenciou a agilidade (esforço e velocidade) do desenvolvimento em relação a evolução realizada com o Scrum. O experimento utilizou como base as seguintes etapas definidas por (WOHLIN et al., 2012): **Escopo** (Subseção 4.1.1), **Planejamento** (Subseção 4.1.2), **Operação do Experimento** (Subseção 4.1.3), e **Análise e Interpretação dos Dados** (Subseção 4.1.4).

4.1.1 Escopo

Na etapa de escopo são definidos o objetivo do experimento, suas respectivas questões de pesquisa e suas métricas. Para apoiar essa definição utilizamos a técnica de *Goal-Question-Metric*(GQM) proposta por (CALDIERA; ROMBACH, 1994).

4.1.1.1 Objetivo do Experimento. Neste experimento buscamos avaliar o metaprocesso ScrumDDM quanto a sua capacidade de possibilitar a evolução de um software e a agilidade do desenvolvimento. Para isto utilizamos um software já existente, desenvolvido previamente também através do ArqProjProcess em (SALES, 2017). Definimos e especificamos uma nova versão para o software, contemplando novas estórias do usuário e /ou modificações nas estórias existentes.

Para avaliar a eficácia da documentação gerada pelo ScrumDDM no apoio a evolução de um software existente já desenvolvido através do ScrumDDM, analisamos o código fonte gerado pelo ScrumDDM em relação ao código fonte gerado usando somente o processo Scrum no que se refere aos atributos de *Corretude* e *Completude* no código. Para avaliar a agilidade na manutenibilidade das evoluções solicitadas para a nova versão, o atributo de *Tempo* em minutos foi utilizado. A Figura 4.1 apresenta a definição do objetivo do experimento de acordo com o *template* GQM (CALDIERA; ROMBACH, 1994)

Analisar o código desenvolvido ao longo de uma Sprint .
Com propósito de avaliar manutenibilidade e a agilidade do processo de software.
Com respeito aos atributos corretude, completude e o tempo de desenvolvimento das estórias dos usuários definidas em uma determinada *sprint*.
Do ponto de vista do método de desenvolvimento utilizado.
No contexto de estudantes de graduação, pós-graduação e profissionais da área de tecnologia.

Figura 4.1 Objetivo do experimento para avaliação do metaprocesso ScrumDDM

4.1.1.2 Questões de Pesquisa. Após a definição do objetivo principal do experimento, as questões de pesquisa a seguir foram estabelecidas:

- **Q1:** Os artefatos gerados pelo ScrumDDM influenciam a corretude do desenvolvimento corretivo/evolutivo das estórias dos usuários em relação ao desenvolvimento realizado diretamente no código?
  Esta questão busca investigar se o ScrumDDM influenciou na quantidade de estórias dos usuários desenvolvidas corretamente em relação as mesmas estórias desenvolvidas somente usando o Scrum.

- **Q2:** Os artefatos gerados pelo ScrumDDM influenciam a completude do desenvolvimento corretivo/evolutivo das estórias dos usuários em relação ao desenvolvimento realizado diretamente no código?
  Esta questão busca investigar se o ScrumDDM influenciou na quantidade de estórias dos usuários desenvolvidas de forma completa em relação as mesmas estórias desenvolvidas com o Scrum.

- **Q3:** Os artefatos gerados pelo ScrumDDM influenciam no tempo de desenvolvimento corretivo/evolutivo através das estórias dos usuários em relação ao desenvolvimento realizado diretamente no código através das mesmas estórias dos usuários? Esta questão busca investigar se o ScrumDDM influenciou na agilidade do desenvolvimento quando comparado com a agilidade do desenvolvimento realizado com o Scrum.

4.1.1.3 Métricas Para avaliar a capacidade do ScrumDDM em evoluir um software e a agilidade no desenvolvimento dessas evoluções, definimos três atributos *Corretude, Completude* e *Tempo.*

Corretude: atributo utilizado para medir o número de estórias dos usuários desenvolvidas de forma correta. O desenvolvimento da evolução é considerado correto se o quociente da divisão realizada entre o número de histórias implementadas e o número total de histórias a serem evoluídas for: (a) mais próximo de 1 ou (b) igual a 1.

- (a) O desenvolvedor não realizou a evolução toda corretamente;
- (b) O desenvolvedor realizou a evolução toda corretamente;

Completude: atributo utilizado para medir o número de estórias dos usuários desenvolvidas de forma completa. O desenvolvimento é considerado completo se todas as histórias de usuários forem implementadas. Nesse caso, consideramos o valor de completude como 1, caso contrário, é 0

Tempo: medido de acordo com o tempo (minutos) de desenvolvimento do conjunto das estórias dos usuários definidas para evolução do software.

4.1.2 Planejamento do Experimento

Nesta etapa é realizada a definição do contexto, o *design* (Tipo do experimento), a formulação das hipóteses, a seleção (definição) das variáveis dependentes e independentes e a instrumentação.

4.1.2.1 Definição do Contexto. O experimento controlado foi conduzido em ambiente acadêmico composto por alunos de graduação e pós-graduação em cursos relacionados à ciência da computação e profissionais da área de desenvolvimento de software. Neste experimento, além de conhecimento em UML e na linguagem de programação JAVA

(Eclipse) foi requerido dos participantes um conhecimento básico sobre DDM. Para garantir que os alunos tinham estes requisitos, foi aplicado um questionário (apêndice D).

Dois estudos, um piloto e outro principal, foram planejados e executados. O estudo piloto serviu para verificar se o material e o tempo disponibilizado aos participantes do estudo principal estavam adequados. O estudo principal é o próprio experimento controlado reestruturado através dos resultados obtidos do estudo piloto.

4.1.2.2 *Design*/Tipo do Experimento. O tipo do *design* utilizado neste experimento compreendeu um fator, *o método de desenvolvimento* e dois tratamentos: a abordagem do ScrumDDM e a abordagem Scrum. Desta forma o experimento foi realizado por dois grupos organizados da seguinte forma: *Grupo ScrumDDM*, para o desenvolvimento das evoluções do software usando o metaprocesso ScrumDDM; e *Grupo Dirigido*, para o desenvolvimento das evoluções do software usando Scrum.

4.1.3 Planejamento: Hipóteses

Com o objetivo de avaliar os atributos definidos, foram formuladas hipóteses correspondentes a cada atributo. Para isso, dois grupos de hipóteses foram definidas, as hipóteses nulas e as hipóteses alternativas. A hipótese nula (HP0) indica que o resultado dos atributos para ambos os grupos apresentaram o mesmo valor. A hipótese alternativa indica que as medias alcançadas em cada atributo são diferentes. Com o objetivo de facilitar o entendimento e analise a hipótese alternativa foi dividida em duas sub-hipóteses, chamamos de (HP1), quando resultado do Grupo ScrumDDM for superior ao resultado do Grupo Dirigido e a chamamos de (HP2) quando o resultado do Grupo ScrumDDM for inferior ao resultado do Grupo Dirigido.

4.1.3.1 Formulação das Hipóteses. Para conduzir a avaliação foram formuladas hipóteses correspondentes a cada atributo (métrica) avaliada. Desta forma três grupos de hipóteses foram definidos: A hipótese nula (HP0) indica que a média do atributo, para ambos os grupos analisados é a mesma. A hipótese alternativa (HPn) indica que as médias alcançadas em cada grupo são diferentes. Com o objetivo de melhorar o entendimento a hipótese alternativa foi dividida em duas sub hipóteses: (HP1) quando a média do Grupo ScrumDDM for superior a média do Grupo Dirigido; (HP2) quando a média do Grupo ScrumDDM for inferior a média Grupo Dirigido.

- Avaliando o atributo **Corretude**:

  Hipótese Nula: A utilização do metaprocesso ScrumDDM não influenciou no nível de correção do código gerado na manutenção (evolução) das estórias.

  (HP0) Grupo ScrumDDM = Grupo Dirigido

  Hipóteses Alternativas: A utilização do metaprocesso ScrumDDM influenciou no percentual médio de acertos da manutenção (evolução) das estórias, gerando resultado da conformidade diferente de 1.

(HP1) Grupo ScrumDDM >Grupo Dirigido
(HP2) Grupo ScrumDDM <Grupo Dirigido

- Avaliando o atributo **Completude**:

  Hipótese Nula: A utilização do metaprocesso ScrumDDM não influenciou o nível de completude do código da manutenção (evolução) das estórias dos usuários.

  (HP0) Grupo ScrumDDM = Grupo Dirigido

  Hipóteses Alternativas: A utilização do metaprocesso ScrumDDM influenciou no percentual médio das estórias dos usuários desenvolvidas de forma completa, gerando resultado da conformidade diferente de 1.

  (HP1) Grupo ScrumDDM >Grupo Dirigido
  (HP2) Grupo ScrumDDM <Grupo Dirigido

- Avaliando o atributo **Tempo** de execução (desenvolvimento):

  Hipótese Nula: A utilização do metaprocesso ScrumDDM não influenciou no prazo de desenvolvimento da manutenção (evolução) dos requisitos.

  (HP0) Grupo ScrumDDM = Grupo Dirigido

  Hipóteses Alternativas: A utilização do metaprocesso ScrumDDM influenciou o prazo de desenvolvimento da manutenção/evolução dos requisitos.

  (HP1) Grupo ScrumDDM >Grupo Dirigido
  (HP2) Grupo ScrumDDM <Grupo Dirigido

4.1.3.2 Seleção/definição das Variáveis. Em um experimento existem dois tipos de variáveis, as variáveis independentes e as variáveis dependentes. As variáveis independentes são variáveis que podem ser controladas (modificadas) ao decorrer do experimento e as variáveis dependentes são as variáveis observadas para medir o efeito do tratamento (WOHLIN et al., 2012).

Neste experimento a variável independente consistiu na abordagem de desenvolvimento que assumiu dois níveis: o desenvolvimento das evoluções através do metaprocesso ScrumDDM e o desenvolvimento das evoluções diretamente através do Scrum. As variáveis dependentes foram os atributos (métricas) *Corretude*, *Completude* e *Tempo*, diretamente relacionadas ás medidas para teste das hipóteses.

4.1.3.3 Instrumentação. O cenário utilizado como objeto de estudo foi o mesmo para ambos os grupos e consistiu no desenvolvimento da evolução de um software no domínio de conciliação de cartão de crédito, definido pelo pesquisador, onde o software deveria ser modificado considerando as estórias dos usuários na versão. Para execução do experimento controlado foram disponibilizados alguns documentos. Nas figuras 4.2 e 4.3 podemos observar os documentos disponibilizados para cada grupo respectivamente.

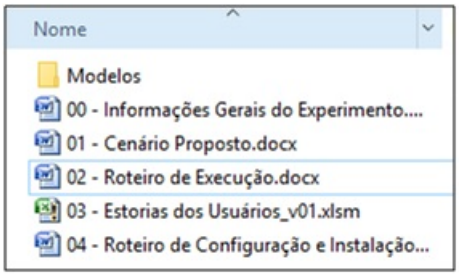


Figura 4.2 Documentos disponibilizados para o grupo ScrumDDM.

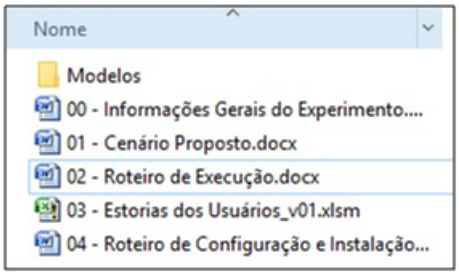


Figura 4.3 Documentos disponibilizados para o grupo Dirigido.

Conforme figura 4.2, foi disponibilizado ao *Grupo ScrumDDM* um documento contendo as informações sobre a execução do experimento (00 - Informações Gerais do Experimento.docx), o cenário com o domínio de negócio proposto (01 - Cenário Proposto.docx), o roteiro de execução para evolução das estórias dos usuários no metaprocesso ScrumDDM (02 - Roteiro de Execução.docx), um documento contendo as estórias dos usuários (03 - Estórias dos Usuários_v01.xlsm), um documento com as configurações das transformações (04 - Roteiro de Configuração e Instalação.docx) e os modelos do projeto arquitetural da versão inicial (Modelos). Conforme figura 4.3 foi disponibilizado ao *Grupo Dirigido* um documento contendo o cenário com o domínio de negócio (01 - Cenário Proposto.docx), o roteiro de execução para evolução das estórias dos usuários diretamente no código contendo informações sobre o que deveria ser evoluído no projeto arquitetural existente (02 - Roteiro de Execução.docx), um documento contendo as estórias dos usuários (03 - Estórias dos Usuários_v01.xlsm). Foi disponibilizado também, o ambiente (JAVA) Eclipse contendo o modelo de funcionalidade representado por um diagrama de caso de uso e o modelo de informações do negocio representado por um diagrama de classes, ambos na versão inicial.

O experimento foi conduzido em três fases. Fase 1: apresentação de informações gerais do experimento para os participantes. Fase 2: marcação do tempo utilizado na execução do experimento controlado e acompanhamento da execução do experimento controlado. Fase 3: avaliação sobre a evolução do software.
Na (*Fase 1*) o pesquisador apresentou o experimento controlado aos participantes de ambos os grupos. Na (*Fase 2*) os participantes de ambos os grupos realizaram a marcação

inicio/fim do tempo utilizado na execução do experimento controlado e o pesquisador executou a tarefa de acompanhar a execução do experimento controlado. Na (*Fase 3*) a avaliação sobre a evolução do software foi realizada considerando três aspectos: (i) a quantidade de estórias dos usuários desenvolvidas corretamente; (ii) a quantidade de estórias dos usuários desenvolvidas de forma completa e (iii) a quantidade de tempo em minutos utilizados na execução do experimento controlado. Os três aspectos citados foram avaliados para ambos os grupos de participantes.

4.1.4 Operação do experimento.

Antes de começar a executar o experimento controlado (estudo principal), foi necessário realizar um estudo piloto para identificar possíveis falhas que impactasse na execução do estudo principal. Com a execução do estudo piloto foi possível realizar correções das falhas encontradas na documentação e na apresentação realizada aos participantes. Posteriormente foi iniciado o estudo principal.

4.1.4.1 Execução do Estudo Piloto: Para participar do estudo piloto foram selecionados dois participantes, um aluno do mestrado do PGCOMP e outro profissional da área de tecnologia ambos da UFBA; Um representou o *Grupo ScrumDDM* e o outro representou o *Grupo Dirigido*, o cenário foi o mesmo para ambos os participantes. O representante do *Grupo ScrumDDM* não conhecia os modelos da instancia ArqProjProcess, assim como, o representante do *Grupo Dirigido* não conhecia o código da versão inicial do software nem os modelos disponibilizados na proposta.

O estudo piloto foi realizado no mês de Setembro de 2018 com duração de (2 horas e meia) de execução para cada participante. Para os dois participantes foi realizada uma apresentação sobre, o cenário de negócio, o ambientes especifico de desenvolvimento para cada grupo e sobre os documentos entregues. Foi disponibilizado para ambos os participantes do estudo piloto, um ambiente pré-configurado e um roteiro específico para cada grupo, e para ambos os grupos foi disponibilizado o mesmo cenário de negócio e o *Product Backlog* para evolução do projeto arquitetural do software.

A realização do estudo piloto teve relevância, pois detectou a necessidade das seguintes melhorias:

- *Melhorar a apresentação* para passar conhecimento sobre o cenário de negócio "Software de Conciliação de Cartão";
- *Elaborar um tutorial* contendo a ordem de leitura e o objetivo para cada documento disponibilizado;
- *Rescrever as estórias dos usuários* para evitar ambiguidade e melhorar a compreensão do cenário de negócio e das estórias dos usuários disponibilizadas;

4.1.4.2 Preparação do Estudo Principal. Após a execução do estudo piloto e dos ajustes citados na secção anterior, o experimento controlado foi realizado através da

execução das seguintes tarefas: (i) Seleção dos participantes e a disponibilização da documentação do experimento a cada participante; (ii) Execução do experimento controlado; (iii) Coleta dos dados e (vi) Validação dos dados coletados.

4.1.4.3 Seleção dos Participantes para o Experimento. Na seleção dos participantes foi utilizada uma amostragem de conveniência, na qual os participantes foram escolhidos por possuírem características desejáveis ao estudo, conforme relatado na subseção(4.1.2.1 - Definição do Contexto). Buscou-se identificar o nível de experiência e conhecimento que os possíveis participantes possuíam em UML, DDM e JAVA. A seguir, a figura representa o conhecimento dos participantes nos assuntos relacionados ao tema do estudo.

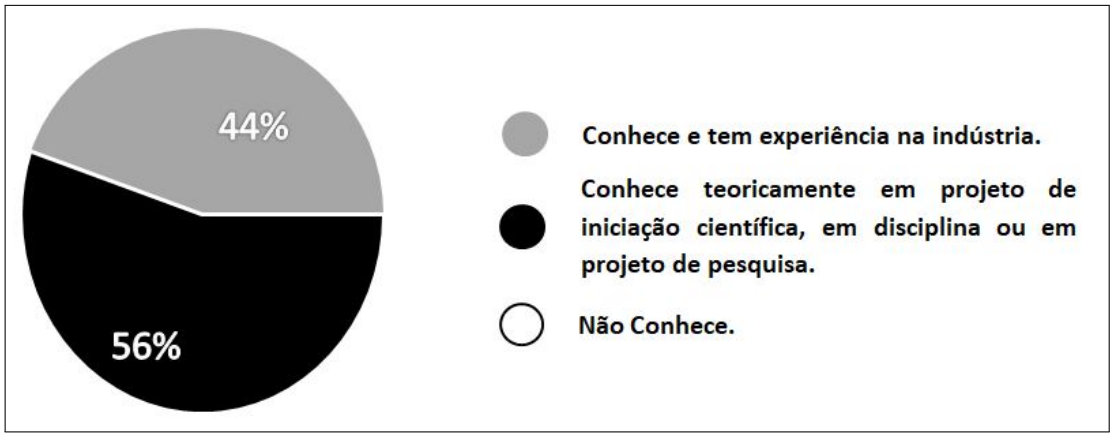


Figura 4.4 Conhecimento dos participantes em UML.

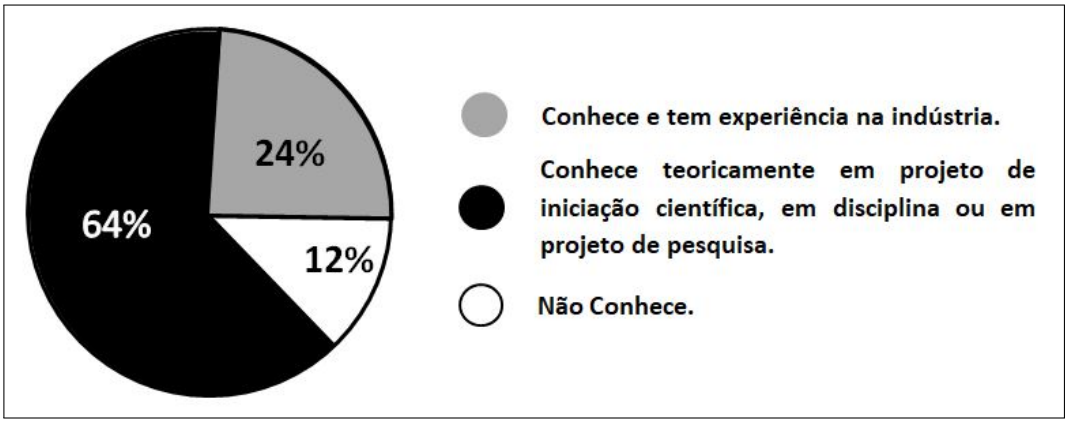


Figura 4.5 Conhecimento dos participantes em JAVA

Podemos observar nas figuras 4.4, 4.5 e 4.6, o nível e a forma de conhecimento dos participantes sobre os assuntos relacionados ao tema do estudo. Nota-se que a maioria

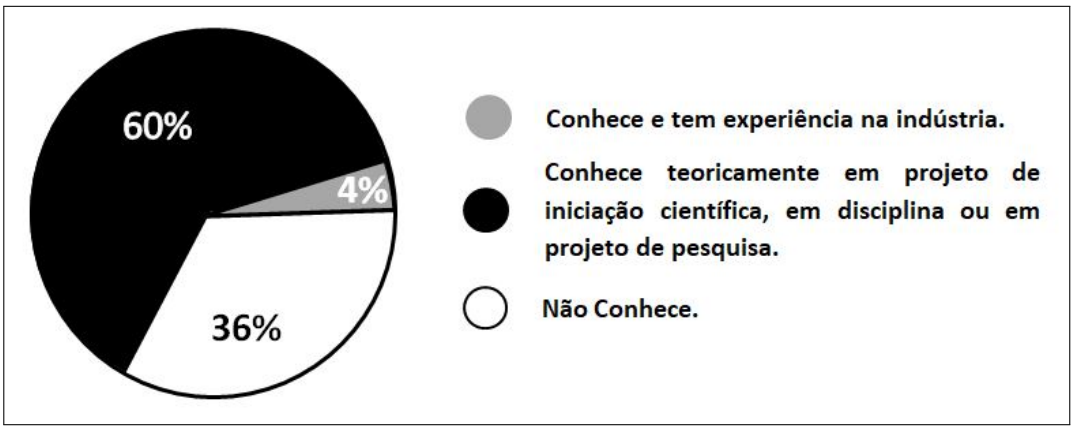


Figura 4.6 Conhecimento dos participantes em DDM

dos participantes obtiveram conhecimento devido a sua participação em projeto de iniciação científica, em disciplina ou em projeto de pesquisa.

O estudo foi realizado em ambiente acadêmico, com alunos de graduação e pós-graduação e alguns profissionais de tecnologia selecionados de acordo com um perfil pré-definido. Um grupo de 25 pessoas responderam o questionário de seleção para participar do experimento. Entre eles, 20 participantes foram selecionados e divididos aleatoriamente em dois grupos de 10 participantes cada, conforme sua disponibilidade em participar do experimento e em seu conhecimento/experiência sobre (4.1.2.1 - Definição do Contexto). Os demais não possuíam o conhecimento mínimo necessário para a realização do estudo ou estavam indisponíveis.

4.1.4.4 Execução: O experimento foi realizado entre os meses de Outubro e Dezembro de 2018 com duração média de 55 (minutos) de execução para o Grupo Dirigido e 38 (minutos) de execução para o Grupo ScrumDDM. Assim como no estudo piloto, conciliar o dia e o horário para realizar o experimento controlado e a disponibilidade dos participantes foi uma dificuldade encontrada.

Para os dois grupos a execução do experimento aconteceu em várias sessões distintas tanto no laboratório da UFBA quanto no laboratório da UNEB. O pesquisador supervisionou a execução do experimento para ambos os grupos.

Somente após as evoluções do produto de software a avaliação teve inicio, pois foi necessário aguardar até que todos os participantes indicassem que tinham terminado as evoluções.

4.1.4.5 Coleta dos Dados: Os dados coletados através do experimento foram utilizados como entrada de dados para analise do experimento. Os primeiros dados coletados durante o estudo piloto permitiram fazer ajustes para execução do experimento. Contudo,

esses dados, para efeito de análise, foram descartados. Desta forma, utilizamos apenas os dados resultantes da execução do experimento controlado (estudo principal).Utilizamos para avaliar os dados do experimento o barema da Tabela 4.1, contendo os resultados esperados após a evolução do produto software resultante do desenvolvimento de estória do usuário por cada participante.

Tabela 4.1 Barema de avaliação dos resultados obtidos após a conclusão do experimento por cada participante.

Barema						
A	B	C	D	E	F	G
6	4	4	4	4	8	10

No barema da Tabela 4.1, as colunas (A) até (G) representam as estórias evoluídas ou modificadas. Os valores atribuídos a estas colunas representam o numero de modificações esperadas com o desenvolvimento de cada estória no software existente. Por exemplo, buscou-se no desenvolvimento da estória do usuário representada pela coluna (D) (Executar_Conciliacao_Bancaria) que 4 (quatro) modificações fossem realizadas. O detalhe das estórias dos usuários estão disponíveis no (apêndice A).

Tabela 4.2 Dados coletados na avaliação do atributo corretude.

	Grupo	PARTICIPANTE	Estórias dos Usuários Evoluídas						
			A	B	C	D	E	F	G
ATRIBUTO CORRETUDE	Dirigido	1	0,33	1,00	0,50	1,00	0,50	1,00	1,00
		2	0,33	0,50	0,50	0,50	0,50	0,50	0,50
		3	0,33	0,50	0,50	0,50	0,50	0,25	0,00
		4	0,83	1,00	1,00	0,75	1,00	1,00	1,00
		5	1,00	0,00	0,50	0,50	0,50	1,00	0,50
		6	0,67	0,50	0,50	0,00	0,50	0,50	0,50
		7	0,67	0,50	0,75	0,50	1,00	0,50	0,20
		8	1,00	1,00	1,00	0,50	0,50	0,38	0,40
		9	1,00	1,00	1,00	1,00	1,00	0,75	0,80
		10	1,00	1,00	0,50	1,00	1,00	0,75	0,80
	ScrumDDM	11	0,67	1,00	0,50	1,00	1,00	1,00	1,00
		12	0,33	0,50	1,00	1,00	1,00	1,00	1,00
		13	1,00	1,00	1,00	1,00	1,00	1,00	1,00
		14	0,83	0,50	0,50	0,50	0,50	1,00	1,00
		15	1,00	1,00	0,50	1,00	0,50	0,75	0,80
		16	0,33	0,50	1,00	1,00	1,00	1,00	0,80
		17	1,00	1,00	1,00	1,00	1,00	1,00	0,80
		18	1,00	1,00	1,00	1,00	0,50	1,00	1,00
		19	0,17	0,50	1,00	1,00	1,00	0,75	0,80
		20	1,00	1,00	1,00	1,00	1,00	1,00	1,00

A Tabela 4.2 representa a tabela contendo os dados coletados na avaliação do atributo corretude. A primeira coluna da tabela indica qual o atributo coletado. A segunda coluna representa o grupo de participantes. As demais colunas representam as estórias dos usuários e os valores obtidos no desenvolvimento dessas estórias segundo o atributo corretude. Os valores obtidos no desenvolvimento de cada estória do usuário é o resultado da seguinte operação: numero de modificações atingidas pelo participante dividido pelo numero de modificações esperadas. Por exemplo, para estória do usuário representado na coluna (A) *Registrar Venda* um participante do *Grupo Dirigido* realizou 2 (duas) modificações, e o esperado eram 6 (seis) modificações, logo ele obteve como resultado o valor (0,33).

Tabela 4.3 Dados coletados na avaliação do atributo completude.

ATRIBUTO COMPLETUDE	Grupo	PARTICIPANTE	Estórias dos Usuários Evoluídas						
			A	B	C	D	E	F	G
	Dirigido	1	0,00	1,00	0,00	1,00	0,00	1,00	1,00
		2	0,00	0,00	0,00	0,00	0,00	0,00	0,00
		3	0,00	0,00	0,00	0,00	0,00	0,00	0,00
		4	1,00	1,00	1,00	0,00	1,00	1,00	1,00
		5	1,00	0,00	0,00	0,00	0,00	1,00	0,00
		6	0,00	0,00	0,00	0,00	0,00	0,00	0,00
		7	0,00	0,00	0,00	0,00	1,00	0,00	0,00
		8	1,00	1,00	1,00	0,00	0,00	0,00	0,00
		9	1,00	1,00	1,00	1,00	1,00	0,00	0,00
		10	1,00	1,00	0,00	1,00	1,00	0,00	0,00
	ScrumDDM	11	1,00	1,00	0,00	1,00	1,00	1,00	1,00
		12	0,00	0,00	1,00	1,00	1,00	1,00	1,00
		13	1,00	1,00	1,00	1,00	1,00	1,00	1,00
		14	0,00	0,00	0,00	0,00	0,00	1,00	1,00
		15	1,00	1,00	0,00	1,00	0,00	0,00	1,00
		16	0,00	0,00	1,00	1,00	1,00	1,00	0,00
		17	1,00	1,00	1,00	1,00	1,00	1,00	0,00
		18	1,00	1,00	1,00	1,00	0,00	1,00	1,00
		19	0,00	0,00	1,00	1,00	1,00	0,00	0,00
		20	1,00	1,00	1,00	1,00	1,00	1,00	1,00

A Tabela 4.3 representa a tabela contendo os dados coletados na avaliação do atributo completude. A primeira coluna da tabela indica qual o atributo coletado. A segunda coluna representa o grupo de participantes. As demais colunas representam as estórias dos usuários e os valores obtidos no desenvolvimento dessas estórias segundo o atributo completude. Os valores obtidos no desenvolvimento de cada estória do usuário é o resultado da seguinte operação: se o participante realizou todas as modificações o valor obtido será (1,00); se não realizou, o valor é (0,00). Por exemplo, para estória do usuário representado na coluna (A) *Registrar Venda* um participante do *Grupo Dirigido* não conseguiu completar o desenvolvimento, logo ele obteve o valor (0,00). Porém na estória (B) *Verificar Venda ADM* ele conseguiu completar o desenvolvimento e obteve o valor (1,00).

4.1.4.6 Validação dos Dados. Nenhuma inconsistência foi encontrada, portanto, todas as respostas foram utilizadas na análise dos dados.

4.1.5 Análise e Interpretação dos Dados

Nesta etapa são apresentadas as análise dos dados coletados para que se possam tirar conclusões sobre o estudo. Para cada atributo, grupo de participantes e estórias dos usuários são apresentados: *o mínimo valor observado* (Min), *o primeiro quartil* (Q1), *a mediana, a média, o terceiro quartil* (Q3), *o máximo valor observado* (Max), *o desvio padrão* (SD) *e o coeficiente de variação* (CV). Conforme podemos observar na Tabela 4.4.

Tabela 4.4 Apresentação da tabela contendo as medidas para os atributos *(Corretude, Completude e Tempo)* em relação aos grupos de participantes *(ScrumDDM e Dirigido).*

ATRIBUTO	Grupo	Medidas Descritivas Calculadas Grupo							
		Min	Q1	Mediana	Média	Q3	Max	SD	CV%
CORRETUDE	Dirigido	0,37	48%	64%	66%	86%	0,94	21%	32%
	ScrumDDM	0,69	79%	86%	86%	97%	1,00	11%	13%
COMPLETUDE	Dirigido	0,00	0%	36%	36%	57%	0,86	32%	89%
	ScrumDDM	0,29	57%	79%	72%	86%	1,00	24%	33%
TEMPO (Em Minutos) Atualização parcial da Documentação	Dirigido	35	43	52	55	68	87	17	31%
	ScrumDDM	12	27	42	38	48	64	17	45%
TEMPO (Em Minutos) Atualização total da Documentação	Dirigido	35	43	52	55	68	87	17	31%
	ScrumDDM	20	43	59	59	77	87	21	36%

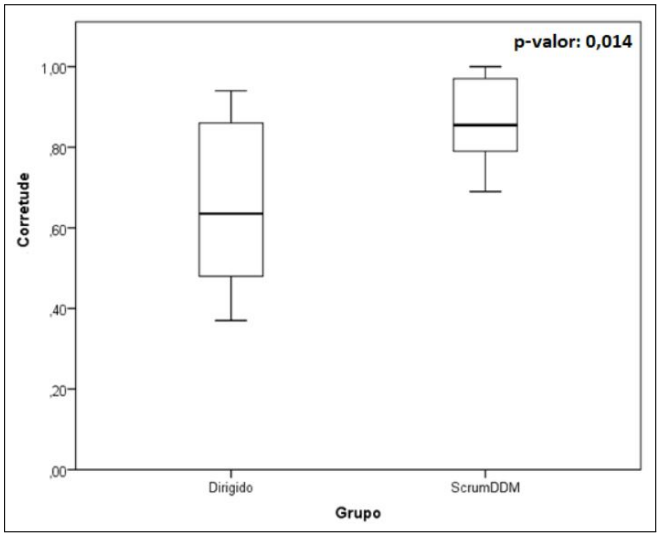


Figura 4.7 Comparação entre Grupo ScrumDDM e o Grupo Dirigido para o atributo *Corretude*

Para o **Atributo Corretude**, a medida descritiva *média* das estórias desenvolvidas de forma correta de cada grupo indica que o desenvolvimento realizado pelo Grupo ScrumDDM (mediana= média= 86%) teve mais estórias desenvolvidas de forma correta em relação as estórias desenvolvidas pelo Grupo Dirigido (mediana=64% ; média= 66%), correspondente à hipótese HP1 (Percentual médio do atributo de *Corretude* para o Grupo ScrumDDM foi maior que o percentual médio do atributo de *Corretude* para o Grupo Dirigido). Isso se confirma, se analisarmos o *coeficiente de variação (CV)*, uma medida de dispersão que considera tanto a média quanto o desvio padrão, observamos que as estórias desenvolvidas pelo Grupo ScrumDDM (13%) tiveram uma variação menor do que as estórias desenvolvidas pelo Grupo Dirigido (32%). Essa variação pode ser vista também no gráfico boxplot da Figura 4.7, pois a amplitude do box do Grupo Dirigido foi maior que o do Grupo ScrumDDM. Se compararmos a mediana dos dois grupos, observamos que o desenvolvimento realizado pelo Grupo ScrumDDM também foi superior ao do Grupo Dirigido, confirmando uma melhor concisão da evolução das estórias dos usuários pelo Grupo ScrumDDM. No Grupo Dirigido, 75% dos participantes obtiveram um numero de acertos abaixo de 86% em relação aos acertos obtidos pelos participantes do Grupo ScrumDDM que superaram este valor.

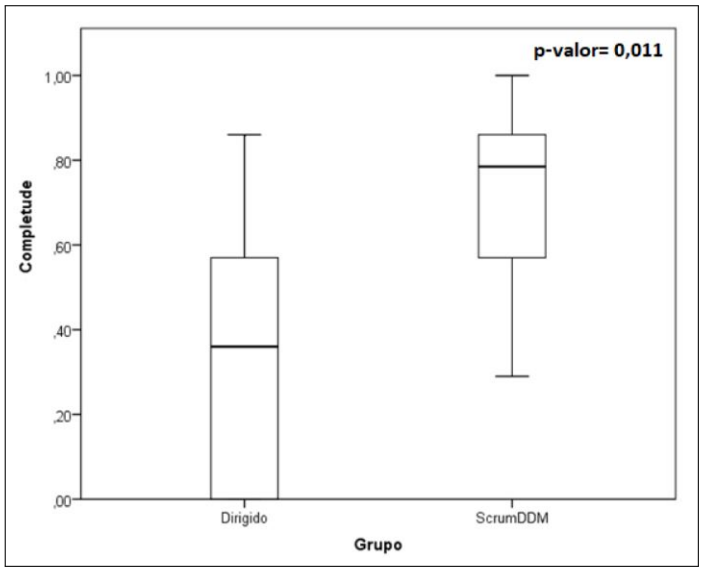


Figura 4.8 Comparação entre Grupo ScrumDDM e o Grupo Dirigido para o atributo *Completude*

Para o **Atributo Completude**, a medida descritiva *média* das estórias, desenvolvidas de forma completa de cada grupo, indica que o desenvolvimento realizado pelo Grupo ScrumDDM (mediana= 79% ; média= 72%) teve mais estórias desenvolvidas de forma completa em relação as estórias desenvolvidas pelo Grupo Dirigido (mediana= média= 36%), correspondente à hipótese HP1 (O percentual médio das estórias dos usuários desenvolvidas de forma completa do Grupo ScrumDDM foi maior que o percentual médio das estórias dos usuários desenvolvidas de forma completa do Grupo Dirigido). Isso se

confirma, se analisarmos o *coeficiente de variação (CV)*, observamos que as estórias desenvolvidas pelo Grupo ScrumDDM (33%) tiveram uma variação menor que as estórias desenvolvidas pelo Grupo Dirigido (89%). Esta menor variação que o Grupo ScrumDDM obteve pode indicar que os resultados deste grupo foram mais homogêneos que os resultados do Grupo Dirigido. Logo a menor variabilidade do Grupo ScrumDDM pode evidenciar que o mesmo obteve um comportamento mais padronizado em relação ao Grupo Dirigido.

O **Atributo Tempo de Desenvolvimento** foi analisando considerando dois aspectos: (i) Quando atualizada somente a documentação (Diagrama de Classes) que gera o código fonte do projeto a ser evoluído e (ii) Quando atualizado toda a documentação (Diagrama de Caso de Uso, Diagrama de Arquitetura, Diagrama de Sequencia, Diagrama de Componente, Diagrama de Arquitetura de Sistemas), além da documentação que gera o código fonte do projeto a ser evoluído.

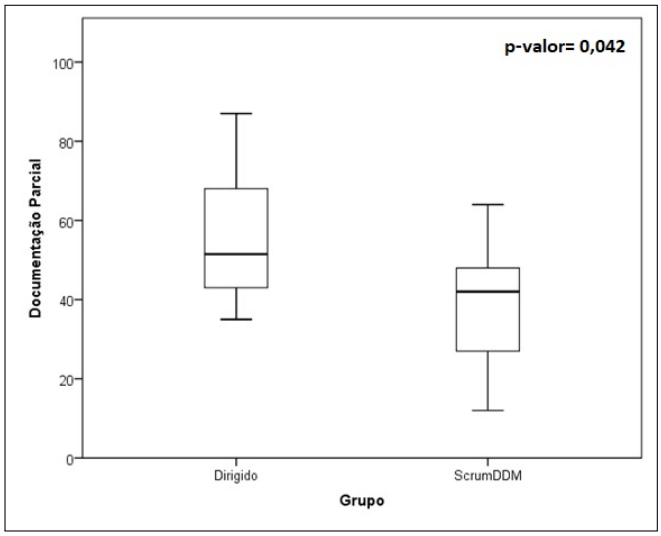


Figura 4.9 Avaliação do atributo *Tempo*, Quando atualizada somente a parte da documentação que geram o código fonte do projeto a ser evoluído.

(i) Quando atualizada somente a parte da documentação que gera o código fonte do projeto a ser evoluído, as medidas de tendência central (*mediana* e *média*) para o atributo *Tempo* de desenvolvimento/evolução das estórias dos usuários desenvolvidas por cada grupo indica que o Grupo ScrumDDM (mediana= 42 ; média= 38 minutos) realizou a evolução do projeto de forma mais rápido/ágil em relação ao Grupo Dirigido (mediana= 52 ; média= 55 minutos), o que corresponde à hipótese (HP2). O tempo utilizado na evolução do software pelo Grupo ScrumDDM foi menor que o tempo utilizado pelo Grupo Dirigido. Contudo se analisarmos o *coeficiente de variação (CV)* no gráfico *boxplot* da figura 4.9, observamos que o tempo de desenvolvimento do Grupo ScrumDDM(43%) apresenta uma maior variabilidade em relação ao tempo de desenvolvimento do Grupo Dirigido (31%). O Grupo ScrumDDM apresentou uma assimetria a esquerda, fato explicado pelos menores tempos obtido por metade dos participantes deste

grupo que foram mais ágeis que (75%) dos participantes do Grupo Dirigido.

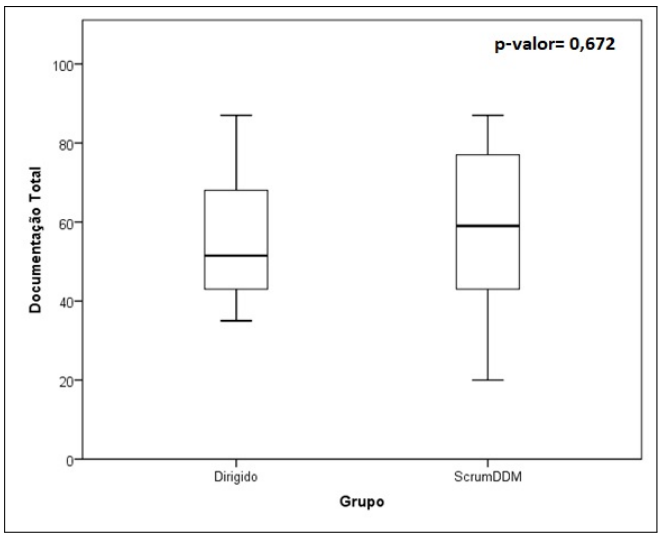


Figura 4.10 Avaliação do atributo *Tempo*, quando atualizada toda a documentação (modelos), além da documentação que gera o código fonte do projeto a ser evoluído.

(ii) Quando atualizada toda a documentação (modelos), além da documentação que gera o código fonte do projeto a ser evoluído, as medidas de tendência central revelou que o Grupo Dirigido (mediana= 52; média= 55 minutos) apresentou menor tempo para o desenvolvimento das evoluções solicitadas em relação ao tempo de desenvolvimento utilizado pelo Grupo ScrumDDM (mediana= média= 59 minutos), o que corresponde à hipótese (HP1). O tempo de desenvolvimento utilizado pelo Grupo ScrumDDM foi maior que o tempo utilizado pelo Grupo Dirigido. O Grupo Dirigido apresentou uma menor variabilidade, possuindo um coeficiente de variação de (31%), enquanto o Grupo ScrumDDM apresentou um coeficiente de variação de (36%). Podemos observar que mesmo o Grupo ScrumDDM ter utilizado mais tempo e ter variado mais, não significou que este aumento de tempo (4 minutos) influenciou na velocidade na utilização do metaprocesso podemos considerar que toda a documentação do projeto, assim como o código fonte, foram atualizados.

Em resumo, podemos perceber uma melhor avaliação do Grupo ScrumDDM tanto nos atributos avaliados Corretude, Completude. No atributo Tempo dois fatores foram observados: (i) Quando somente a documentação para transformação M2T (Diagrama de Classes para código) foi atualizada, o tempo de desenvolvimento utilizado pelo Grupo ScrumDDM foi menor que o tempo de desenvolvimento utilizado pelo Grupo Dirigido e (ii) quando além do documento (Diagrama de classes), os documentos (Diagrama de Caso de Uso, Diagrama de Arquitetura, Diagrama de Sequencia, Diagrama de Componente e Diagrama de Arquitetura de Sistemas) foram atualizados, foi possível perceber que o Grupo ScrumDDM utilizou mais tempo para desenvolver as estórias dos usuários e atualizar a documentação em relação ao Grupo Dirigido. Porém, este aumento de tempo pelo

Grupo ScrumDDM, não pode ser tido como significativo. Desta forma a característica de agilidade da metodologia ágil foi preservada no metaprocesso ScrumDDM.

4.1.5.1 Avaliação das Hipóteses: Com o objetivo de verificar a existência de diferença significativa no código desenvolvido pelos dois grupos de participante do estudo, foi executado o teste de hipótese (WOHLIN et al., 2012) através do *Student´s t-test*.

Tabela 4.5 Dados coletados na análise dos atributos.

ATRIBUTO	P-Valor
Corretude	0,014
Completude	0,011
TEMPO (Em Minutos) Atualização parcial da Documentação	0,042
TEMPO (Em Minutos) Atualização total da Documentação	0,672

A Tabela 4.5 apresenta para cada atributo avaliado qual a probabilidade sobre a (HP0) hipótese nula (*p-value*).

A significância da diferença entre as medias para os Grupos ScrumDDM e Dirigido, foi avaliada com o teste t-Student para amostras independentes. Os pressupostos deste método estatístico, a saber, as normalidades das distribuições e a homogeneidade de variâncias nos dois grupos foram avaliados respectivamente, com o teste de Shapiro Wilk (SW(10) Grupo ScrumDDM >0,932; p= 0,471; SW(10) Grupo Dirigido >0.932; p= 0,471) e consideraram-se estatisticamente significativas as diferenças de médias cujo p-valor do teste foi inferior ou igual a (0,05) (MIOT, 2017).

Os participantes do Grupo ScrumDDM para o atributo de *Corretude*, obtiveram em média 86% (desvio padrão: 11%) na quantidade de estórias evoluídas de forma correta, enquanto que os participantes do Grupo Dirigido obtiveram em média 66% (desvio padrão: 21%). De acordo com o teste t-Student, as diferenças observadas (0,200), o que equivale que o Grupo ScrumDDM obteve 20% a mais de acertos em relação ao Grupo Dirigido, são estatisticamente significativas (t(18)= -2,72; p-valor= 0,014 d= 0,54). A dimensão do efeito do atributo *Corretude* para o Grupo ScrumDDM é elevada (d= 0,54) e de acordo com o IC a 95%] -0,35; -0,05[, os participantes do Grupo ScrumDDM obtiveram uma média de acertos entre (5%) e (35%) a mais em relação aos participantes do Grupo Dirigido (MIOT, 2017).

Os participantes do Grupo ScrumDDM para o atributo de *Completude*, obtiveram em média 72% (desvio padrão: 24%) na quantidade de estórias evoluídas de forma completa, enquanto que os participantes do Grupo Dirigido obtiveram em média 36% (desvio padrão: 32%). De acordo com o teste t-Student, as diferenças observadas (0,360) equivale que o Grupo ScrumDDM obteve 36% de completude em relação ao Grupo Dirigido

esta diferença é estatisticamente significativa (t(18)= -2,83; p-valor= 0,011 d= 0,55). A dimensão do efeito/confiança do atributo *Completude* para o Grupo ScrumDDM é elevada (d= 0,55) e de acordo com o IC a 95%] -0,62; -0,09[. Esta dimensão significa que o Grupo ScrumDDM desenvolveu em média entre (9%) e (62%) a mais de estórias de forma completa em relação aos participantes do grupo dirigido (MIOT, 2017).

Os participantes do Grupo ScrumDDM para o atributo *Tempo* quando atualizado/evoluído parte da documentação, obtiveram uma média de 38 minutos (desvio padrão: 17 minutos), enquanto que os participantes do Grupo Dirigido realizaram a evolução utilizando em média 55 minutos (desvio padrão: 17 minutos). De acordo com o teste t-Student, as diferenças observadas (17 minutos) entre os tempos médios dos dois grupos, mostra que esta diferença é estatisticamente significativa (t(18)= -2,18; p-valor= 0,042; d=0,45). A dimensão do efeito do atributo *Tempo* para o Grupo ScrumDDM é elevada (d= 0,45) e de acordo com o IC a 95%] -30,2; -0,6[, os participantes do Grupo ScrumDDM para parte da documentação, demoraram em média entre 0,6 e 30,2 minutos a menos que os participantes do grupo dirigido (MIOT, 2017).

Os participantes do Grupo ScrumDDM para o atributo *Tempo considerando atualização de toda a documentação*, demoraram em média 59 minutos (desvio padrão: 21 minutos), enquanto que os participantes do Grupo Dirigido para toda a documentação, demoraram em média 55 minutos (desvio padrão: 17 minutos). De acordo com o teste t-Student, as diferenças observadas (4 minutos) entre os tempos médios dos dois grupos não são estatisticamente significativas (t(18)= 0,43; p-valor= 0,672 d= 0,10). A dimensão do efeito do atributo *Tempo* para o Grupo ScrumDDM é pequena (d= 0,10) e de acordo com o IC a 95%] -14,3; 21,7[em minutos, a variabilidade das diferenças de tempo entre os participantes não foi suficiente para evidenciar alguma diferença entre os grupos (MIOT, 2017).

4.1.5.2 Validade da Análise: Esta seção apresenta as ameaças à validação interna, externa, de construção e de conclusão do experimento.

Validade interna: Ameaças à validação interna estão relacionadas a fatores não controlados que podem influenciar os efeitos dos tratamentos nas variáveis.

Com o objetivo de minimizar possíveis ameaças quanto ao nível de conhecimento dos participantes sobre UML, na linguagem JAVA (Eclipse), foi requerido de todos os participantes conhecimento sobre UML ou sobre JAVA. Para garantir este conhecimento prévio, um questionário foi aplicado com questões que incluíam trechos e código e conceitos nas linguagens especificas. Com bases nas respostas dos questionários, os participantes foram selecionados em cada Grupo (ScrumDDM ou Dirigido) e foi realizado o agendamento da sua participação para execução do experimento. Para evitar desvios no desenvolvimento, somente após o termino da evolução das estórias dos usuários, foi que os participantes tiveram conhecimento que seria avaliada a qualidade do código produzido por eles e o tempo utilizado nesse desenvolvimento.

Os participantes do Grupo ScrumDDM assim como os participantes do Grupo Dirigido foram acompanhados pelo pesquisador e nenhum dos participantes puderam contribuir na execução do outro grupo.

Validade externa: Ameaças a validade externa estão relacionadas a possibilidade de generalização dos resultados para outros ambientes.

Com o objetivo de minimizar o risco de representatividade da amostra foram selecionados 20 participantes o que possibilitou a formação de 2 grupos com 10 participantes, balanceados e escolhidos ao acaso.

A falta de possibilidade de execução do experimento na industria foi minimizada com a seleção de participantes com conhecimento na indústria na formação de ambos os grupos. Contudo, experimentos com casos reais são necessários para validar os resultados do estudo.

Validade construção: Ameaças à validade de construção estão relacionadas ao projeto do experimento.

Com o objetivo de diminuir as ameaças à construção, os documentos (Informações Gerais do Experimento, Cenário Proposto, Roteiro de Execução, Estórias dos Usuários_v01, Roteiro de Configuração e Instalação) foram construídos e disponibilizados. Assim como as transformações M2M e M2T. Os modelos e os códigos da versão também foram disponibilizados aos participantes. Esses documentos disponibilizados foram avaliados na aplicação do estudo piloto como suficientes para execução do experimento.

Em relação a qualidade dos atributos definidos, os participantes executaram a evolução das estórias dos usuários sem ter conhecimento sobre como seriam avaliados. Possíveis distorções no entendimento sobre as atividades a serem realizadas foram diminuídas com a disponibilização do documento Informações Gerais do Experimento. O atributo de tempo somente foi considerado no inicio da execução e não quanto os participantes realizavam a leitura das atividades.

Validade conclusão: A validade de conclusão está relacionada às questões que influenciam nas conclusões obtidas com o experimento.

O método estatístico utilizado na análise dos dados pode influir na validade de conclusão. Para avaliar as hipóteses, utilizamos um método com um fator e dois tratamentos. Além disso, a analise e os gráficos construídos foram discutidos com um especialista em analise descritiva.

Outra questão que pode ter influenciado foi a quantidade de participantes que realizaram o experimento controlado. Com o objetivo de validar a conclusão é relevante repetir o experimento com mais participantes.

Dificuldades Encontradas: Algumas dificuldades foram encontradas ao decorrer da execução do experimento como:

- Dificuldade em manipular a ferramenta de modelagem, mesmo com a disponibilização do roteiro de execução aos participantes. Contudo, após um tempo de utilização da ferramenta foi observado que a dificuldade foi superada.

- Alguns participantes do Grupo ScrumDDM apresentaram dificuldade com os conceitos de provedor, consumidor de serviço e entidade de serviço em SOA para ela-

boração do Modelo de Arquitetura, apesar dos conceitos estarem definidos quando solicitados na modelagem.

4.2 CONSIDERAÇÕES SOBRE O CAPÍTULO

Apresentamos neste capitulo um experimento controlado realizado com o objetivo de avaliar o código gerado com o metaprocesso ScrumDDM, no contexto da evolução das estórias dos usuários. Desta forma, o código gerado com a utilização do metaprocesso ScrumDDM foi analisado com o propósito de validar se o metaprocesso influenciou nos atributos de corretude, completude e o tempo de desenvolvimento. Estes atributos possibilitaram a verificação da quantidade de acertos, o quanto completo as estórias dos usuários foram desenvolvidas e a quantidade de tempo que foi utilizado na evolução dessas estórias do usuário.

Para realizar o experimento, foi feita uma seleção com os possíveis participantes, para verificar se os mesmos possuíam conhecimento desejado. Sendo assim, os participantes foram selecionados e distribuídos de forma aleatória em dois grupo com 10 participantes cada. Um grupo chamado de Grupo ScrumDDM, composto dos participantes que trabalharam na evolução das estórias dos usuários com o metaprocesso ScrumDDM e um segundo grupo chamado de Grupo Dirigido, composto pelos participantes que trabalharam na evolução das estórias dos usuários diretamente com o Scrum. Um conjunto de artefatos (modelos, manuais e roteiros) foram disponibilizados para ambos os grupos.

Com o resultado do processo de experimentação, ficou evidenciado que o metaprocesso ScrumDDM influenciou no desenvolvimento das evolução das estórias dos usuários em um software existente já desenvolvido com a utilização do metaprocesso ScrumDDM. Em relação ao Grupo Dirigido, o Grupo ScrumDDM obteve as melhores medias nos atributos Corretude 86% e completude 71%. Para o atributo Tempo, duas médias foram obtidas: a média 38 minutos foi menor/melhor quando somente a documentação para a geração do código fonte foi atualizada e a outra média 59 minutos foi maior quando foi atualizada toda documentação do projeto, além da geração de código, porém, este aumento de tempo não foi significativo. Desta maneira, podemos dizer que Grupo ScrumDDM realizou as evoluções mais corretamente, conseguiu gerar um código mais completo e realizou a atividade de evoluir as estórias dos usuários em menos tempo. Porém, quando considerando a atualização de toda a documentação do projeto, o Grupo ScrumDDM realizou o desenvolvimento em mais tempo. Em resumo, o experimento de avaliação do metaprocesso ScrumDDM para integração da abordagem de desenvolvimento dirigido por modelo e o framework ágil SCRUM se mostrou satisfatório para os atributos avaliados, além de possibilitar a geração de documentação do projeto arquitetural proposto.

Mesmo com o resultado obtido no experimento controlado, aspectos como construção de transformação e metamodelagem do arcabouço (modelos e transformações M2M e M2T) na instanciação do processo não foram avaliados, isso porque realizar este desenvolvimento pode ter o custo/tempo adicional no desenvolvimento do software.

Capítulo 5

TRABALHOS RELACIONADOS

Neste capítulo será realizada uma pesquisa bibliográfica com o objetivo de identificar e apresentar a conclusão sobre a investigação do uso de práticas de Desenvolvimento Dirigido a Modelo em Métodos ágeis. Foram incluídos os trabalhos que propõem estratégias de integração entre a abordagem DDM e Scrum, relacionados à avaliação proposta nesta dissertação.

5.1 PROPOSTAS PARA INTEGRAR PRÁTICAS DE DESENVOLVIMENTO DIRIGIDO POR MODELO A MÉTODOS ÁGEIS

Para mapear o estado da arte em direção à integração do Desenvolvimento Dirigido por Modelo em Métodos ágeis foi realizada uma pesquisa bibliográfica baseada em técnicas sistemáticas (KITCHENHAM, 2004),organizadas em três etapas:

1. Planejamento. Nesta etapa foi definida a questão de pesquisa e o protocolo a ser utilizado para conduzir a pesquisa;

2. Condução da pesquisa. Nesta momento foi executada uma busca de acordo com as abordagens DDM e métodos ágeis;

3. Sumarização. Nesta última etapa foram apresentados os resultados da pesquisa.

A questão de pesquisa (QP) que direcionou este estudo, buscou identificar as estratégias propostas na literatura para investigar o uso de práticas de Desenvolvimento Dirigido a Modelo em Métodos ágeis, sendo formulada da seguinte maneira: "QP: Como está o cenário atual sobre hibridização (integração) entre desenvolvimento dirigido a modelos e métodos ágeis?"

Com base na questão de pesquisa, foi definido o protocolo do estudo, o qual compreendeu a definição das bases de dados utilizadas na pesquisa, a definição da *String* de busca para filtrar os trabalhos, os critérios de inclusão e exclusão dos artigos encontrados e o procedimento de extração dos artigos relevantes.

Os trabalhos foram analisados de acordo com características consideradas relevantes ao contexto da integração, quais sejam, práticas do DDM em métodos ágeis, praticas de DDM em outras abordagens e integração entre métodos ágeis. A Tabela 5.1 apresenta a lista de características que foram analisadas. A primeira e segunda colunas apresentam respectivamente o número e a descrição da característica avaliada. A terceira coluna indica como a característica foi avaliada em cada trabalho.

Tabela 5.1 Características analisadas nos trabalhos relacionados à hibridização/integração em desenvolvimento dirigido por modelo e métodos ágeis

NR	Caracteristica analisada	Como a caracteristica é validada
C1	É uma abordagem DDM	1. Sim 2. Não
C2	É uma abordagem ágil	1. Sim 2. Não
C3	Integra DDM com Metodos ágeis (vice-versa)	1. Sim 2. Não
C4	Avalia a integração	1. Sim 2. Não
C5	Contempla desenvolvimento de evolução de software	1. Contempla 2. Não Contempla 3. Não informada

Além da utilização da revisão sistemática realizada por (SALES, 2017), as bases de dados utilizadas para a realização da pesquisa pontuais foram *ACM Digital Library*, *IEEE Digital Library*, *Outros*, as quais foram submetidasa seguinte String de busca: "(("Agile Model Driven Development"OR "Model Driven Development"OR "Model Driven Architecture"OR "Model Driven Engineering") AND ("Agile practices"OR "Agile Methods"OR "Agile techniques"OR "Agile Methodology"OR Agile OR Scrum OR "Extreme Programing"))".

Foram incluídos na pesquisa os trabalhos publicados em inglês, com propostas na indústria ou na academia e que datavam de 2010 a 2019. Os trabalhos que não correspondiam a hibridização/integração em desenvolvimento dirigido por modelo e métodos ágeis foram excluídos da pesquisa, ou seja, trabalhos excluídos por título, resumo e *abstract* fora do tema proposto. Os critérios extração utilizados para selecionar os trabalhos foram título, resumo e introdução, nesta ordem. Após esta triagem, os artigos selecionados foram lidos.

Conforme pode ser observado na Tabela 5.2, 10 trabalhos foram considerados relevantes para o contexto de hibridização (integração) em desenvolvimento dirigido por modelo e métodos ágeis. Os trabalhos selecionados estão detalhados a seguir considerando a integração de DDM com outras abordagens, a integração entre métodos ágeis e métodos ágeis com outras abordagens e, por fim, a integração de DDM com ágil (vice-versa).

Os itens a seguir apresentam um resumo dos trabalhos selecionados da Tabela 5.2, organizados em ordem cronológica segundo a integração de DDM com outras abordagens,

Tabela 5.2 Resumo dos trabalhos selecionados

Local Pesquisa	Critério (Qualitade de Trabalhos)	Ano	Trabalhos compatíveis com o tema proposto
Revisão Sistemática (SALES,2017)	• O Uso de MDA e Métodos Ágeis no Desenvolvimento de Sistemas de Informação (Aguiar et al., 2012) • Agile MDA - A White Paper (MELLOR et al., 2005), (ROSA; GONÇALVES; PANTOJA, 2013) • Agile model driven development is good enough (AMBLER, 2003), (SANTOS et al., 2018)	2012 2013 2018	**3**
ACM Digital Library	• Qualitas: um processo de desenvolvimento de software orientado a modelos (AL- MEIDA et al., 2014) • Analysis of Service oriented Modeling Approaches for Viewpoint specific Model driven Development of Microservice Architecture (RADEMACHER; SACHWEH; ZUNDORF, 2018) • Scaling for agility: A reference model for hybrid traditional agile software development methodologies (GILL; HENDERSON-SELLERS; NIAZI, 2018) • Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys (WAGNER et al., 2019)	2014 2018 2018 2019	**4**
IEEE Xplore	• User Engagement in the Era of Hybrid Agile Methodology (SCHMITZ; MAHA- PATRA; NERUR, 2018)	2018	**1**
Science Direct	• A Framework for Evaluating Model-Driven Self-adaptive Software Systems(MAGABLEH, 2019)	2019	**1**
arXiv (Cornell University)	• Agile Software Development Methodologies:Survey of Surveys (AL-ZEWAIRI et al., 2017)	2017	**1**
Total		**2012 -2019**	**10**

a integração entre métodos ágeis e com outros métodos, e por fim, a integração de DDM com métodos ágeis.

Associado indiretamente aos trabalhos selecionados, o trabalho *Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys* (WAGNER et al., 2019), O Trabalho de (WAGNER et al., 2019) apresenta uma pesquisa com base empírica externamente válida em (ER) *Engenharia de Requisitos*, com o objetivo de conduzir os engenheiros de software no desenvolvimento de processos de ER efetivos e eficientes com foco no problema.

A pesquisa no contexto da teoria empírica da prática de ER foi aplicada e testada primeiramente na Alemanha e, após ajustes, foi replicada em 10 países. Ao final da realização da pesquisa, uma base teórica formada por proposições inferidas da experiência e da realização do estudo, bem como os resultados do estudo piloto na Alemanha, foram avaliados. Cada proposição foi avaliada em um intervalos de confiança *bootstrapped* que possibilitou potenciais explicações para as proposições inferidas.

Os resultados desta pesquisa revelam, por exemplo, que existem grandes diferenças entre organizações em diferentes países e regiões em relação a prática de ER; que entrevistas, reuniões facilitadas e prototipagem são as técnicas de elicitação mais usadas; que os requisitos são frequentemente documentados textualmente; que os documentos são comuns para traçar uma ligação entre requisitos e código ou design; que as próprias

especificações de requisitos raramente são alteradas; e que os esforços de melhoria de engenharia de requisitos (processo) são, em sua maioria, direcionados internamente.

(WAGNER et al., 2019) relacionou as necessidades de desenvolver engenharia de requisitos independente do método ou da abordagem adotada, considerando o problema como ponto mais relevante.

A integração do desenvolvimento dirigido por modelos com outras abordagens tem sido uma tendência. A seguir iremos apresentar alguns métodos e abordagens integrados ao desenvolvimento dirigido por modelos.

A integração de DDM com outras abordagens

A hibridização/integração entre abordagens, métodos e tecnologias para o desenvolvimento de software vem sendo uma tendência por considerar que a integração pode potencializar os benefícios e as vantagens existentes em cada uma delas. Um exemplo de hibridização entre métodos é a integração entre os métodos ágeis Scrum e XP. Enquanto um apresenta como finalidade seguir um cronograma de gerenciamento de projeto, o outro possui, em suas fases, atividades e tarefas voltadas para a construção do código fonte. Um objetivo desta integração esta em executar as fases *PreGame, Game* e *PostGame* a fim de promover o desenvolvimento do software, sem desvirtuar a agilidade e o objetivo de gerenciamento de projeto do SCRUM, buscando, por fim, entregar o produto de software produzido.

- *A Framework for Evaluating Model-Driven Self-adaptive Software Systems* (MAGABLEH, 2019)
  O Trabalho de (MAGABLEH, 2019) apresenta uma pesquisa para integrar práticas do Desenvolvimento Dirigido por Modelo (DDM), a exemplo da construção de modelos e construção/execução de transformações (M2M e M2T) à abordagem de Desenvolvimento de Software Baseado em Componentes (CBSD) e softwares baseado em contextos, visto que são uma alternativa interessante para construção de sistemas de software auto-adaptativos. Isso se dá porque o objetivo dessas tecnologias é reduzir os custos e esforços no desenvolvimento, pois melhora a modularidade, a flexibilidade, a adaptabilidade e a confiabilidade dos sistemas de software.

  A proposta da pesquisa é desenvolver uma estrutura de avaliação para analisar e avaliar as características de abordagens orientadas a modelos e sua capacidade de suportar software com alta-adaptabilidade e confiabilidade em ambiente dinâmicos. Essa estrutura de avaliação pode facilitar aos desenvolvedores de software a seleção de uma metodologia de desenvolvimento que atenda aos requisitos de software e reduza o esforço de desenvolvimento de sistemas de software auto-adaptativos. Este estudo destaca as principais desvantagens das abordagens baseadas em modelos sustentados nos trabalhos relacionados e enfatiza os aspectos voláteis do software auto-adaptativo nas fases de análise, projeto e implementação das metodologias de desenvolvimento. Além disso, argumentamos que as metodologias de desenvolvimento devem deixar a seleção de linguagens de modelagem e ferramentas de modelagem para os desenvolvedores de software.

- *An empirical comparative evaluation of gestUI to include* (PARRA et al., 2019)
  O Trabalho de (PARRA et al., 2019) se assemelha ao trabalho proposto nesta dissertação, por realizar um experimento controlado para comparar a utilização de práticas de desenvolvimento dirigido por modelo no desenvolvimento de evolução do software em relação a ao mesmo desenvolvimento realizado diretamente no código fonte. Para isso (PARRA et al., 2019) considera que existem diversas ferramentas que suportam a personalização dos gestos dos usuários. A manutenibilidade corretiva/evolutiva de novos gestos implica escrever novas linhas de código que dependam fortemente da plataforma de destino em que o sistema é executado.

  Para evitar ou diminuir a dependência de plataforma, (PARRA et al., 2019) propôs o desenvolvimento do método gestUI, baseado em modelo para: (i) definição de gestos baseados em toque e (ii) a inclusão da interação baseada em gestos em interfaces de usuário existentes na computação de plataformas desktop. A finalidade da proposta de (PARRA et al., 2019) é, através de um experimento controlado, comparar um método MDD para lidar com gestos versus um método centrado em código para incluir interação baseada em gestos em interfaces de usuário. Na comparação foi analisada a usabilidade por meio da eficácia, eficiência e satisfação. A satisfação foi medida usando os aspectos da facilidade de uso, a utilidade percebida e a intenção de uso dos sujeitos. Participaram do experimento controlado 21 indivíduos, mestres em ciência da computação e estudantes de doutorado. Cada sujeito aplicou os dois métodos na realização do experimento. Os indivíduos realizaram tarefas relacionadas à definição de gestos personalizados e à modificação do código-fonte da interface do usuário para incluir interação baseada em gestos.

  Ao final, os dados foram coletados por meio de questionários e analisados por meio de testes estatísticos não paramétricos. Os resultados mostraram que o gestUI é mais eficiente e eficaz. Além disso, os resultados concluem que o gestUI é percebida como mais fácil de usar do que o método centrado no código. De acordo com esses resultados, o gestUI foi considerado um método promissor na definição de gestos personalizados e na inclusão de interação baseada em gestos em interfaces de usuários existentes de sistemas de software de computação de desktop.

- *Analysis of Service oriented Modeling Approaches for Viewpoint specific Model driven Development of Microservice Architecture* (RADEMACHER; SACHWEH; ZÜNDORF, 2018)
  O Trabalho de (RADEMACHER; SACHWEH; ZÜNDORF, 2018) apresenta as características que assemelham e distinguem o *Microservice Architecture* (MSA) da Arquitetura Orientada a Serviços (SOA). Ambos os estilos arquitetônicos contem serviços construídos em blocos e, sendo assim, enfrentam desafios parecidos em relação, por exemplo, identificação, composição e provisionamento de serviços. Divergindo da MSA, a SOA possui conhecimento para enfrentar esses desafios. Devido a essas duas abordagens arquiteturais serem baseados em serviços, surge a necessidade de saber até que ponto a MSA pode se basear nas descobertas das pesquisas e da prática de SOA.

Neste trabalho, (RADEMACHER; SACHWEH; ZÜNDORF, 2018) essas necessidades foram abordadas com a utilização do Model driven Development (MDD) em projeto e operação de arquiteturas em serviços. Por isso, uma análise das Abordagens MDD para SOA foi apresentada, visando contribuir com a identificação e agrupamento semântico de conceitos de modelagem para projeto e operação de SOA.

Para cada conjunto de conceito apresentado, é realizada uma avaliação da aplicabilidade ao MDD de MSA (MSA-MDD) e adicionado um ponto de vista específico na modelagem. O objetivo da análise apresentada é fornecer uma base conceitual para um metamodelo MSA-MDD.

- *Qualitas: um processo de desenvolvimento de software orientado a modelos* (ALMEIDA et al., 2014)
  O Trabalho de (ALMEIDA et al., 2014) além de apresentar a definição do desenvolvimento dirigido por modelo e seu objetivo, contribuiu de forma significativa na elaboração desta dissertação, pois possibilitou a instanciação do processo Qualitas a partir do metaprocesso ScrumDDM. Considerando que a área de Engenharia de Software possui como objetivo principal o desenvolvimento de produtos de software com qualidade, é necessário que este desenvolvimento envolva cada vez mais abordagens de teste de software, com o intuito de realizar os testes desde as fases iniciais software. Dessa forma, objetiva-se identificar e corrigir os erros o quanto antes no desenvolvimento, contribuindo assim com a melhora da qualidade do software. Uma abordagem que faz uso de modelos no teste de software é o *Desenvolvimento Dirigido por Teste* (DDT), a qual faz uso de práticas do MDD, através da geração automática de artefatos de teste de acordo com as regras de transformação predefinidas a partir de modelos de desenvolvimento.

  O objetivo deste trabalho é apresentar o Qualitas, um processo para o desenvolvimento de software orientado a modelos, que possibilite tanto o uso de modelos quanto a efetiva hibridização/integração entre DDM e MDT. A finalidade do processo esta em promover um maior controle das etapas e atividades do processo de desenvolvimento de software, como também agregar qualidade aos produtos de software desenvolvidos. Para avaliar a integração das práticas DDM ao DDT no processo de desenvolvimento, um estudo experimental do processo Qualitas foi realizado através da implementação de funcionalidades relacionadas ao Sistema de Triagem Neonatal do Hospital Universitário (HU) da Universidade Federal de Sergipe (UFS), destacando as vantagens e mostrando as limitações do modelo.

A integração entre métodos ágeis com outros métodos

Assim como acontece com a abordagem de desenvolvimento dirigido por modelo, a adoção de métodos ágeis pode requerer a integração entre métodos ágeis e não ágeis para arquitetar uma metodologia adaptativa híbrida:

- *Agile Software Development Methodologies:Survey of Surveys* (AL-ZEWAIRI et al., 2017)
  O Trabalho de (AL-ZEWAIRI et al., 2017) apresenta um levantamento bibliográfico das pesquisas sobre diferentes métodos ágeis. Nos últimos 15 anos, um número excessivo de pesquisas tem sido desenvolvidas. Um grande número de métodos notáveis tem sido propostos e vários levantamentos foram apresentados por muitos pesquisadores, todos em metodologia ágil de desenvolvimento de software.

  No levantamento bibliográfico das pesquisas das diferentes metodologias ágeis, entre os anos 2000 e 2015,foi utilizado uma metodologia de pesquisa intuitiva definida de "*Compare and Review*" (CR). Além disso, esses artigos selecionados na pesquisa foram classificados em quatro categorias principais de acordo com sua área de estudo. Uma consideração importante na pesquisa realizada foi o fato da categoria mais pesquisada no período citado ter sido "*Hybrid Agile*".

- *Scaling for agility: A reference model for hybrid traditional agile software development methodologies* (GILL; HENDERSON-SELLERS; NIAZI, 2018)
  O Trabalho de (GILL; HENDERSON-SELLERS; NIAZI, 2018) apresenta uma lacuna onde pode ser necessário integrar elementos do desenvolvimento ágeis e não ágeis com o objetivo de projetar uma metodologia adaptativa híbrida. Contudo, como identificar os elementos e componentes (ágeis ou não ágeis), relevantes na construção da arquitetura de referência para metodologia adaptativa híbrida sensível ao contexto?"

  O trabalho busca responder quais os elementos e componentes são relevantes para a construção de um modelo de arquitetura de referência de metodologia adaptativa híbrida usando um modelo empírico qualitativo construtivo. Para responder a questão levantada, uma pesquisa foi realizada. Assim, foi possível descobrir os elementos ou componentes de agilidade, abstração, valor comercial, política comercial, regras, jurídico, contexto e facilidade que não foram explicitamente modelados ou discutidos em Padrões Internacionais (IS), como o metamodelo ISO/IEC 24744.

- *User Engagement in the Era of Hybrid Agile Methodology* (SCHMITZ; MAHAPATRA; NERUR, 2018)
  O Trabalho de (SCHMITZ; MAHAPATRA; NERUR, 2018) apresenta um estudo com dois projetos híbridos, adaptando e combinando práticas ágeis em uma estrutura de projeto tradicional (e às vezes vice-versa). O projeto híbrido adaptado e utilizado pelas equipes de projeto enfatiza a flexibilidade ao adotar o contexto de projeto local, mas corre o risco de negligenciar importantes dinâmicas de comunicação necessárias à troca efetiva de informações.

  Foi proposto neste trabalho um estudo com dois projetos híbridos com a finalidade de evidenciar as diferenças entre a participação frequente e engajamento efetivo dos participantes. Os resultados desses projetos sugerem que a adaptação deve remover a orquestração do desenvolvedor das atividades de feedback do usuário, a fim de expor requisitos difíceis de encontrar e substituir cenários pseudo-reais por situações reais para treinamento efetivo do usuário.

Em resumo sobre a hibridização com métodos ágeis, o primeiro trabalho evidenciou, através de uma pesquisa bibliográfica, que o termo mais procurado foi o *Hybrid Agile* ou Ágil Hibrido, confirmando assim uma tendência de hibridização crescente no mercado e na academia. O segundo trabalho apresentou o desafio de construir um modelo que permitisse a integração entre elementos da metodologia ágil e a metodologia tradicional. Nesta construção foi utilizada uma abordagem de pesquisa empírica para preencher uma parte da lacuna (entre teoria e prática) em métodos ágeis. As restrições que foram abordadas no estudo que combinou práticas ágeis em uma estrutura de projeto tradicional contribuíram para restaurar o equilíbrio do processo de troca de informações, melhorando o acoplamento de comunicação entre as equipes do projeto e os usuários. Todos os trabalhos apresentados mostram uma tendência em integrar os métodos ágeis entre si ou com a metodologia tradicional, a fim de extrair suas vantagens e benefícios.

A integração de DDM com métodos ágeis

Especialmente métodos ágeis e o desenvolvimento dirigido por modelo são alvos de constante hibridização. Este crescimento pode esta pautado pela: (i) possibilidade e necessidade de reduzir os custos e esforços no desenvolvimento, (ii) por melhorar a modularidade, (iii) a flexibilidade, (vi) a adaptabilidade e (v) a confiabilidade. A hibridização entre desenvolvimento dirigido por Modelo (DDM) e a metodologia ágil pode ter três vertentes: (1) *AMDD* (Agile Model Drinven Development) (AMBLER, 2006),(SANTOS et al., 2018), (2) à *ágil MDA* (*Agile Model Driven Architecture*) (AMBLER, 2003),(SANTOS et al., 2018) e, por fim, (3) a *MDD ágil* (BRAGA; LEAL, 2013), (AGUIAR, 2012),(ELSHEIKH; OMRAN, 2011) e (SALES, 2017).

- *Agile model driven development is good enough* (AMBLER, 2003)
  O Trabalho de (AMBLER, 2003), apresenta a primeira vertente que propõe integrar *Agile modeling* ao DDM. Desta forma, é definido o conceito de *AMDD (Agile Model Drinven Development)*, ou seja, a versão ágil do desenvolvimento dirigido por modelos.

  O objetivo desta proposta visa atender à necessidade de adequar os processos de software às constantes mudanças, para obter resposta rápida às mudanças do ambiente, maior produtividade, menor risco e custo do desenvolvimento, tudo isso dentro do prazo estimado e com a qualidade desejada. É utilizado nesta proposta de integração o *(AM) Agile modeling*, buscando-se obter as vantagens da modelagem DDM com as práticas do desenvolvimento ágil. Nesta proposta não é necessário a utilização de ferramentas sofisticadas, devendo a integração ser o mais simples possível, sem a necessidade de utilizar o padrão MDA da OMG.

- *Agile MDA - A White Paper* (MELLOR et al., 2005)
  O Trabalho de (MELLOR et al., 2005), apresenta a segunda vertente que propõe integrar os métodos ágeis a (MDA) *Agile Model Driven Architecture*. Esta integração é baseada na construção de modelos de código executados, testados e modificados em ciclos curtos, iterativos e incrementais. Estes modelos devem ser o mais

completo possível para que eles possam ser executados sozinhos. Nesta proposta é necessário o uso de tecnologias que suportem os padrões da OMG, na qual são adicionadas práticas ágeis no desenvolvimento dirigido a modelos.Desta forma, o processo é semelhante a um processo tradicional com inserção de práticas ágeis.

- *O Uso de MDA e Métodos Ágeis no Desenvolvimento de Sistemas de Informação* (SALES, 2017)
  O Trabalho de (SALES, 2017) apresenta a terceira vertente que propõe integrar práticas do desenvolvimento dirigido por modelos (DDM) a metodologia ágil, através do método ágil Scrum. A base é o método ágil que ganha práticas DDM durante o ciclo de vida. Sendo assim, esta proposta apresenta o conceito do DDM Ágil. Práticas do DDM como metamodelagem, modelagem, transformações M2M e transformações M2T são incluídas nas fases de *PreGame, Game* e *PostGame* do Scrum, além da integração de outros métodos ágeis no Scrum.

  Esta ultima vertente é a base do desenvolvimento desta dissertação, pois em (SALES, 2017) é proposto um metaprocesso de integração das práticas de DDM ao método ágil Scrum definido como ScrumDDM. A proposta de (SALES, 2017) é avaliada através da execução de um experimento controlado para avaliar viabilidade da utilização do metaprocesso na construção de um software, ou seja, se foi possível seguir o processo, gerar os artefatos intermediários e o código fonte do software. No entanto, a eficácia da documentação gerada pelo ScrumDDM no apoio a evolução de software e a agilidade (esforço e velocidade) no desenvolvimento não foram avaliados. Por isso, uma segunda avaliação foi planejada para o ScrumDDM contemplando um segundo experimento controlado que foi realizado nesta dissertação.

Em resumo, as três vertentes apresentadas propõem integrar DDM e ágeis ou vice-versa: (i) O ágil MDA faz uso de práticas ágeis integradas ao processo MDA; (ii) O AMDD integra o *Agile modeling* com técnicas de desenvolvimento DDM e a terceira vertente (iii) MDD ágil integra práticas DDM aos métodos ágeis. O que diferencia as vertentes (i) e (ii) é o fato da necessidade de utilizar ou não ferramentas no processo de desenvolvimento. A similaridade entre essas vertentes é o fato delas objetivarem tornar o processo DDM mais ágil, não sendo este nosso foco.

5.2 ANÁLISE DOS TRABALHOS

A Tabela 5.3 exibe uma tabela que sumariza o resultado da pesquisa com base nos critérios estabelecidos no protocolo definido na seção 5.1. A primeira coluna da tabela identifica os trabalhos avaliados e as demais colunas exibem as avaliações de cada trabalho. A ultima linha apresenta as avaliações da Integração de práticas do DDM ao processo ágil, avaliando aspectos de evolução de software propostos nesta dissertação de mestrado.

Como pode ser observado na Tabela 5.3, de modo geral, os trabalhos selecionados apresentam a hibridização de abordagens como uma tendência. Alguns trabalhos propuseram esta hibridização através da integração de práticas do DDM com outras abordagens (MAGABLEH, 2019), (PARRA et al., 2019), (RADEMACHER; SACHWEH;

Tabela 5.3 Comparação entre os trabalhos selecionados de acordo com os critérios estabelecidos

Trabalhos analisados	É uma abordagem DDM	É uma abordagem ágil	Integra DDM com Metodos ágeis (vice-versa)	Avalia a proposta	Contempla Evolução de Software
	C1	C2	C3	C4	C5
• (AGUIAR, 2012), (AGUIAR, 2012), (EL-SHEIKH; OMRAN, 2011) , (SALES, 2017)	Sim	Sim	Sim	Sim	Não Contempla
• (MELLOR et al., 2005), (ROSA; GONÇALVES; PANTOJA, 2013)	Sim	Não	Sim	Sim	Não Contempla
• (AMBLER, 2003), (SANTOS et al., 2018)	Sim	Não	Sim	Sim	Não Contempla
• (ALMEIDA et al., 2014).	Sim	Não	Não	Sim	Não Contempla
• (PARRA et al., 2019).	Sim	Não	Não	Sim	Contempla
• (RADEMACHER; SACHWEH; ZUNDORF, 2018).	Sim	Não	Não	Sim	Não Contempla
• (GILL; HENDERSON-SELLERS; NIAZI, 2018)	Não	Sim	Não	Sim	Não Contempla
• (WAGNER et al., 2019)	Sim	Sim	Não	Não	Contempla
• (SCHMITZ; MAHA- PATRA; NERUR, 2018)	Não	Sim	Não	Sim	Não Contempla
• (MAGABLEH, 2019)	Sim	Não	Não	Sim	Não Contempla
• (AL-ZEWAIRI et al., 2017)	Não	Sim	Não	Sim	Não Contempla
Integração de páticas do DDM ao processo ágil avaliando aspectos de evolução de Software	Sim	Sim	Sim	Sim	Contempla

ZÜNDORF, 2018),(ALMEIDA et al., 2014). A avaliação da hibridização nesses trabalhos foi através da realização de uma pesquisa teórica ou pelo desenvolvimento de um experimento controlado. Porém, somente o trabalho (PARRA et al., 2019) contemplou aspectos de evolução de software.

Outros trabalhos propuseram hibridização entre métodos ágeis com outras abordagens e métodos (AL-ZEWAIRI et al., 2017),(GILL; HENDERSON-SELLERS; NIAZI, 2018), (SCHMITZ; MAHAPATRA; NERUR, 2018). A avaliação da hibridização nesses trabalhos foi através da realização de uma pesquisa teórica. Um aspectos relevante foi que nenhum dos trabalhos contemplou aspectos de evolução de software em seu desenvolvimento.

Além dos trabalhos já citados, outros trabalhos propuseram hibridizar práticas do DDM com métodos ágeis (AMBLER, 2003), (SANTOS et al., 2018), (MELLOR et al., 2005), (ROSA; GONÇALVES; PANTOJA, 2013), (AGUIAR, 2012),(EL-SHEIKH; OMRAN, 2011) e (SALES, 2017). Todos estes trabalhos integraram práticas de DDM e métodos ágeis, porem somente os trabalhos de (AGUIAR, 2012),(EL-SHEIKH; OMRAN, 2011) e (SALES, 2017) utilizaram o método ágil como abordagem principal, adicionando práticas do DDM no processo de desenvolvimento. Um aspectos relevante foi que nenhum dos trabalhos contemplou aspectos de evolução de software em seu desenvolvimento.

A abordagem proposta nesta dissertação avança no estado da arte em relação aos trabalhos aqui apresentados pois além de hibridizar/integrar práticas (metamodelagem,

modelagem, transformações M2M e M2T) do DDM ao método ágil Scrum, todas as características buscadas nos trabalhos relacionados são atendidas nesta dissertação.

5.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou um levantamento sobre as propostas utilizadas para conduzir à avaliação de uma integração entre as abordagens DDM e o método ágil Scrum. Notou-se que na maioria dos trabalhos, a avaliação da hibridização/integração foi realizada através de pesquisas com o objetivo de disponibilizar base teórica sobre a integração entre metodologias e abordagens.

Destacou-se, também, que as práticas de DDM foram utilizadas no desenvolvimento do software, enquanto aspectos relacionados à evolução de software não foram contemplado pela maioria dos trabalhos.

Este trabalho avança do estado da arte, pois, através do experimento controlado proposto nesta dissertação, foi possível concluir que o software foi evoluído de forma mais correta e com o maior numero de estórias dos usuários desenvolvidas. Quanto a agilidade, o mesmo preservou a agilidade nos dois aspectos: (i) quando atualizada somente a documentação que gera código e (ii) quando toda a documentação do projeto foi atualizada juntamente com o código. Os resultados da avaliação realizada podem ser mais um indicativo que a hibridização entre metodologias, abordagens e tecnologia possibilita extrair os benefícios e as vantagens entre ambas as abordagens.

Por fim, quando da avaliação do metaprocesso ScrumDDM, através da instanciação de outros processos de desenvolvimento de software, foi possível concluir que as fases, atividades, subatividades e tarefas pertencentes ao ScrumDDM podem ser utilizadas em conjunto com outro processo de desenvolvimento de software. Para avaliar esta capacidade de instanciação, o processo proposto por (ALMEIDA et al., 2014) foi instanciado a partir do ScrumDDM avaliado.

Capítulo

6

Este capitulo apresenta as considerações finais sobre a avaliação do metaprocesso ágil ScrumDDM, no contexto da evolução das estórias dos usuários.

CONSIDERAÇÕES FINAIS

Em desenvolvimento de software que utiliza alguma prática ágil, o artefato principal e mais atualizado é o código fonte em detrimento a documentação. Contrapondo-se a esta realidade, a abordagem de desenvolvimento dirigido por modelos, coloca o modelo como artefato principal no processo de desenvolvimento de software.

Com o objetivo de integrar a abordagem de desenvolvimento dirigido por modelo ao framework ágil SCRUM, extraindo desta forma os benefícios das duas abordagens, (SALES, 2017) propôs um metaprocesso ágil chamado de ScrumDDM. Contudo, não ficou evidente em sua proposta se instancias do metaprocesso preservam a agilidade característica do Scrum ou se propicia a evolução do software nos requisitos de usuário representado pelas estórias dos usuários, assim como, não ficou evidente se o ScrumDDM possibilita a instanciação de novos processos através dele.

Para verificar os pontos não evidenciados por (SALES, 2017), neste trabalho foi desenvolvido um experimento controlado que avaliou o metaprocesso ScrumDDM quanto a sua capacidade de evoluir um software e quanto a agilidade desta evolução. Para isso, foi utilizado o processo instanciado ArqProjProcess, também instanciado em (SALES, 2017). Através de um projeto arquitetural de software já desenvolvido com o SrumDDM, definimos uma nova versão para o software contemplando novas estórias do usuário e /ou modificações nas estórias existentes. Ao final da execução do experimento, foi evidenciado que o mataprocesso propiciou a evolução do software, gerando código mais correto e com o maior numero de estórias desenvolvidas de forma completa em relação ao desenvolvimento realizado com o Scrum. Além disso, as evoluções ocorreram de forma ágil, confirmando que o metaprocesso preservou a agilidade da metodologia ágil.

Ainda na avaliação do metaprocesso, um novo processo foi instanciado, o ScrumDDM-Qualitas, a partir do ScrumDDM, permitindo responder que o ScrumDDM possibilita a instanciação de novos processos através da integração das Fases, Atividades e Tarefas do novo processo com as Fases, Atividades e Tarefas existentes do metaprocesso ScrumDDM.

Ao final da avaliação realizada neste trabalho, o metaprocesso se mostrou que é ágil, pois preservou a característica de agilidade do Scrum, que permite que o software desenvolvido possa ser evoluído e que novos processos possam ser instanciados a partir dele. Além desses resultados, a utilização do metaprocesso possibilitou a atualização dos artefatos de documentação e do código fonte do produto do software.

6.0.1 Limitações e Trabalhos Futuros

Com o objetivo de ampliar a possibilidade de utilização do ScrumDDM, seria interessante testar o mesmo para outros domínios ou propósitos. Além disso, mesmo tendo profissionais da área de tecnologia como participantes do experimento controlado, seria relevante que o ScrumDDM fosse avaliado em cenário industrial, a inclusão do engenheiro de metamodelagem no processo de desenvolvimento, com o objetivo de construir os elementos de DDM antes da realização do desenvolvimento do software, otimizando assim o trabalho dos envolvidos.

Sendo assim, identificamos para trabalhos futuro:

- *Instanciar o ScrumDDM para outros propósitos e domínios*, diferentes dos já abordados como SOA e DDM, de modo a possibilitar a avaliação de mais aspectos do processo instanciado;
- *Reavaliar o ScrumDDM* em um cenário real fora do meio acadêmico;
- *Avaliar outros aspectos de qualidade de software.* Além dos aspectos de código e geração de documentação, pode ser relevante buscar a qualidade do processo de desenvolvimento do software;
- *Instanciar o ScrumDDM para teste de software.* Pode ser relevante instanciar um processo de teste de software ou um processo com a abordagem de desenvolvimento dirigido por teste a partir do metaprocesso ScrumDDM;

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, G. Uso de mda em um framework para seleção de práticas ágeis. 2012.

AITKEN, A.; ILANGO, V. A comparative analysis of traditional software engineering and agile software development. In: IEEE. *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. [S.l.], 2013. p. 4751–4760.

AL-ZEWAIRI, M. et al. Agile software development methodologies: survey of surveys. *Journal of Computer and Communications*, Scientific Research Publishing, v. 5, n. 05, p. 74, 2017.

ALMEIDA, C. C. d. J. et al. Qualitas: uma modelo de processo de desenvolvimento de software orientado a modelos. Ciência da Computação, 2014.

ALVES, E. L.; MACHADO, P. D.; RAMALHO, F. Uma abordagem integrada para desenvolvimento e teste dirigido por modelos. In: *2nd Brazilian Workshop on Systematic and Automated Software Testing*. [S.l.: s.n.], 2008.

AMBLER, S. W. Agile model driven development is good enough. *IEEE Software*, IEEE, v. 20, n. 5, p. 71–73, 2003.

AMBLER, S. W. Agile model driven development (amdd). p. 13, 2006.

BECK, K. *Extreme programming explained: embrace change*. [S.l.]: addison-wesley professional, 2000.

BECK, K. et al. Manifesto para desenvolvimento agil de software. *AGILE manifesto*, 2001.

BEYDEDA, S. et al. *Model-driven software development*. [S.l.]: Springer, 2005.

BRAGA, A. d. A.; LEAL, R. d. S. Estudo sistemático em dependabilidade e métodos ágeis: uma análise de falhas e defeitos. 2013.

BRAGA, V. *Um Processo para Projeto Arquitetural de Software Dirigido a Modelos e Orientado a Serviços*. Tese (Doutorado) — Dissertação, Universidade Federal de Pernambuco, Brasil, 2011.

CALDIERA, V. R. B.-G.; ROMBACH, H. D. Goal question metric paradigm. *Encyclopedia of software engineering*, v. 1, p. 528–532, 1994.

EL-SHEIKH, A.; OMRAN, A. Suggested framework for agile mda and agile methodologies. 2011.

GILL, A. Q.; HENDERSON-SELLERS, B.; NIAZI, M. Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, Springer, v. 20, n. 2, p. 315–341, 2018.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

LUCIA, A. D.; QUSEF, A. Requirements engineering in agile software development. *Journal of emerging technologies in web intelligence*, Academy Publisher, PO Box 40 Oulu 90571 Finland, v. 2, n. 3, p. 212–220, 2010.

MACIEL, R. S. P. et al. Supporting model-driven development using a process-centered software engineering environment. *Automated Software Engineering*, Springer, v. 20, n. 3, p. 427–461, 2013.

MAGABLEH, B. A framework for evaluating model-driven self-adaptive software systems. *arXiv preprint arXiv:1901.04020*, 2019.

MAGALHãES, A. P. Sistematizando o desenvolvimento de transformações modelo a modelo em uma abordagem dirigida a modelos. Salvador, 2016.

MDA, O. *Object Management Group Model Driven Architecture*. 2014.

MELLOR, S. J. et al. Mda destilada: Princípios de arquitetura orientada por modelos. *Ciência Moderna Ltda*, 2005.

MIOT, H. A. Assessing normality of data in clinical and experimental trials. *Jornal Vascular Brasileiro*, SciELO Brasil, v. 16, n. 2, p. 88–91, 2017.

OMG, S.; NOTATION, O. Software & systems process engineering meta-model specification. *OMG Std., Rev*, v. 2, p. 18–71, 2008.

PARRA, O. et al. An empirical comparative evaluation of gestui to include gesture-based interaction in user interfaces. *Science of Computer Programming*, Elsevier, v. 172, p. 232–263, 2019.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016.

RADEMACHER, F.; SACHWEH, S.; ZÜNDORF, A. Analysis of service-oriented modeling approaches for viewpoint-specific model-driven development of microservice architecture. *arXiv preprint arXiv:1804.09946*, 2018.

ROSA, A.; GONÇALVES, I.; PANTOJA, C. E. A mda approach for database modeling. *Lecture Notes on Software Engineering*, IACSIT Press, v. 1, n. 1, p. 26, 2013.

RUTLE, A. et al. Automatic transformation co-evolution using traceability models and graph transformation. In: SPRINGER. *European Conference on Modelling Foundations and Applications*. [S.l.], 2018. p. 80–96.

SALES, P. M. Integrado práticas do desenvolvimento dirigido a modelos ao scrum. 2017.

SANTOS, N. et al. Modeling in agile software development: Decomposing use cases towards logical architecture design. In: SPRINGER. *International Conference on Product-Focused Software Process Improvement*. [S.l.], 2018. p. 396–408.

SCHMITZ, K.; MAHAPATRA, R.; NERUR, S. User engagement in the era of hybrid agile methodology. *IEEE Software*, IEEE, 2018.

SCHOEPPING, G. et al. Um estudo exploratório a partir de um framework para seleção de práticas ágeis. Florianópolis, 2012.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide. PDF document*. 2016.

SEYFI, A.; PATEL, A. Briefly introduced and comparatively analysed: Agile sd, component-based se, aspect-oriented sd and mashups. In: IEEE. *Information Technology (ITSim), 2010 International Symposium in*. [S.l.], 2010. v. 2, p. 977–982.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2011.

TOMÁS, M. R. Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação. IET, 2009.

UTSUNOMIYA, C. T. *Métricas 00 aplicadas a código objeto java*. Tese (Doutorado) — Universidade Estadual de Maringá, 2003.

VILAIN, P.; FAGUNDES, P. B.; MACHADO, T. L. A framework for selecting agile practices and defining agile software processes. In: *SEKE*. [S.l.: s.n.], 2007. p. 25–28.

WAGNER, S. et al. Status quo in requirements engineering: A theory and a global family of surveys. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 28, n. 2, p. 9, 2019.

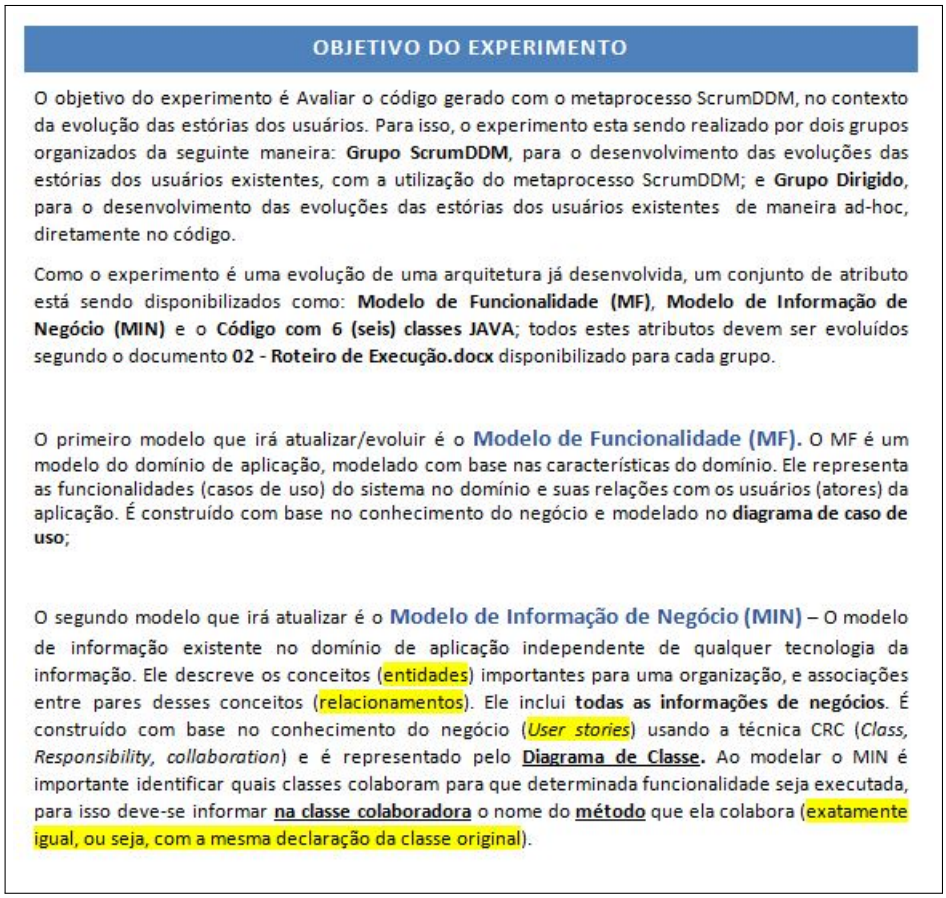
WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

YANG, H. et al. Linking functions and quality attributes for software evolution. In: IEEE. *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*. [S.l.], 2012. v. 1, p. 250–259.

Apêndice A

Este apêndice apresenta os documentos disponibilizados aos participantes do experimento controlado.

DOCUMENTOS DISPONIBILIZADOS



OBJETIVO DO EXPERIMENTO

O objetivo do experimento é Avaliar o código gerado com o metaprocesso ScrumDDM, no contexto da evolução das estórias dos usuários. Para isso, o experimento esta sendo realizado por dois grupos organizados da seguinte maneira: **Grupo ScrumDDM**, para o desenvolvimento das evoluções das estórias dos usuários existentes, com a utilização do metaprocesso ScrumDDM; e **Grupo Dirigido**, para o desenvolvimento das evoluções das estórias dos usuários existentes de maneira ad-hoc, diretamente no código.

Como o experimento é uma evolução de uma arquitetura já desenvolvida, um conjunto de atributo está sendo disponibilizados como: **Modelo de Funcionalidade (MF)**, **Modelo de Informação de Negócio (MIN)** e o **Código com 6 (seis) classes JAVA**; todos estes atributos devem ser evoluídos segundo o documento **02 - Roteiro de Execução.docx** disponibilizado para cada grupo.

O primeiro modelo que irá atualizar/evoluir é o **Modelo de Funcionalidade (MF).** O MF é um modelo do domínio de aplicação, modelado com base nas características do domínio. Ele representa as funcionalidades (casos de uso) do sistema no domínio e suas relações com os usuários (atores) da aplicação. É construído com base no conhecimento do negócio e modelado no **diagrama de caso de uso**;

O segundo modelo que irá atualizar é o **Modelo de Informação de Negócio (MIN)** – O modelo de informação existente no domínio de aplicação independente de qualquer tecnologia da informação. Ele descreve os conceitos (entidades) importantes para uma organização, e associações entre pares desses conceitos (relacionamentos). Ele inclui **todas as informações de negócios**. É construído com base no conhecimento do negócio (*User stories*) usando a técnica CRC (*Class, Responsibility, collaboration*) e é representado pelo **Diagrama de Classe.** Ao modelar o MIN é importante identificar quais classes colaboram para que determinada funcionalidade seja executada, para isso deve-se informar **na classe colaboradora** o nome do **método** que ela colabora (exatamente igual, ou seja, com a mesma declaração da classe original).

Figura A.1 00 - Informações Gerais do Experimento Página 01

ELEMENTOS DO EXPERIMENTO

Abaixo os documentos que devem ser seguidos pelos participantes:

01 - Cenário Proposto.docx (Domínio do Negocio);

02 - Roteiro de Execução.docx (O Roteiro de execução do experimento);

03 - Estórias dos Usuários_v01.xlsm (Documento com as estórias dos usuários que devem ser evoluídas);

04 - Roteiro de Configuração e Instalação.docx (Roteiro de configuração para instalação, configuração e importação);

EXECUÇÃO DO EXPERIMENTO

Passo 1: Abrir e configurar o ambiente conforme descrito no documento: **04 - Roteiro de Configuração e Instalação.docx** "Para esta etapa o TEMPO NÃO será marcado"

Passo 2: Lê o documento **01 - Cenário Proposto.docx** para ter conhecimento sobre o domínio do negocio; "Para esta etapa o TEMPO NÃO será marcado"

Passo 3: Com base no documento **03 - Estórias dos Usuários_v01.xlsm** realizar as modificações/evoluções das estórias dos usuários. Existem nesse documento:

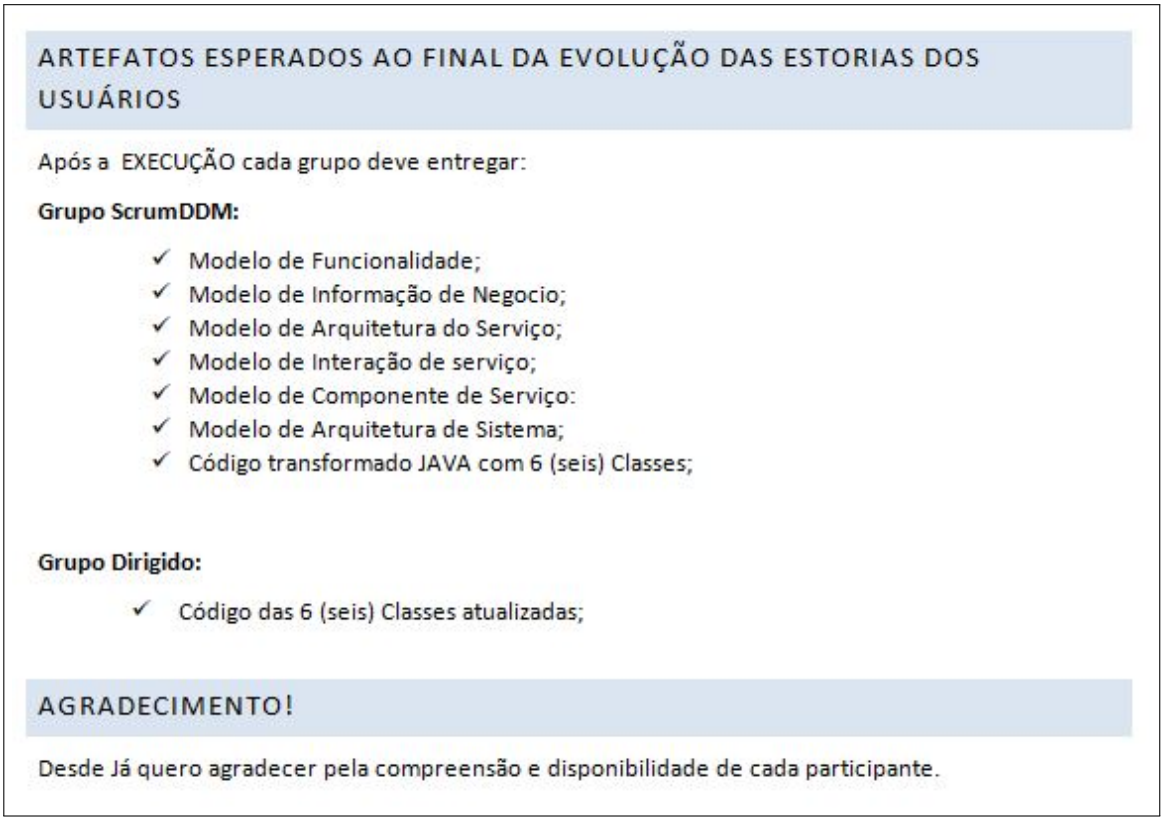
Duas linhas pintadas na cor VERDE, estas linhas/estórias dos usuários devem ser desenvolvidas por completo.

Cinco atributos na cor VERMELHA que devem ser adicionados nos métodos e nas classes das estórias já desenvolvidas.

"Para esta etapa o TEMPO NÃO será marcado"

Passo 4: Após Tomar conhecimento sobre os documentos citados nos passos anteriores o **02 - Roteiro de Execução.docx** deve ser seguindo. "O TEMPO DEVE SER MARCADO"

Figura A.2 00 - Informações Gerais do Experimento Página 02



ARTEFATOS ESPERADOS AO FINAL DA EVOLUÇÃO DAS ESTORIAS DOS USUÁRIOS

Após a EXECUÇÃO cada grupo deve entregar:

Grupo ScrumDDM:

- ✓ Modelo de Funcionalidade;
- ✓ Modelo de Informação de Negocio;
- ✓ Modelo de Arquitetura do Serviço;
- ✓ Modelo de Interação de serviço;
- ✓ Modelo de Componente de Serviço:
- ✓ Modelo de Arquitetura de Sistema;
- ✓ Código transformado JAVA com 6 (seis) Classes;

Grupo Dirigido:

- ✓ Código das 6 (seis) Classes atualizadas;

AGRADECIMENTO!

Desde Já quero agradecer pela compreensão e disponibilidade de cada participante.

Figura A.3 00 - Informações Gerais do Experimento Página 03

NOVO CENÁRIO

CONCILIAÇÃO DE CARTÃO

A venda por meio de cartões de crédito e débito vem se tornando cada vez mais comum. Essa facilidade pode se transformar em prejuízo financeiro se o controle dos recebimentos não for efetuado de maneira eficaz. Os problemas acontecem porque o processamento das vendas é suscetível a erros e o processo de conciliação é manual e ineficiente, o que proporciona um alto custo operacional.

O Dealercard é um produto da DealerAnalysys, que consiste no gerenciamento das conciliações de: **vendas**, **recebimentos** e **bancaria** todas registradas em sistema, pois avalia se os valores foram lançados corretamente e valida se a transação foi paga ou não pela operadora de cartão de crédito ou débito. Este controle ocorre porque são confrontados os extratos eletrônicos de cartão recebidos das operadoras de cartões com as movimentações geradas pelo SCR (Sistema de contas à receber), gerando resultados confiáveis que permite identificar divergências como: Cancelamentos, vendas duplicadas, pagamento de aluguel de maquina e taxas diferentes do contrato.

O produto é composto de importações dos extratos da administradora/operadora, processo de conciliação de vendas, conciliação dos recebimentos com a baixa automática dos títulos no Contas a Receber, conciliação bancaria além de consultas e relatórios que permitam a avaliação do processo como um todo.

PROCESSO 1: CONCILIAÇÃO DAS VENDAS

Com base nos arquivo de **EXTRATO DE VENDA** enviado pelas administradoras/operadora o sistema Dealercard possibilita conciliar/confrontar individualmente cada movimento de **VENDA DO DIA ANTERIOR REGISTRADO NO COFRE DA TESOURARIA** com as informações contidas no extrato de venda fornecido pela administradora/operadora.

As vendas conciliadas terão suas saídas de cofre da tesouraria, direcionadas/incluídas no contas a receber (SCR), acompanhados dos resumos de venda. Caso haja alguma diferença de valor do saldo existente no cofre e o valor das saídas, esta pode ser apurada nos relatórios de ajustes ou "*requests*".

PROCESSO 2: CONCILIAÇÃO DOS RECEBIMENTOS

O arquivo de **EXTRATO DE PAGAMENTO** enviado pela administradoras/operadora permite conciliar/confrontar com o **EXTRATO DE VENDA** também enviado pela administradora/operadoras, evidenciando o recebimento de cada transação, possibilitando a baixa dos títulos a receber no SCR e emitindo as críticas das vendas não recebidas.

São disponibilizadas consultas e relatórios com as divergências encontradas nas conciliações, os "requests" solicitados pelas administradoras, ajustes efetuados e os detalhes dos comprovantes de vendas.

PROCESSO 3: CONCILIAÇÃO BANCÁRIA

Com base nos arquivos de **EXTRATO DE PAGAMENTO** das administradoras/operadoras e o **EXTRATO DE RECEBIMENTO DOS BANCOS**, ambos enviados pela administradora/operadora o sistema realiza o confronto entre eles e disponibiliza as divergências encontradas nas conciliações.

RESUMO: DAS INTEGRAÇÕES DOS PROCESSOS

Todas as VENDAS REALIZADAS são enviadas/transmitidas do caixa para o cofre da tesouraria pelo gerente de vendas que também tem a função de verificar se todas as vendas foram enviadas para administradora/operadora. No Dia posterior da VENDA REALIZADA o gerente financeiro é o responsável por receber o arquivo de extrato de venda da administradora/operadora com todas as vendas realizadas e executar a CONCILIAÇÃO DAS VENDAS com as vendas armazenadas no cofre. As vendas conciliadas são excluídas do cofre da tesouraria e enviadas para o SCR (Contas a receber) gerando assim "títulos" em aberto. Com o objetivo de baixar (marcar como pago) os "títulos" no SCR o gerente financeiro vai receber o arquivo de extrato de pagamento e conciliar com o extrato de venda. Após a conciliação das vendas e dos recebimentos o gerente financeiro precisa executar a conciliação bancaria, verificando assim, se todos os pagamentos demonstrados no extrato de pagamento foram enviados/recebidos na conta do banco. Ambos os gerentes podem acompanhar os títulos vencidos em aberto através da geração dos relatórios gerenciais.

Figura A.4 01 - Cenário Proposto Páginas 01 e 02

ROTEIRO PARA EXECUÇÃO DO EXPERIMENTO

Nome do Participante: _____

ATIVIDADE 1:

Especificar modelo de negócio: Consiste na modelagem do conceito de negócio;

Marcar: Hora início atv1: _____

O primeiro modelo que irá atualizar/evoluir é o **Modelo de Funcionalidade (MF):**

Marcar Hora de INICIO Subatividade: _____

Passo 1 : Para atualizar o MF, selecione o diagrama de caso de uso "**MF.umlusc**" no caminho: TransfM2M\Modelos\Entrada:

TransfM2M
Modelos
Entrada
MF.uml
MF.umlusc
MIN.uml
MIN.umlclass
Saida
Perfis
Transformacao
TransfM2tUML_Java

Passo 2: No o diagrama de caso de uso "**MF.umlusc**" aberto, será necessário incluir as novas funcionalidades que estão pintadas de verde no arquivo: **03 - Estórias dos Usuários_v01.xlsm**.

Marcar Hora de FIM Subatividade: _____

O segundo modelo que irá atualizar é o **Modelo de Informação de Negócio (MIN):**

Marcar Hora de INICIO Subatividade: _____

Passo 1 : Para atualizar o MIN, selecione o diagrama de classes "**MIN.umlclass**" no caminho: TransfM2M\Modelos\Entrada

Figura A.5 02 - Roteiro Proposto Grupo ScrumDDM Páginas 01 e 02

Figura A.6 02 - Roteiro Proposto Grupo ScrumDDM Páginas 03 e 04

Passo 2: Com o arquivo **TCtoAServ.atl** aberto clique **RUN** e selecione a opção **RunArQServ** conforme imagem abaixo, (Caso o RunArqServ, não esteja configurado pode contar o arquivo "04 - Configuração da Transformação.docx")

✓ Aguarde até termino da execução, que será informado no console.

Passo 3: Concluída a transformação, o modelo gerado irá aparecer na pasta **Saída em TransfM2M /Modelos/Saida**. Vá até a aba ***Project Explorer*** e navegue até a referida pasta. Clique com o botão direito do mouse sobre o modelo gerado e selecione a opção ***Initialize Class Diagram*** conforme mostra ilustração:

Passo 4: Será aberta uma nova janela, basta clicar em ***Finish*** e o modelo visual será criado na mesma pasta, **Saida**.

Figura A.7 02 - Roteiro Proposto Grupo ScrumDDM Páginas 05 e 06

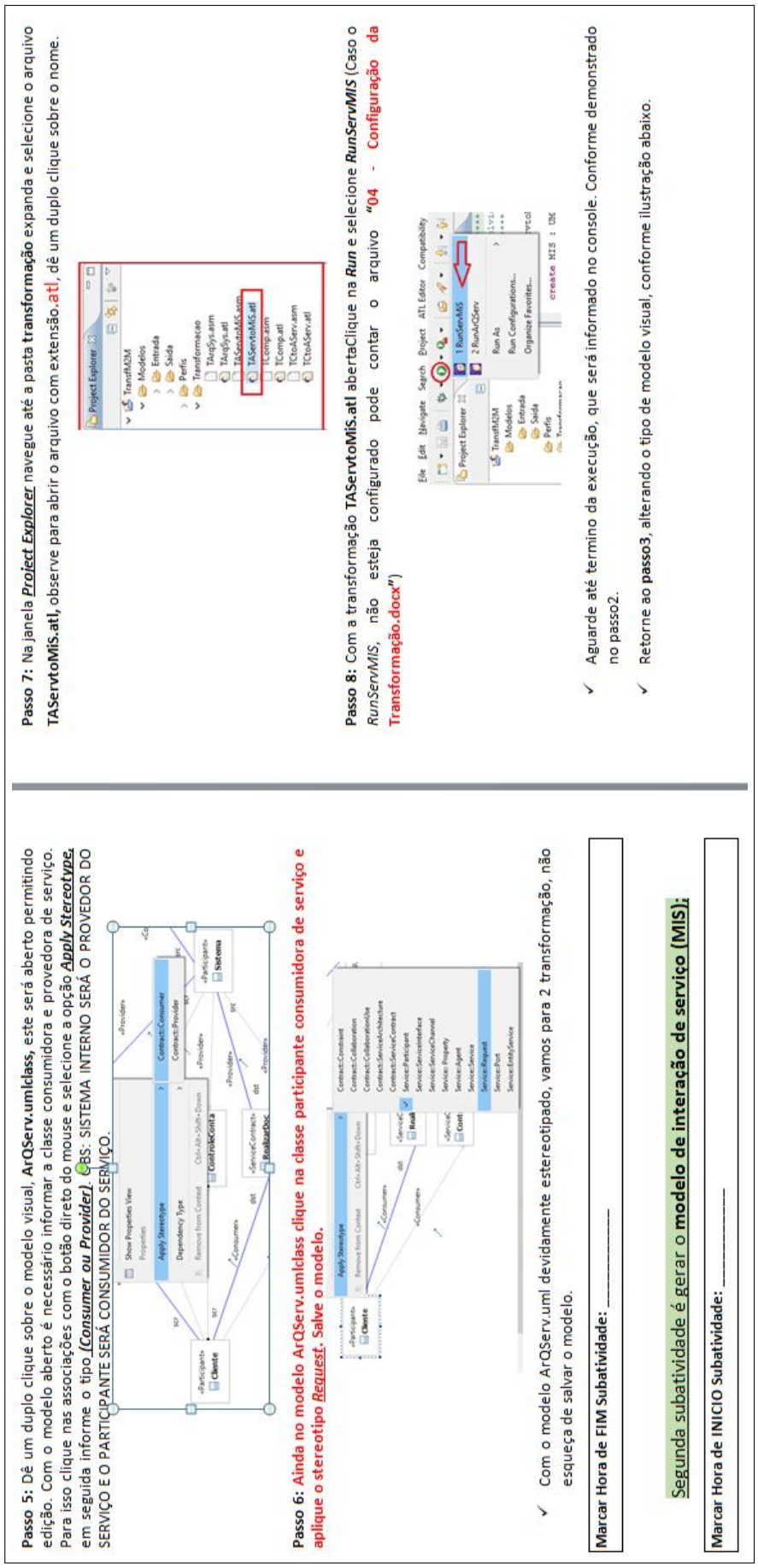


Figura A.8 02 - Roteiro Proposto Grupo ScrumDDM Páginas 07 e 08

Figura A.9 02 - Roteiro Proposto Grupo ScrumDDM Páginas 09 e 10

Marcar: Hora termino atv2: _____

ATIVIDADE 3:

Projetar Serviço: Consiste em gerar a arquitetura do sistema, sendo esta uma evolução do da arquitetura de componentes de serviço;

Marcar: Hora início atv3: _____

Repita o passo 1 da ATIVIDADE 2, tendo como referência a transformação **TArqSys.atl**, para executar utilize a **RunArqSys** , conforme ilustração abaixo. (Caso o *RunArqSys*, não esteja configurado pode contar o arquivo **"04 - Configuração da Transformação.docx"**)

- ✓ Aguarde até termino da execução, que será informado no console.
- ✓ Retorne ao passo 3 da ATIVIDADE 2, alterando o tipo de modelo visual, conforme ilustração abaixo.

- ✓ Execute o passo 4 da ATIVIDADE 2. Com o modelo gerado, você poderá ordenar os elementos e salve o modelo.

Marcar: Hora termino atv3: _____

ATIVIDADE 4:

Gerar Código: Consiste em gerar automaticamente o código através do modelo de informação de negocio desenvolvido;

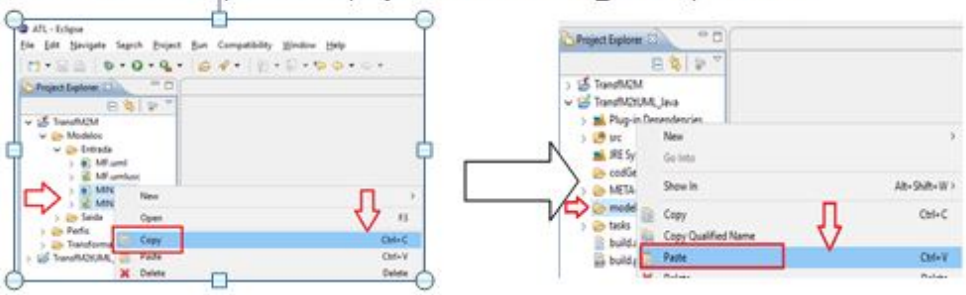
Marcar: Hora início atv4: _____

Concluída as transformações Modelo a Modelo, vamos para **segunda etapa**: transformação Modelo-Texto,ou seja, gerar o código a partir do modelo para uma plataforma específica.

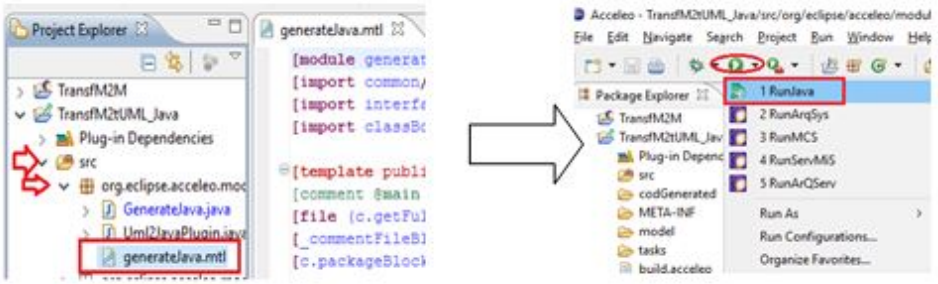
Passo 1: Para isso na janela ***Project Explorer*** procure o projeto **TransfM2tUML_Java** e abra (ou expanda) a pasta do projeto. Nela haverá 5 pastas: ***scr, codGenerated, META-INF, model, task***. As pastas ***codGenerated e model*** estão vazias.

Figura A.10 02 - Roteiro Proposto Grupo ScrumDDM Páginas 11 e 12

Passo 2: Retorne ao projeto **TransfM2M**-> Modelos -> Entrada, selecione e copie os arquivos (**MIN.uml e MIN.umlclass**) e cole no projeto **TransfM2tUML_Java** na pasta **Model.**



Passo 3: Ainda no projeto **TransfM2tUML_Java** navegue até a pasta *scr em: org.eclipse.acceleo.module.example.uml2java*, vá até o arquivo **GenerateJava.mtl** e dê um duplo clique e execute. Conforme ilustração abaixo:



- ✓ Aguarde até termino da execução, que será informado no **console**.
- ✓ Navegue até a pasta ***codGenerated*** e veja o código gerado.

Marcar: Hora termino atv4: _____

Figura A.11 02 - Roteiro Proposto Grupo ScrumDDM Página 13

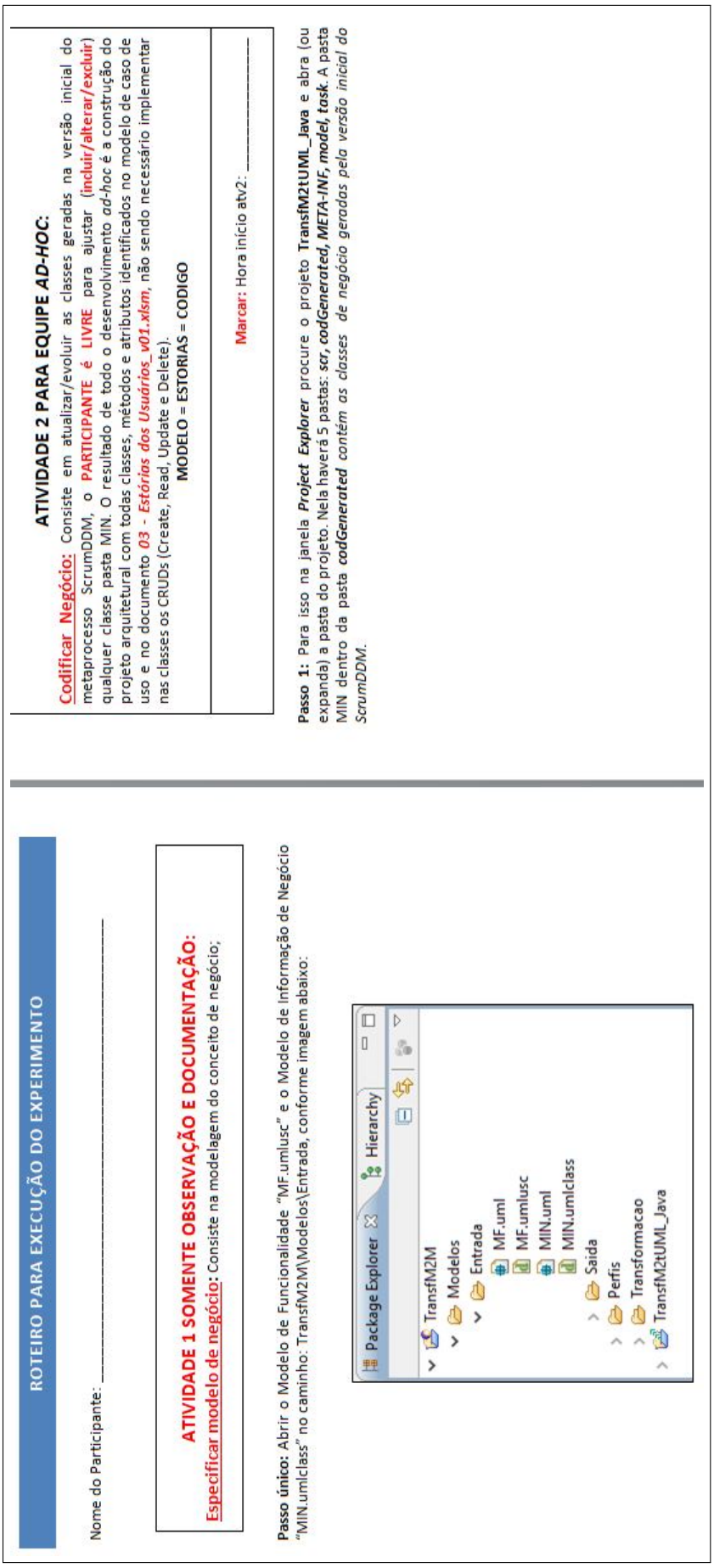


Figura A.12 02 - Roteiro Proposto Grupo Dirigido Páginas 01 e 02

Estórias de Usuários e Critérios de aceitação

ID	Estórias			Critérios de Aceitação		
	Papel	Eu gostaria de / Funcionalidade	Para que eu possa/Resultado	Contexto	Entrada de dados	Saída de dados
1	Como Sistema	Registrar as vendas de cartão de crédito	Registrar_Venda	Ter um SCR (Sistema de Contas à Receber) que gere títulos (Abertos) com base nas informações das vendas.	Operação de Venda	Numero de autenticação, Data da Venda, Valor, **Tipo de Venda**, Administradora	Mensagem de retorno: "Operação realizada com ou sem sucesso; Gerar um SCR por venda"
2	Como gerente de vendas	Verificar se a venda realizada no administradora/operadora de cartão	Verificar_Venda_ADM	Saber se a venda foi registrada pela operadora	Conciliação de Venda	Num autenticação, **Data da Venda**	Mensagem de Conformidade ou inconformidade no registro da operação
3	Como gerente de financeiro	Importar arquivo de **EXTRATO DE VENDA** da administradora/ operadora	Importar_ARQ_Extrato_Venda	Armazenar as informações de venda enviada pela administradora/operadora no sistema	Conciliação de Venda	Arquivo fornecido pela administradora do cartão layout (dados)	Mensagem de "Operação realizada com ou sem sucesso; arquivo importado!"
4	Como gerente financeiro	Executar a conciliação de venda	Executar_Conciliacao_Venda	Validar as vendas armazenadas no sistema com as vendas registradas na administradora	Conciliação de Venda	Numero de autenticação da venda, Valor Bruto	Um arquivo contendo as divergencias encontradas. Resumo com Divergências: **Cancelamento** **Duplicidade** **Aluguel de maquina** **Taxas**

Figura A.13 03 - Estórias dos Usuários Página 01

Estórias de Usuários e Critérios de aceitação

ID	Estórias			Critérios de Aceitação		
	Papel	Eu gostaria de / Funcionalidade	Para que eu possa/Resultado	Contexto	Entrada de dados	Saída de dados
5	Como gerente de financeiro	Importar arquivo de **EXTRATO DE PAGAMENTO** da administradora/ operadora Importar_ARQ_Extrato_Pagamento	Armazenar as informações de pagamento enviado pela administradora/ operadora no sistema	Conciliação de Pagamento	Arquivo fornecido pela administradora do cartão layout (dados)	Mensagem de "Operação realizada com ou sem sucesso; arquivo importado!"
6	Como gerente financeiro	Executar a conciliação dos recebimentos Executar_Conciliacao_Pagamento	Confrontar/conciliar informações registradas no **EXTRATO DE VENDAS** com o **EXTRATO DE PAGAMENTO**, ambos disponibilizado pela administradora/ operadora	Conciliação de Pagamento	Numero de autenticação de venda, numero de autenticação pagamento, Valor Liquido	Um arquivo contendo as divergencias encontradas. Resumo com Divergências: **Cancelamento** **Duplicidade** **Aluguel de maquina** **Taxas**
7	Como gerente financeiro	Verificar o cancelamento das vendas Verificar_Cancelamento	Controlar os cancelamentos e evitar perdas.	Conciliação de Pagamento	Numero de autenticação de cancelamento, **Numero de autenticação de venda**, Data do cancelamento.	Status de venda cancelada
8	Como gerente financeiro	Verificar se no valor recebido a taxa cobrada pela administradora esta correta Verificar_Taxas_ADM	Controlar os recebíveis	Conciliação de Pagamento	Valor Bruto, Taxa da ADM,	Valor a receber por cada transação

Figura A.14 03 - Estórias dos Usuários Página 02

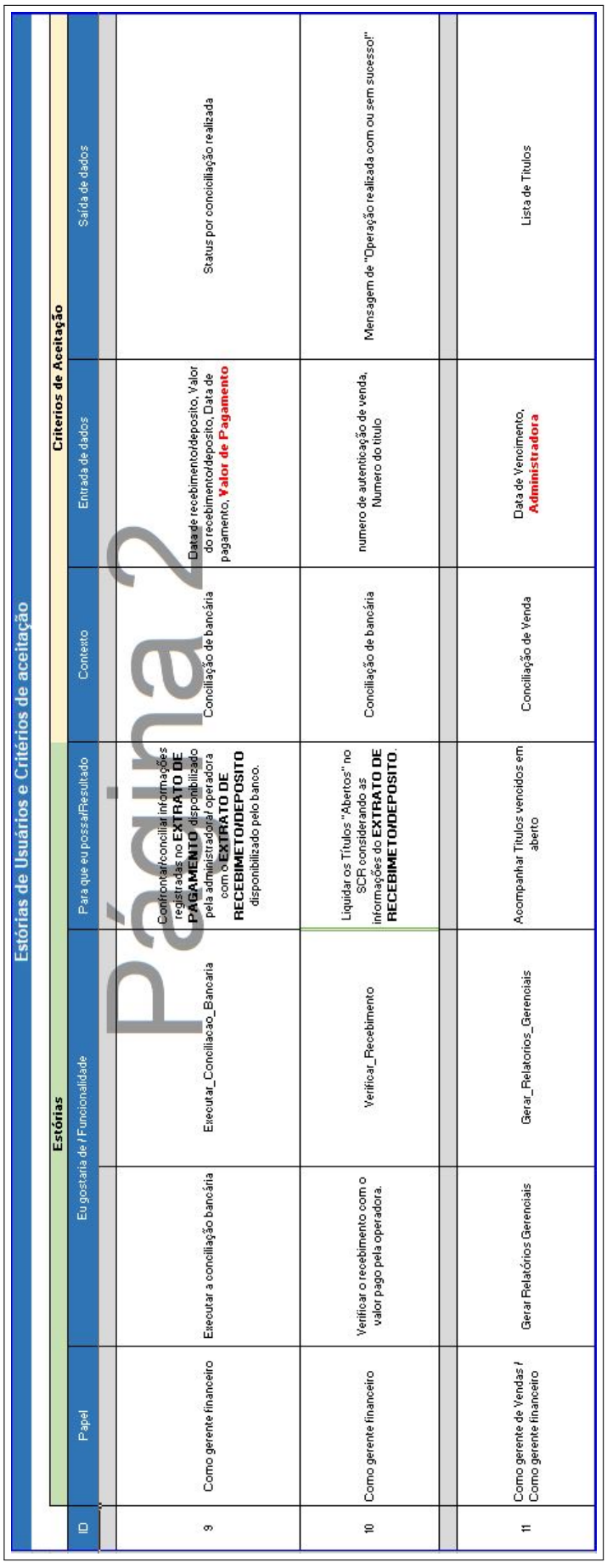
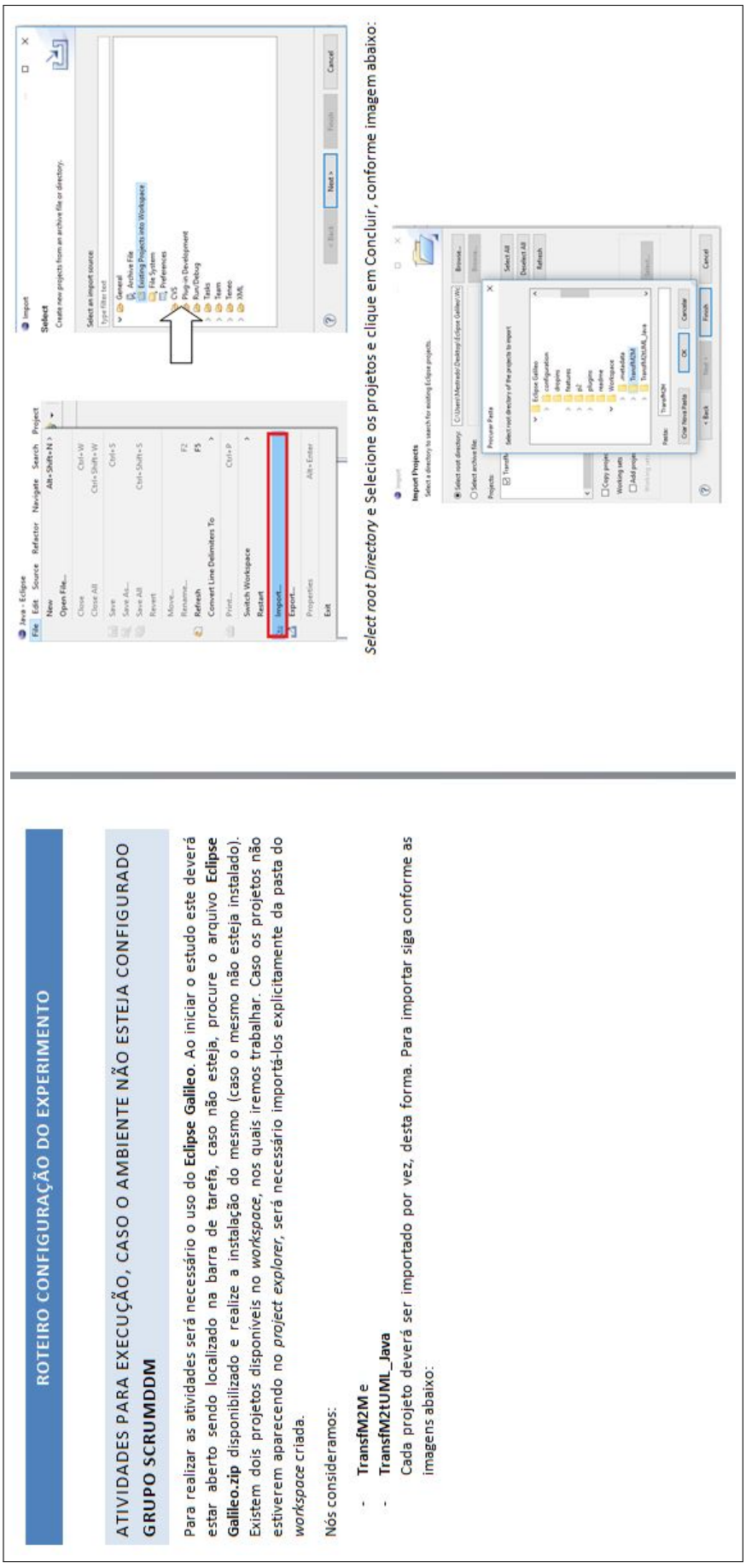


Figura A.15 03 - Estórias dos Usuários Página 03



ROTEIRO CONFIGURAÇÃO DO EXPERIMENTO

ATIVIDADES PARA EXECUÇÃO, CASO O AMBIENTE NÃO ESTEJA CONFIGURADO
GRUPO SCRUMDDM

Para realizar as atividades será necessário o uso do **Eclipse Galileo**. Ao iniciar o estudo este deverá estar aberto sendo localizado na barra de tarefa, caso não esteja, procure o arquivo **Eclipse Galileo.zip** disponibilizado e realize a instalação do mesmo (caso o mesmo não esteja instalado). Existem dois projetos disponíveis no *workspace*, nos quais iremos trabalhar. Caso os projetos não estiverem aparecendo no *project explorer*, será necessário importá-los explicitamente da pasta do *workspace* criada.

Nós consideramos:

- **TransfM2M** e
- **TransfM2tUML_Java**

  Cada projeto deverá ser importado por vez, desta forma. Para importar siga conforme as imagens abaixo:

Select root Directory e Selecione os projetos e clique em Concluir, conforme imagem abaixo:

Figura A.16 04-Roteiro de Configuração e Instalação Páginas 01 e 02

- ✓ Expanda o projeto TransfM2M;
- ✓ Verifique se o projeto contém 3 pastas (**Modelos, Perfis e Transformação**), a pasta **modelos** possui 2 subpastas:(**entrada com dois modelos MF e MIN e a pasta saída vazia**);
- ✓ Após leitura e compreensão do cenário disponibilizado no arquivo: 01 - Cenário Proposto.docx, iremos modela-lo, para isso navegue até a pasta modelos->entrada;

ATIVIDADES PARA EXECUÇÃO, CASO O AMBIENTE NÃO ESTEJA CONFIGURADO GRUPO DIRIGIDO

Para realizar as atividades será necessário o uso do **Eclipse Galileo**. Ao iniciar o estudo este deverá estar aberto sendo localizado na barra de tarefa, caso não esteja, procure o arquivo **Eclipse Galileo.zip** disponibilizado e realize a instalação do mesmo(caso o mesmo não esteja instalado). Existem dois projetos disponíveis no *workspace*, nos quais iremos trabalhar. Caso os projetos não estiverem aparecendo no *project explorer*, será necessário importá-los explicitamente da pasta do *workspace* criada.

Nós consideramos:

- **TransfM2M e**
- **TransfM2tUML_Java**

Cada projeto deverá ser importado por vez, desta forma. Para importar siga conforme as imagens abaixo:

Select root Directory e Selecione os projetos e clique em Concluir, conforme imagem abaixo:

Figura A.17 04-Roteiro de Configuração e Instalação Páginas 03 e 04

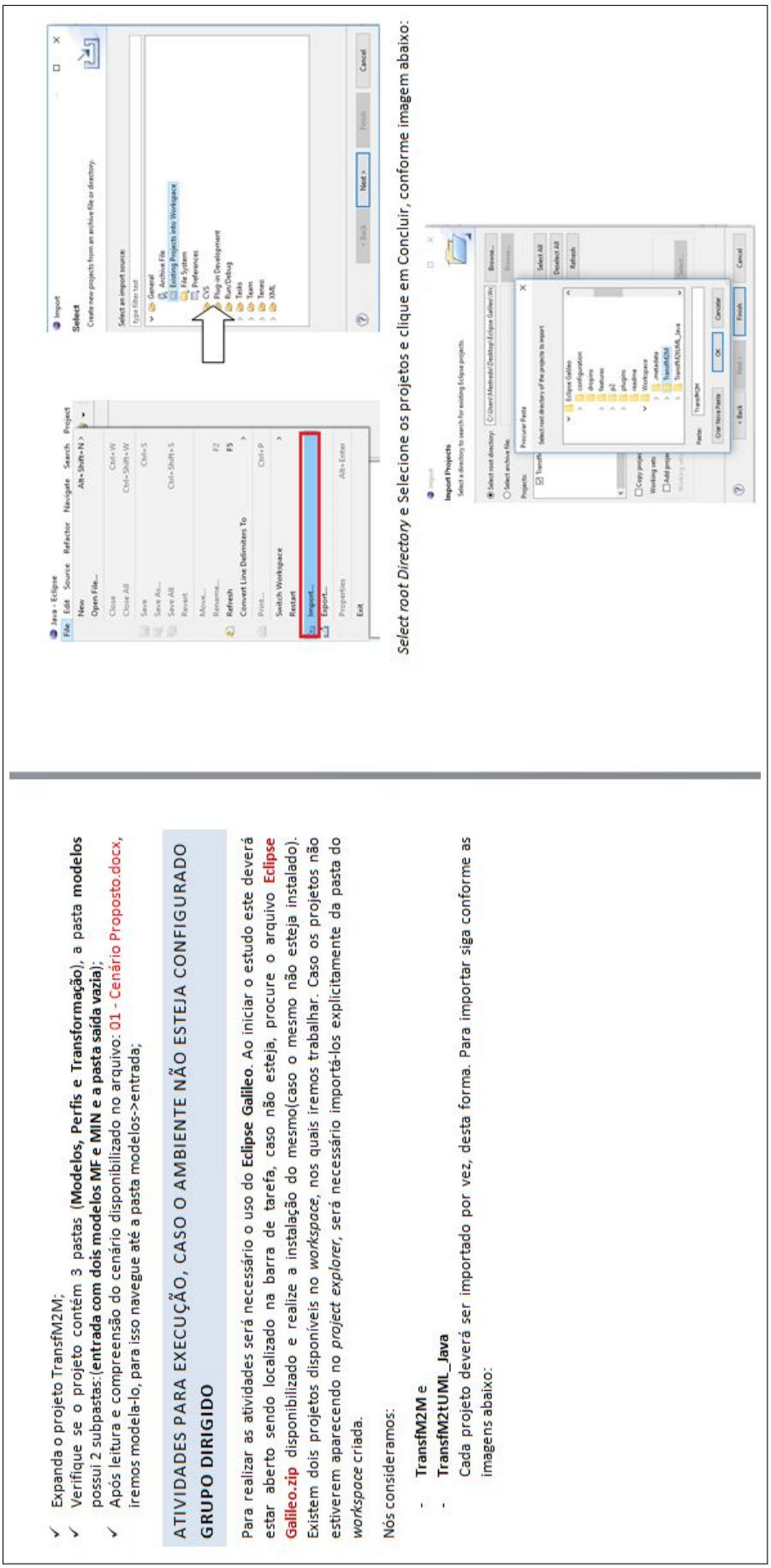


Figura A.18 04-Roteiro de Configuração e Instalação Páginas 05 e 06

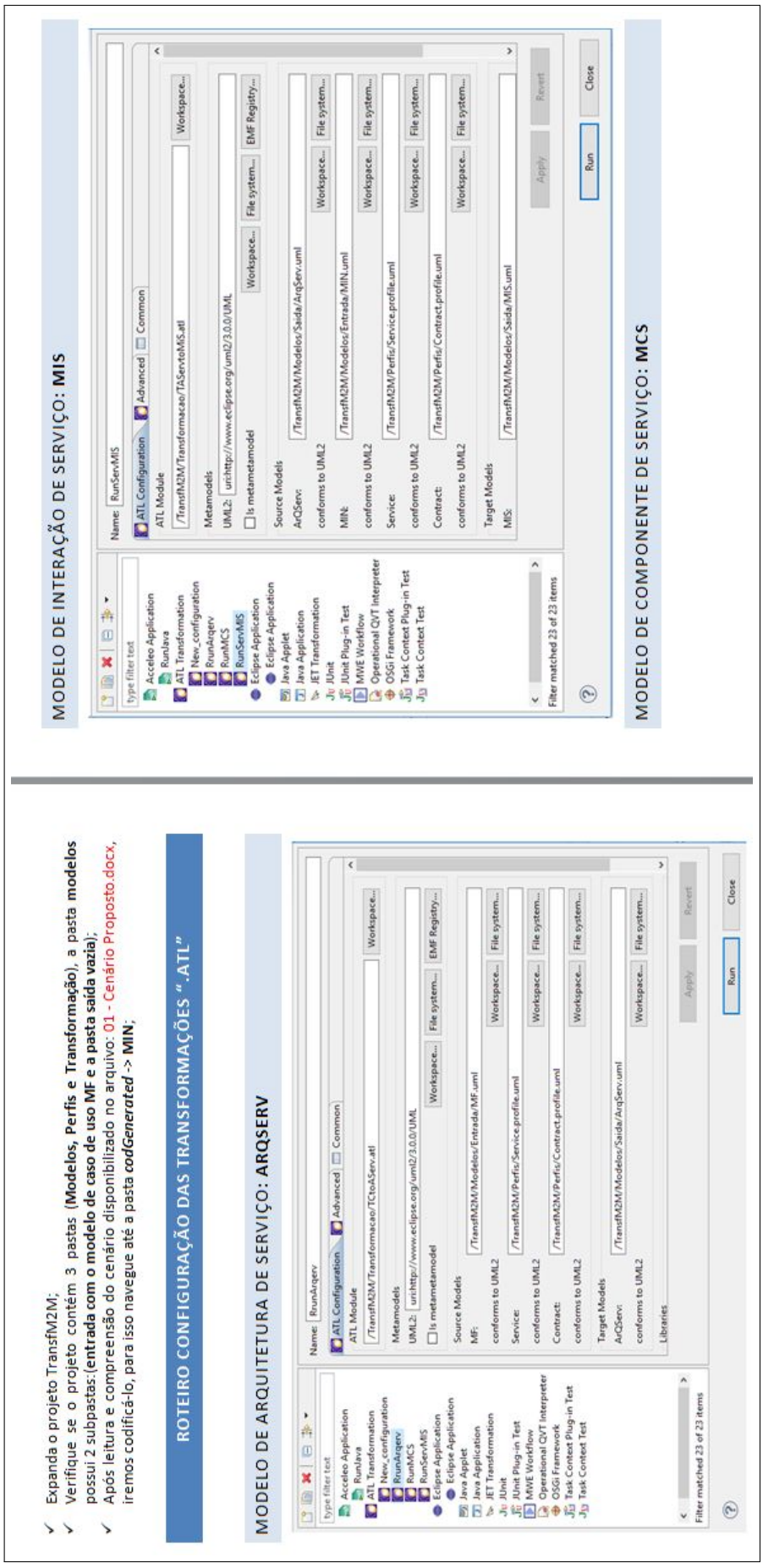


Figura A.19 04-Roteiro de Configuração e Instalação Páginas 07 e 08

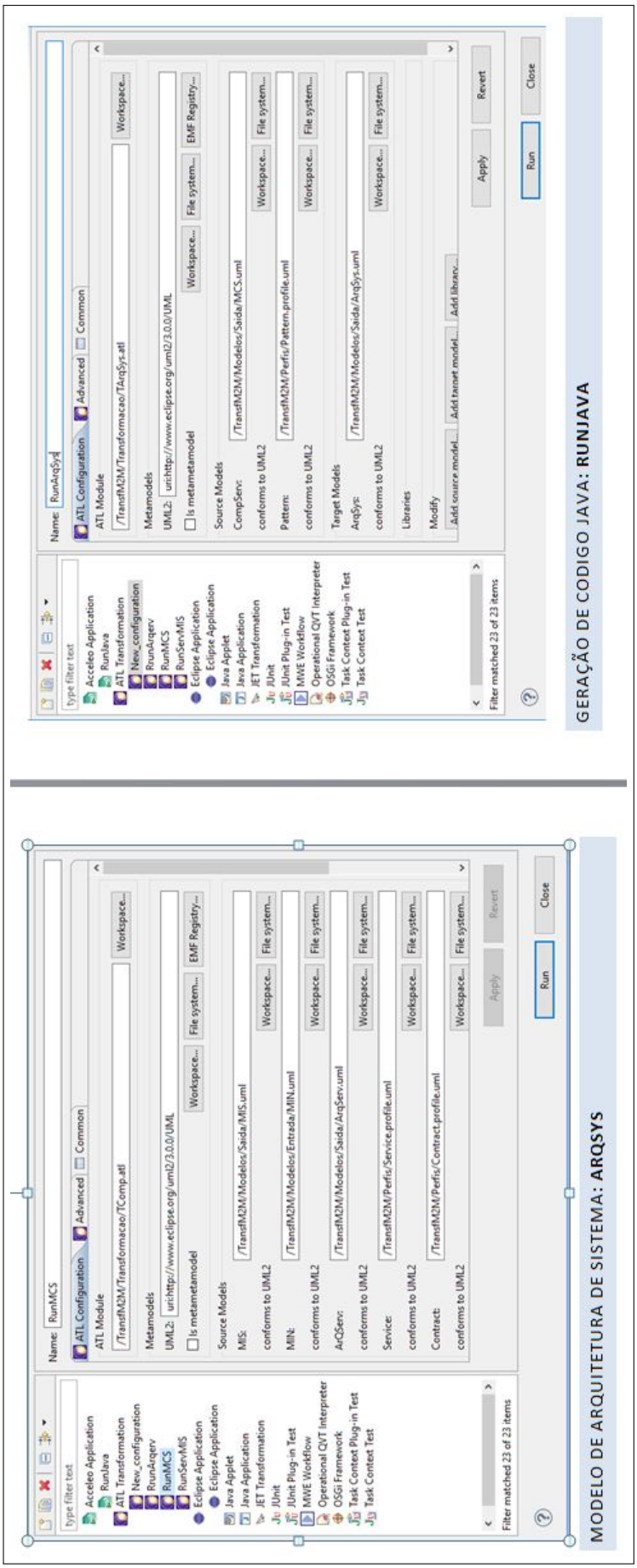


Figura A.20 04-Roteiro de Configuração e Instalação Páginas 09 e 10

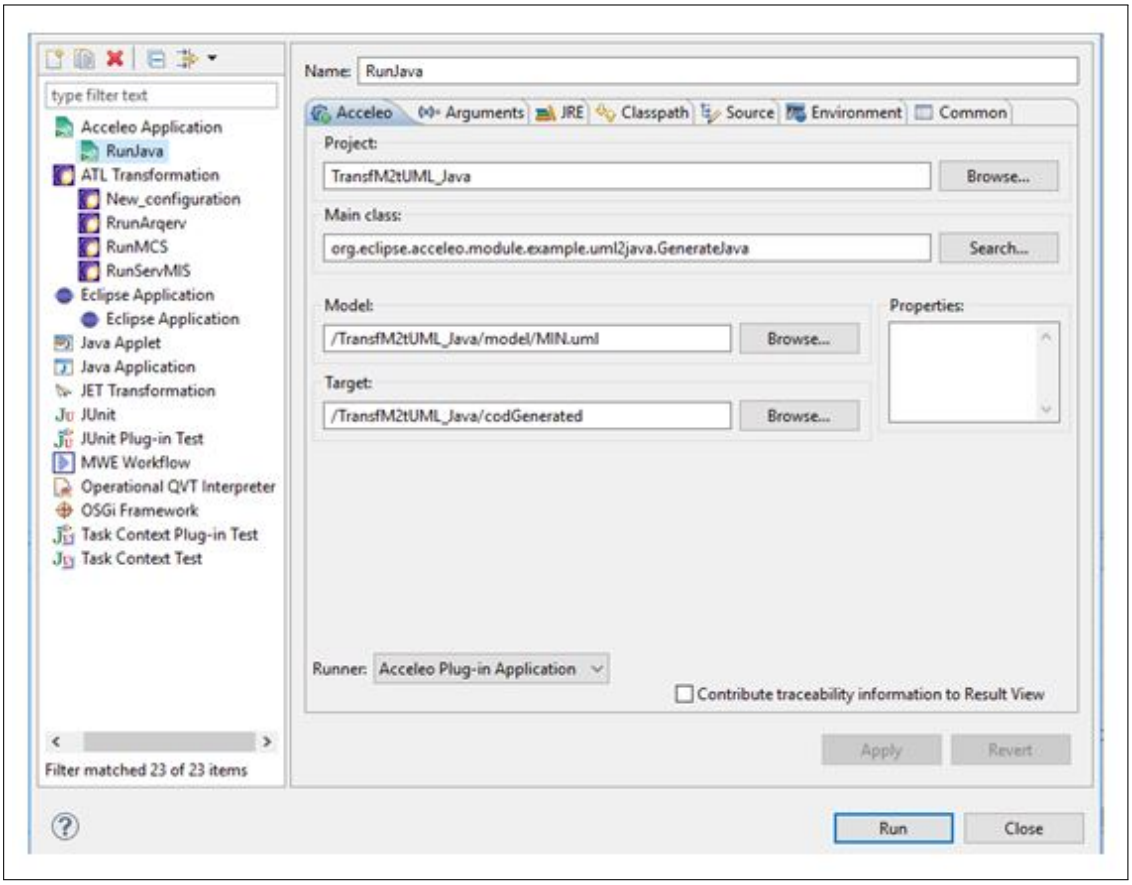


Figura A.21 04-Roteiro de Configuração e Instalação Página 11

Apêndice

B

Este apêndice apresenta os elementos a exemplo de : Diagrama de Caso de Uso (Modelo de Funcionalidade), Diagrama de Classes (Modelo de Informação de Negócio) e Classes (Código-Fonte Java) pertencentes/disponibilizados no metaprocesso ScrumDDM da versão.

ELEMENTOS DA INSTÂNCIA DO METAPROCESSO SCRUMDDM

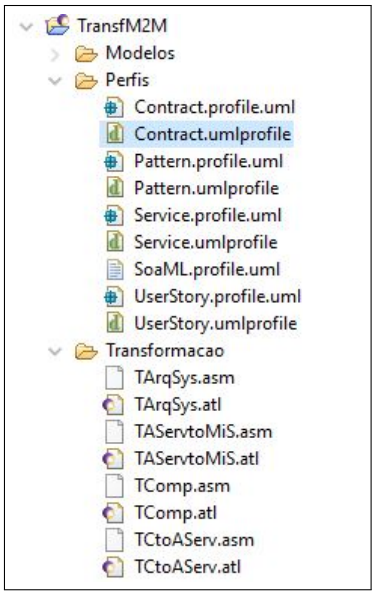


Figura B.1 Perfis em SOAML e Transformações em Acceleo M2M e M2T (SALES, 2017).

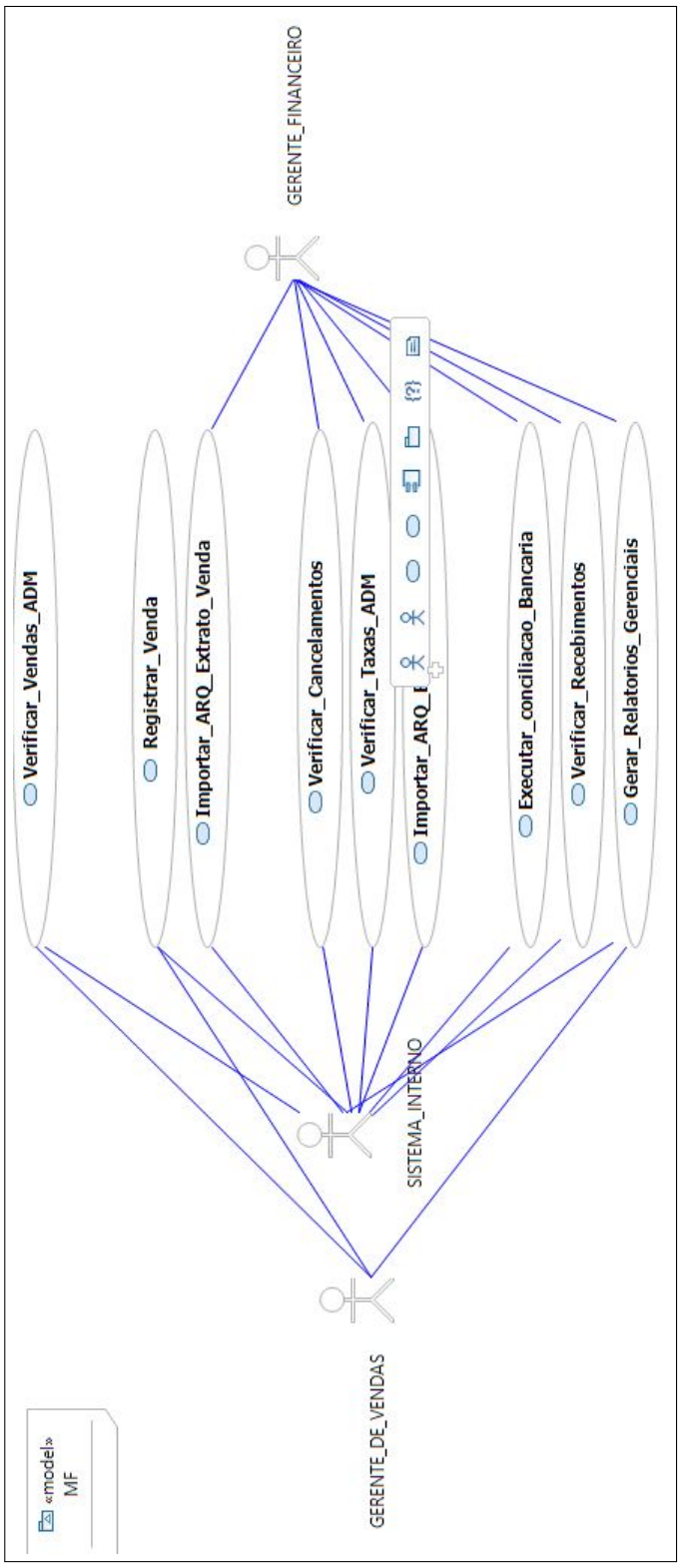


Figura B.2 Diagrama de Caso de Uso/Modelo de Funcionalidade da versão.

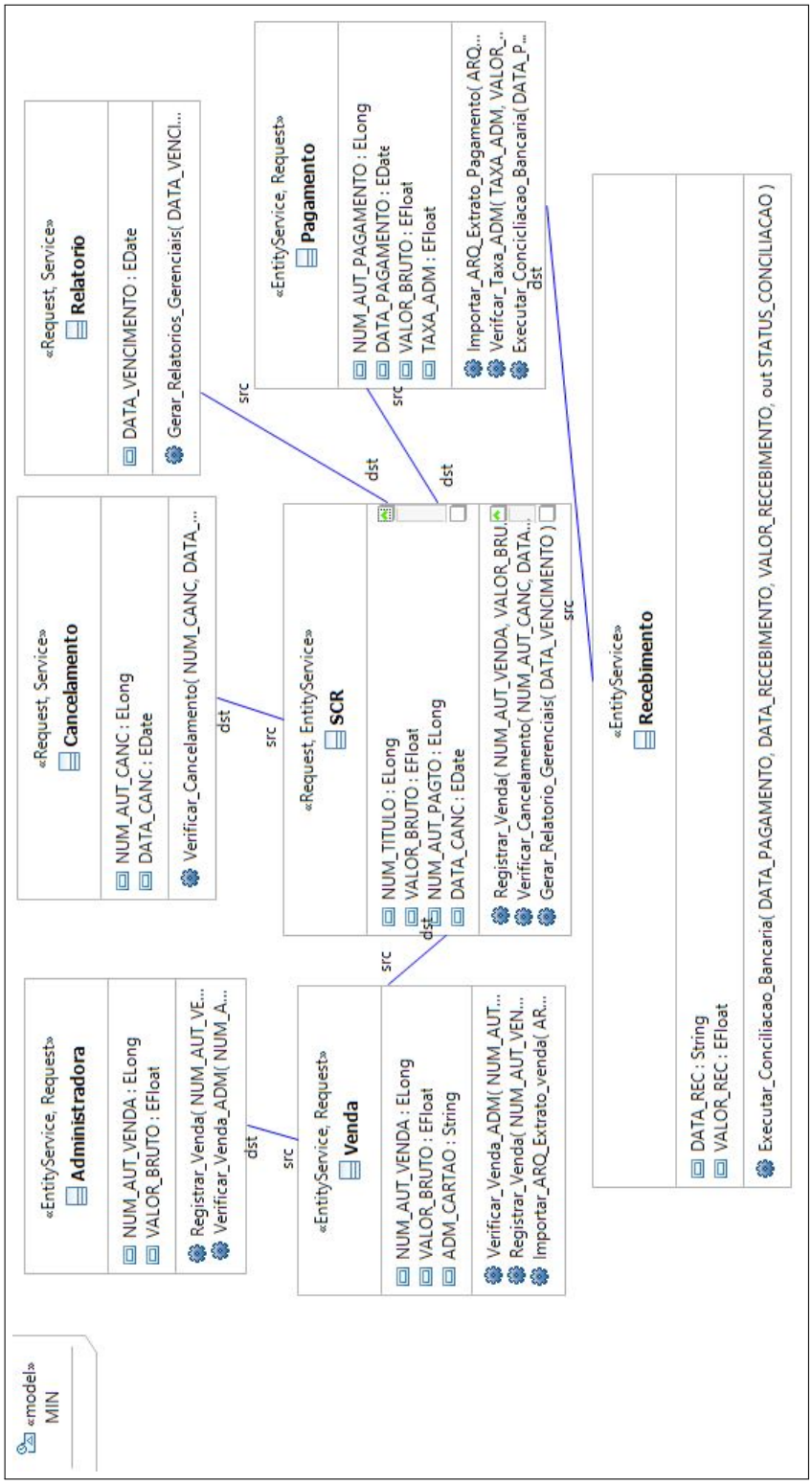


Figura B.3 Diagrama de Classes/Modelo de Informação de Negócio da versão.

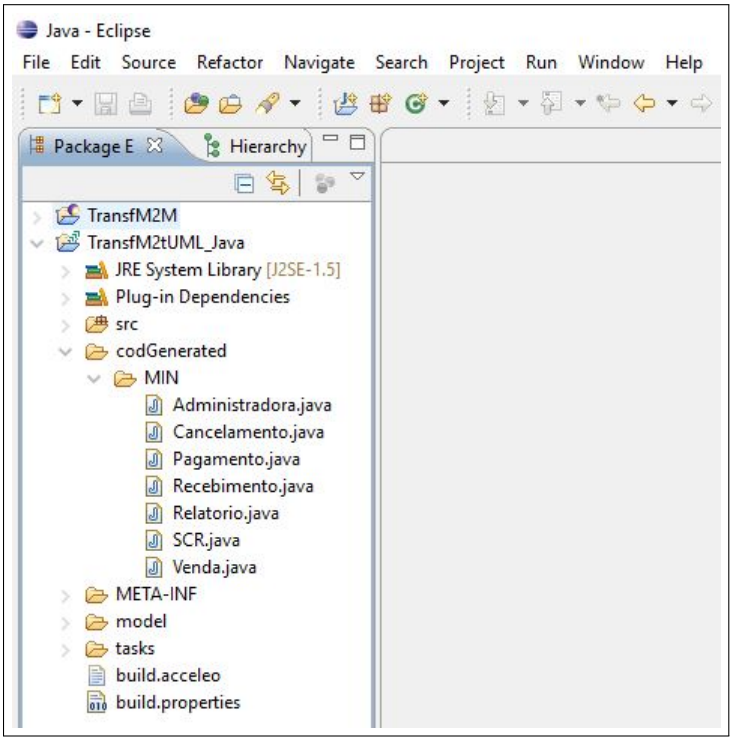


Figura B.4 Todas classes da versão, resultado da transformação M2T

```
Administradorajava
 2⊕ * Generated with MTL UML 2 Java example
 4 package MIN;
 5
 6 // Start of user code for imports
 7⊕import java.io.File;
 9 // End of user code
10 public class Administradora {
12⊕    * the VALOR_BRUTO attribute.
14    private Float VALOR_BRUTO;
16⊕    * the NUM_AUT_VENDA attribute.
18    private Long NUM_AUT_VENDA;
20⊕    * the VALOR_BRUTO getter.
23⊕    public Float getVALOR_BRUTO() {
26
28⊕    * the VALOR_BRUTO setter.
31⊕    public void setVALOR_BRUTO(Float p_VALOR_BRUTO) {
35⊕    * the NUM_AUT_VENDA getter.
38⊕    public Long getNUM_AUT_VENDA() {
41
43⊕    * the NUM_AUT_VENDA setter.
46⊖    public void setNUM_AUT_VENDA(Long p_NUM_AUT_VENDA) {
47        this.NUM_AUT_VENDA = p_NUM_AUT_VENDA;
48    }
50⊕    * the src attribute.
52    private Venda src;
54⊕    * the dst attribute.
56    private Administradora dst;
58⊕    * @param NUM_AUT_VENDA
64⊕    public void Registrar_Venda(Long NUM_AUT_VENDA, Date DATA_VENDA, Float VALOR_BRUTO, String ADM_VENDA, File MSG_RETORNO) {
70⊕    * @param NUM_AUT
74⊕    public void Verificar_Venda_ADM( Long NUM_AUT_VENDA, Float VALOR_BRUTO,String MSG_RETORNO) {
79 }
80 /** @author MTI */
```

Figura B.5 classe Administradora da versão, resultado da transformação M2T

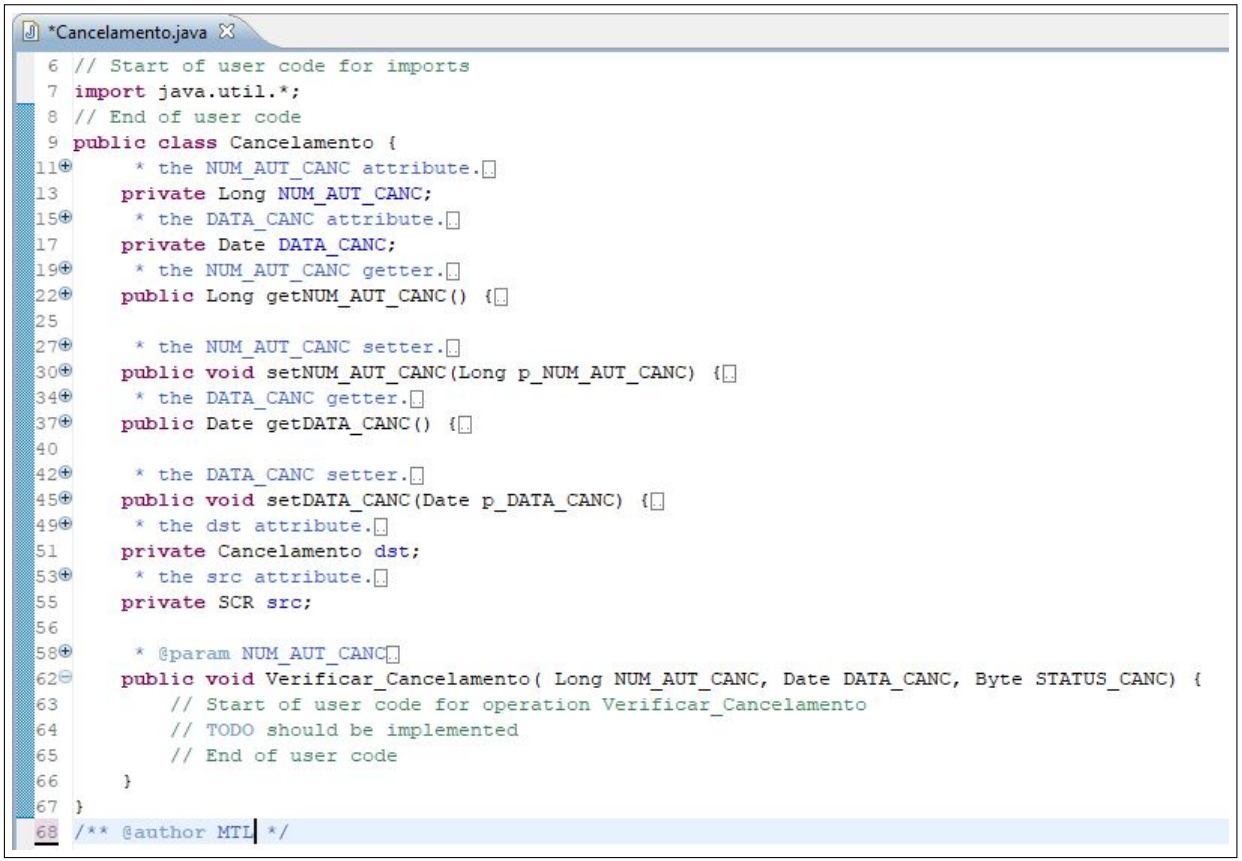


Figura B.6 classe Cancelamento da versão, resultado da transformação M2T

*Pagamento.java

```
2⊕ * Generated with MTL UML 2 Java example
4  package MIN;
5  // Start of user code for imports
6⊕ import java.io.File;
8  // End of user code
9  public class Pagamento {
10     private Float VALOR_BRUTO;
11     private Float TAXA_ADM;
12     private Long NUM_AUT_PAGAMENTO;
13     private Date DATA_PAGAMENTO;
14
15⊕    public Float getVALOR_BRUTO() {
18⊕    public void setVALOR_BRUTO(Float p_VALOR_BRUTO) {
21⊕    public Float getTAXA_ADM() {
24⊕    public void setTAXA_ADM(Float p_TAXA_ADM) {
27⊕    public Long getNUM_AUT_PAGAMENTO() {
30⊕    public void setNUM_AUT_PAGAMENTO(Long p_NUM_AUT_PAGAMENTO) {
33⊕    public Date getDATA_PAGAMENTO() {
36⊕    public void setDATA_PAGAMENTO(Date p_DATA_PAGAMENTO) {
39     private Pagamento dst;
40     private SCR dst;
41     private Recebimento src;
42     private Pagamento src;
43⊕    public void Importar_ARQ_Extrato_Pagamento( File ARQUIVO, String MSG_RETORNO) {
48
49⊕    public void Executar_Conciliacao_Bancaria( Date DATA_PAGAMENTO, Date DATA_RECEBIMENTO, Float VALOR_RECEBIMENTO, Byte STATUS_CONCILIACAO) {
54
55⊕    public void Verifcar_Taxa_ADM( Float TAXA_ADM, Float VALOR_BRUTO, Float VALOR_LIQUIDO) {
60 }
61 /** @author MTL */
```

Figura B.7 classe Pagamento da versão, resultado da transformação M2T

*Recebimento.java

```
2⊕ * Generated with MTL UML 2 Java example
4  package MIN;
5
6  // Start of user code for imports
7  import java.util.*;
8  // End of user code
9  public class Recebimento {
10
11     private String DATA_REC;
12     private Float VALOR_REC;
13
14⊕    public String getDATA_REC() {
17⊕    public void setDATA_REC(String p_DATA_REC) {
21
22⊕    public EFloat getVALOR_REC() {
25
26⊕    public void setVALOR_REC(Float p_VALOR_REC) {
29
30     private Pagamento dst;
31
32     private Recebimento src;
33
34⊖    public void Executar_Conciliacao_Bancaria( Date DATA_PAGAMENTO, Date DATA_RECEBIMENTO, Float VALOR_RECEBIMENTO, Byte STATUS_CONCILIACAO) {
35        // Start of user code for operation Executar_Conciliacao_Bancaria
36        // TODO should be implemented
37        // End of user code
38     }
39 }
40 /** @author MTL */
```

Figura B.8 classe Recebimento da versão, resultado da transformação M2T

Figura B.9 classe Relatorio da versão, resultado da transformação M2T

*SCR.java

```
2  * Generated with MTL UML 2 Java example
4  package MIN;
5
6  // Start of user code for imports
7  import java.io.File;
9  // End of user code
10 public class SCR {
11     private Long NUM_AUT_PAGTO;
12     private Date DATA_CANC;
13     private Long NUM_TITULO;
14     private Float VALOR_BRUTO;
15     public Long getNUM_AUT_PAGTO() {
18     public void setNUM_AUT_PAGTO(Long p_NUM_AUT_PAGTO) {
21     public Date getDATA_CANC() {
24     public void setDATA_CANC(Date p_DATA_CANC) {
27     public Long getNUM_TITULO() {
30     public void setNUM_TITULO(Long p_NUM_TITULO) {
33     public Float getVALOR_BRUTO() {
36     public void setVALOR_BRUTO(Float p_VALOR_BRUTO) {
39     private SCR dst;
40     private SCR dst;
41     private SCR dst;
42     private Cancelamento dst;
43     private Pagamento src;
44     private SCR src;
45     private Venda src;
46     private Relatorio src;
47     public void Gerar_Relatorio_Gerenciais(Date DATA_VENCIMENTO) {
52
53     public void Verificar_Cancelamento( Long NUM_AUT_CANC, Date DATA_CANC, Byte STATUS_CANC) {
58
59     public void Registrar_Venda(Long NUM_AUT_VENDA, Date DATA_VENDA, Float VALOR_BRUTO, String ADM_VENDA, File MSG_RETORNO) {
64 }/** @author MTL */
65
```

Figura B.10 classe SCR da versão, resultado da transformação M2T

Figura B.11 classe Venda da versão, resultado da transformação M2T

Apêndice C

Este apêndice apresenta o código das análises descritivas (Média,Mediana, Primeiro Quartil, Terceito Quartil, Desvio Padrão, Mínimo, Máximo e Coeficiente de Variação) desenvolvido para analisar o metaprocesso ScrumDDM.

CÓDIGO DA ANÁLISE DESCRITIVA

```
if(!require("ggplot2")) install.packages("ggplot2") ; library(ggplot2)
if(!require("tidyverse")) install.packages("tidyverse") ; library(tidyverse)
if(!require("reshape2")) install.packages("reshape2") ; library(reshape2)


##este script foi utilizado no experimento controlado ##
##este script analisa os dados do questionario da qualidade da transformacao ##


# Controle de opções gráficas zzz

opt <- theme_bw()+
  theme(axis.title = element_text(face = "bold", color = "black", size = 20),
        axis.text.x = element_text(face = "plain", color = "black",
                                   size = 18, angle = 90),
        axis.text.y = element_text(face = "plain", color = "black", size = 18),
        legend.text=element_text(size=20, face = "bold"),
        legend.title = element_text(size = 20, face = "bold"),
        strip.text.x = element_text(size = 18, face = "bold"),
        strip.text.y = element_text(size = 14, face = "bold"))

#*****************************************************************************
##                              ATRIBUTO: COMPLETUDE/CORRETUDE/TEMPO
#*****************************************************************************

##importação dos dados CORRETUDE/COMPLETUDE ##

QualidadeC<-read.table("ARQUIVO.txt",header=TRUE);
QualidadeC;
attach(QualidadeC);


### Descritiva dos dados
```

Figura C.1 Código da análise descritiva em *R, parte 1

```r
### Descritiva dos dados

# Médias ----

# por resposta

mean <- QualidadeC %>%
  group_by(Grupo) %>%
  summarise_if(is.numeric, mean)

# Média geral

mean_all <- mean %>%
  select(-1) %>%
  rowMeans(1:2) %>%
  as.tibble()

mean_all <- cbind(mean[,c("Grupo")], mean_all)

# Medianas ----

# por resposta

median <- QualidadeC %>%
  group_by(Grupo) %>%
  summarise_if(is.numeric, median)

# Mediana Geral

median_all <- median %>%
  select(-1) %>%
  rowMeans(1:2) %>%
  as.tibble()
```

Figura C.2 Código da análise descritiva em *R, parte 2

```
median_all <- median %>%
  select(-1) %>%
  rowMeans(1:2) %>%
  as.tibble()

median_all <- cbind(median[,c("Grupo")], median_all)

# Desvio Padrão ----

# por resposta

sd <- QualidadeC %>%
  group_by(Grupo) %>%
  summarise_if(is.numeric, sd)

# Geral

sd_all <- QualidadeC %>%
  melt() %>%
  group_by(Grupo) %>%
  summarise(sd(value))



# Quantis ----

p <- c(.25,.5,.75)

p_names <- paste0(p*100, "%")
p_funs <- map(p, ~partial(quantile, probs = .x, na.rm = TRUE)) %>%
  set_names(nm = p_names)
variable <- colnames(QualidadeC[,-c(1)])
```

Figura C.3 Código da análise descritiva em *R, parte 2

```
quantis <-  QualidadeC %>%
  group_by(Grupo) %>%
  summarize_at(vars(variable), funs(!!!p_funs))

quantis_25 <- quantis[,c(1:8)]

quantis_50 <- quantis[,c(1,9:15)]

quantis_75 <- quantis[,c(1,16:22)]

# All

quartis_all <-QualidadeC %>%
  melt() %>%
  group_by(Grupo) %>%
  summarize_at(vars(value), funs(!!!p_funs))


# Minimo --------
min_all <- QualidadeC %>%
  melt() %>%
  group_by(Grupo) %>%
  summarise(min(value))

# por resposta

min <- QualidadeC %>%
  group_by(Grupo) %>%
  summarise_if(is.numeric, min)
```

Figura C.4 Código da análise descritiva em *R, parte 3

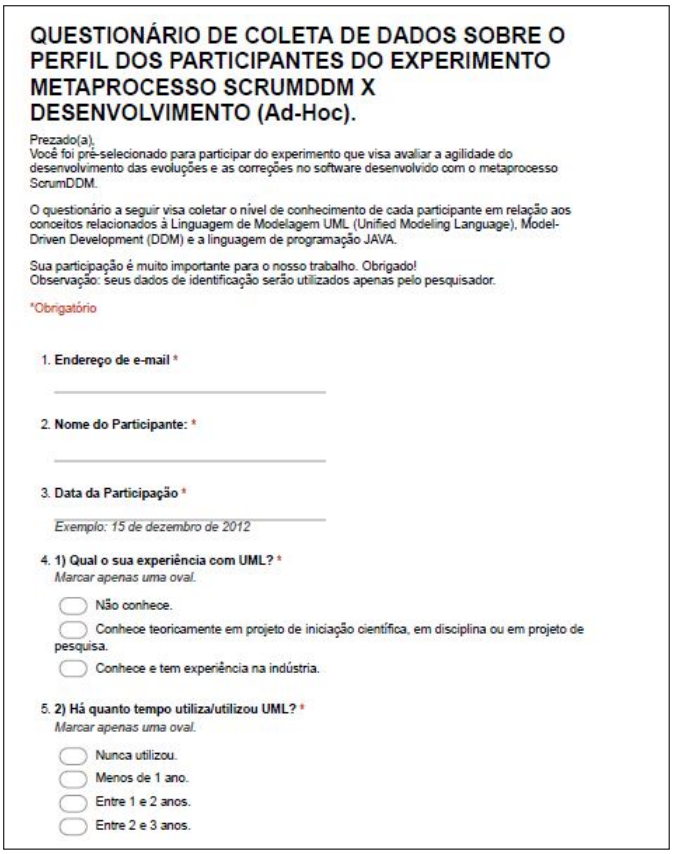
```r
# Maximo--------
max_all <- QualidadeC %>%
  melt() %>%
  group_by(Grupo) %>%
  summarise(max(value))

#Por Resposta
max <- QualidadeC %>%
  group_by(Grupo) %>%
  summarise_if(is.numeric, max)

#CV - Coeficiente de Variação-----
cv_all <- (sd_all[,c(2)]/ mean_all[,c(2)])*100

cv <- (sd[,-c(1)]/mean[,-c(1)])
aux <- c("Dirigido", "ScrummDDM")
cv <- cbind(aux,cv)

# t-test para diferenças de médias ----

dirigido <- QualidadeC %>%
  filter(Grupo == "Dirigido") %>%
  melt()

scrumm <- QualidadeC %>%
  filter(Grupo != "Dirigido") %>%
  melt()

t.test(dirigido[,c(3)], scrumm[,c(3)])

# Existe diferença significativa entre os resultados,
#scrumm possuí valores maiores de qualidade
```

Figura C.5 Código da análise descritiva em *R, parte 4

Apêndice D

QUESTIONÁRIO DE COLETA DE DADOS SOBRE O PERFIL DOS PARTICIPANTES DO EXPERIMENTO CONTROLADO.

figure[htbp]



QUESTIONÁRIO DE COLETA DE DADOS SOBRE O PERFIL DOS PARTICIPANTES DO EXPERIMENTO METAPROCESSO SCRUMDDM X DESENVOLVIMENTO (Ad-Hoc).

Prezado(a),
Você foi pré-selecionado para participar do experimento que visa avaliar a agilidade do desenvolvimento das evoluções e as correções no software desenvolvido com o metaprocesso ScrumDDM.

O questionário a seguir visa coletar o nível de conhecimento de cada participante em relação aos conceitos relacionados à Linguagem de Modelagem UML (Unified Modeling Language), Model-Driven Development (DDM) e a linguagem de programação JAVA.

Sua participação é muito importante para o nosso trabalho. Obrigado!
Observação: seus dados de identificação serão utilizados apenas pelo pesquisador.

*Obrigatório

1. **Endereço de e-mail** *

2. **Nome do Participante:** *

3. **Data da Participação** *
Exemplo: 15 de dezembro de 2012

4. **1) Qual o sua experiência com UML?** *
Marcar apenas uma oval.
- Não conhece.
- Conhece teoricamente em projeto de iniciação científica, em disciplina ou em projeto de pesquisa.
- Conhece e tem experiência na indústria.

5. **2) Há quanto tempo utiliza/utilizou UML?** *
Marcar apenas uma oval.
- Nunca utilizou.
- Menos de 1 ano.
- Entre 1 e 2 anos.
- Entre 2 e 3 anos.

Questionário Página 01

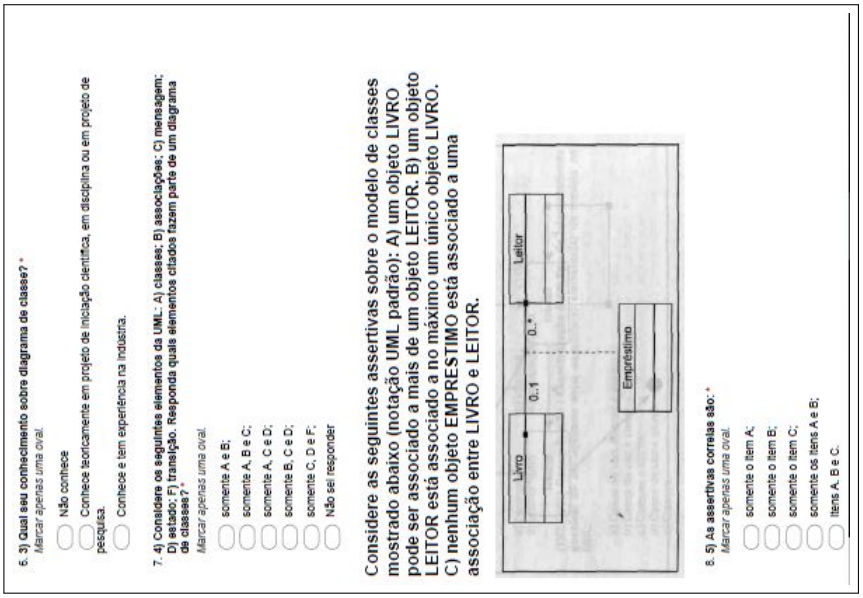


Figura D.1 Questionário Página 02

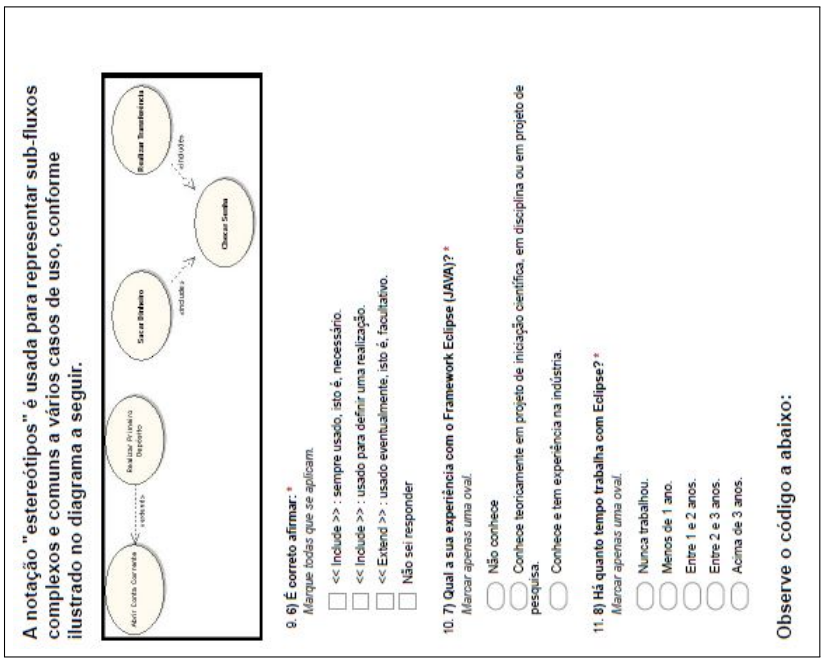
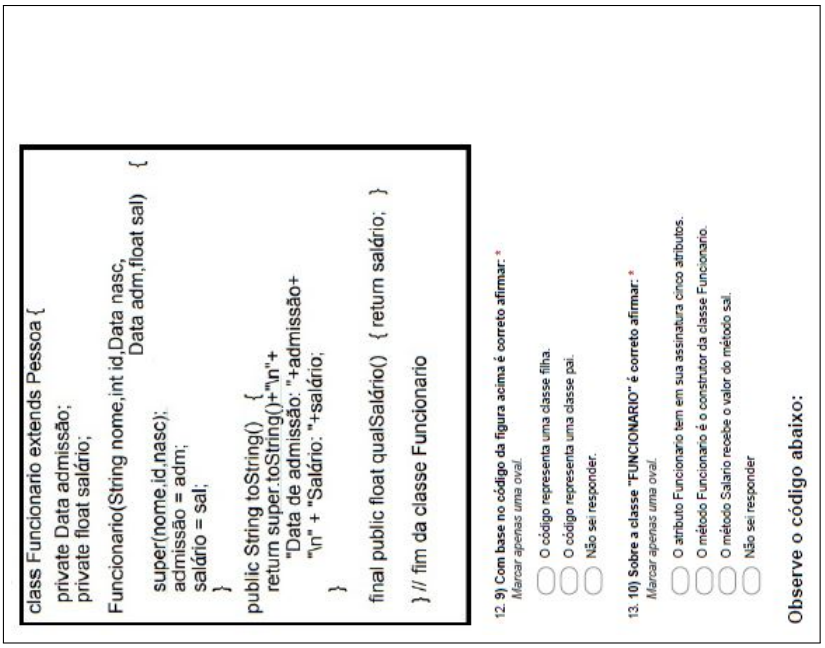


Figura D.2 Questionário Página 03



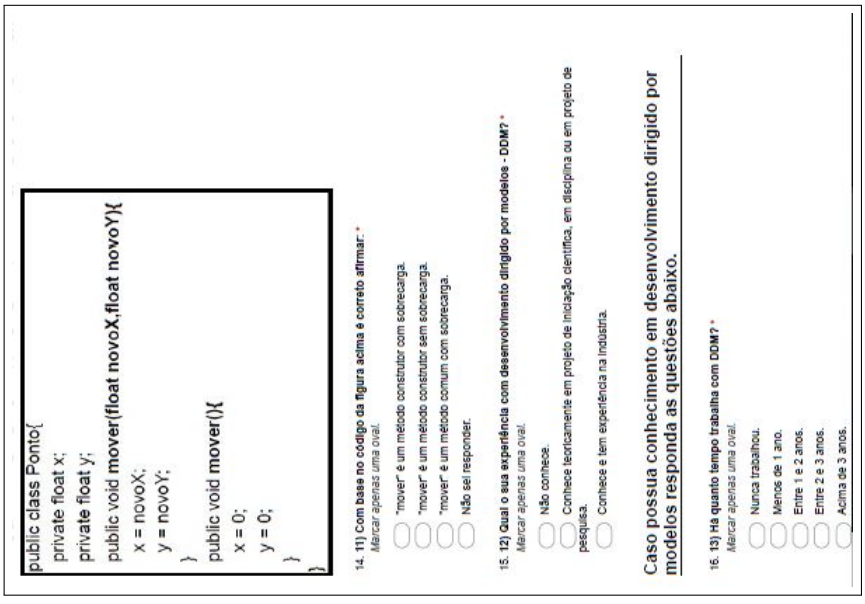
```
class Funcionario extends Pessoa {
  private Data admissão;
  private float salário;
Funcionario(String nome,int id,Data nasc,
                                Data adm,float sal)   {
   super(nome,id,nasc);
   admissão = adm;
   salário = sal;
   }
public String toString()  {
   return super.toString()+"\n"+
        "Data de admissão: "+admissão+
          "\n" + "Salário: "+salário;
   }

final public float qualSalário()  { return salário;  }

} // fim da classe Funcionario
```

12. **9) Com base no código da figura acima é correto afirmar:** *
Marcar apenas uma oval.
- O código representa uma classe filha.
- O código representa uma classe pai.
- Não sei responder.

13. **10) Sobre a classe "FUNCIONARIO" é correto afirmar:** *
Marcar apenas uma oval.
- O atributo Funcionario tem em sua assinatura cinco atributos.
- O método Funcionario é o construtor da classe Funcionario.
- O método Salario recebe o valor do método sal.
- Não sei responder

Observe o código abaixo:

Figura D.3 Questionário Página 04



```
public class Ponto{
  private float x;
  private float y;
  public void mover(float novoX,float novoY){
    x = novoX;
    y = novoY;
  }
  public void mover(){
    x = 0;
    y = 0;
  }
}
```

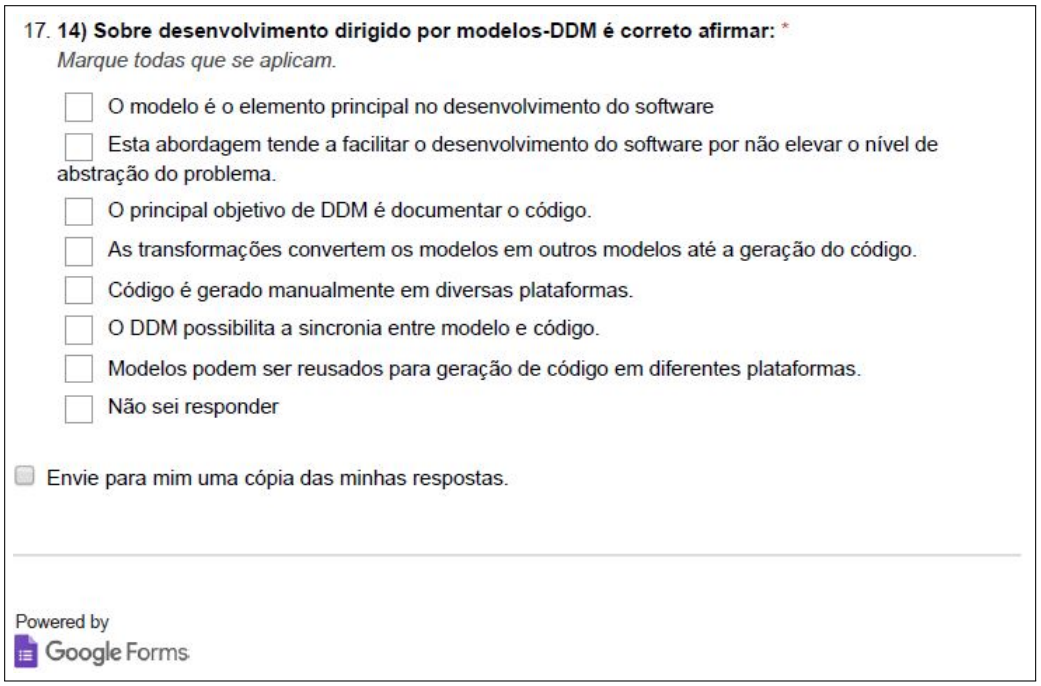
14. **11) Com base no código da figura acima é correto afirmar:** *
Marcar apenas uma oval.
- "mover" é um método construtor com sobrecarga.
- "mover" é um método construtor sem sobrecarga.
- "mover" é um método comum com sobrecarga.
- Não sei responder.

15. **12) Qual o sua experiência com desenvolvimento dirigido por modelos - DDM?** *
Marcar apenas uma oval.
- Não conhece.
- Conhece teoricamente em projeto de iniciação científica, em disciplina ou em projeto de pesquisa.
- Conhece e tem experiência na Industria.

Caso possua conhecimento em desenvolvimento dirigido por modelos responda as questões abaixo.

16. **13) Ha quanto tempo trabalha com DDM?** *
Marcar apenas uma oval.
- Nunca trabalhou.
- Menos de 1 ano.
- Entre 1 e 2 anos.
- Entre 2 e 3 anos.
- Acima de 3 anos.

Figura D.4 Questionário Página 05


17. **14) Sobre desenvolvimento dirigido por modelos-DDM é correto afirmar:** *

Marque todas que se aplicam.

- [ ] O modelo é o elemento principal no desenvolvimento do software
- [ ] Esta abordagem tende a facilitar o desenvolvimento do software por não elevar o nível de abstração do problema.
- [ ] O principal objetivo de DDM é documentar o código.
- [ ] As transformações convertem os modelos em outros modelos até a geração do código.
- [ ] Código é gerado manualmente em diversas plataformas.
- [ ] O DDM possibilita a sincronia entre modelo e código.
- [ ] Modelos podem ser reusados para geração de código em diferentes plataformas.
- [ ] Não sei responder

Envie para mim uma cópia das minhas respostas.

Powered by Google Forms

Figura D.5 Questionário Página 06