Bruno Felipe Arndt

MME-MDD: Um Método para Manutenção e Evolução de sistemas baseados no MDD

Brasil

Março, 2016

Bruno Felipe Arndt

MME-MDD: Um Método para Manutenção e Evolução de sistemas baseados no MDD

Exame de Defesa apresentado ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software. Orientador: Prof. Dr. Daniel Lucrédio.**Versão Revisada**

Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Programa de Pós-Graduação

Orientador: Daniel Lucrédio

Brasil

Março, 2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Arndt, Bruno Felipe
A747m MME-MDD : um método para manutenção e evolução de sistemas baseados no MDD / Bruno Felipe Arndt. -- São Carlos : UFSCar, 2016.
176 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. MDD (Model-Driven Development). 2. Gerador de código. 3. Migração de código. 4. Método orientado a geração de código. 5. Sincronização. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado do candidato Bruno Felipe Arndt, realizada em 08/03/2016.

Prof. Dr. Daniel Lucrédio
(UFSCar)

Profa. Dra. Rosângela Aparecida Dellosso Penteado
(UFSCar)

Prof. Dr. Rohit Gheyi
(UFCG)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Rohit Gheyi e, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Bruno Felipe Arndt .

Prof. Dr. Daniel Lucrédio
Presidente da Comissão Examinadora
(UFSCar)

Este trabalho é dedicado a Deus, minha namorada Camilla
e aos meus pais Maurício e Beatriz.

Agradecimentos

Eu agradeço, primeiramente, a Deus pela vida e pelas lutas diárias.

Eu agradeço ao meu orientador, Professor Doutor Daniel Lucrédio, pela ajuda, compreensão e por toda a orientação durante este projeto.

Eu agradeço à CAPES, pelo apoio financeiro no desenvolvimento desta pesquisa.

Eu agradeço à Professor Doutora Rosângela Aparecida Delloso Penteado e ao Professor Doutor Valter Vieira de Camargo, pelas sugestões e correções apresentadas no exame de qualificação, que foram de suma importância para a realização desta dissertação.

Eu agradeço a todos os professores do Departamento de Computação da Universidade Federal de São Carlos, por todo o conhecimento e experiência compartilhados.

Eu agradeço a todos os participantes do experimento, que contribuíram bastante para com o desenvolvimento deste trabalho.

Eu agradeço especialmente a minha namorada Camilla, por todo apoio, incentivo, ajuda, paciência e companheirismo, e, também, aos meus amigos Mateus e Jorge, pelo apoio, descontração e ajuda.

E, por último, agradeço aos meus pais, Beatriz e Maurício, por toda a ajuda e apoio durante toda a minha vida.

"Do. Or do not. There is no try."
Yoda

Resumo

O MDD (*Model-Driven Development*) tem como proposta a redução da distância semântica entre os domínios problema e solução/implementação. Para isso, são utilizadas algumas ferramentas, sendo o gerador de código comumente usado neste contexto. Os geradores de código são frequentemente implementados com a utilização de *templates*. Para facilitar este tipo de implementação, usualmente é empregado uma Implementação de Referência (IR), favorecendo a evolução/manutenção do software. Contudo, a IR traz a necessidade do processo de migração de código, que consiste na sincronização entre o seu código-fonte e os *templates*, sendo que este é responsável por 20 a 25% do tempo gasto no desenvolvimento. Na literatura não há relatos de solução automatizada, mas o grupo no qual esta pesquisa se insere vem desenvolvendo ferramentas que automatizam este processo e reduzem o tempo aplicado em algumas tarefas testadas. Porém, cada tarefa tem um desempenho diferente em relação ao tempo gasto e, portanto, a automação de algumas tarefas pode ser desvantajosa. Existem poucos relatos na literatura descrevendo tais tarefas e quais são os passos necessários para realizá-las. O objetivo deste estudo foi a investigação do processo de manutenção e evolução de sistemas baseados em MDD com a finalidade de identificar e descrever os tipos de tarefas de manutenção e evolução. Com base nesse estudo, foi elaborado o método MME-MDD que conduz o desenvolvedor durante o processo de manutenção e evolução de sistemas, visando guiar o desenvolvedor durante a realização de cada um dos tipos de tarefas, com o intuito de maximizar os benefícios da utilização dessa abordagem. O método foi validado por meio de um estudo de caso e um estudo experimental, sendo que o método se mostrou efetivo em grande parte das tarefas testadas. Além disso, os estudos apontaram que a utilização do método proposto trouxe ganho na qualidade do código-fonte.

Palavras-chaves: MDD (Model-Driven Development), Gerador de Código, Migração de Código, Método Orientado a Geração de Código, Sincronização.

Abstract

Model-Driven Development (MDD) is proposed to reduce the semantic gap between problem and solution/implementation domains. Some tools are used, and the code generator is commonly used in this context. These generators are often implemented using templates. A Reference Implementation (RI) favoring the development/maintenance of software facilitates this implementation's types. However, RI requires a code migration process which consists in artifacts' synchronization, and it is responsible for 20% to 25% of the time spent on development. The literature has no describe automatic solutions, but the group that this reaserch was included has develop tools that automate this process and reduce the time spent on some tasks. But each task has a different performance in relation to time spent, so automation of some tasks can be disadvantageous. Few reports describe such tasks and which ones are capable of automatic code migration. The aim of this study is to investigate the maintain and evolution process to identify and describe the types of maintain and evolution tasks that are essentially. Based on this study, a method (MME-MDD) that drives the developer during that proccess to guide the developer during the realization of each task, with the aim to maximize the benefits of this approach. The MME-MDD was validated by a case study and a empirical study and the method showed effective in most of tasks. In addition, studies show that using the proposed method brought a gain in the quality of the source code.

Keywords: MDD (Model-Driven Development), Code Generator, Code Migration, Code Generation Method, Synchronization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
ATL	*Atlas Transformation Language*
BOOSTER	*Business Object Oriented Software Technology for Enterprise Reengineering*
BRC	*Breaking but Resolvable Changes*
BUC	*Breaking and Unresolvable Changes*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CCS	*ACM Computing Classification System*
CIM	*Computacional Independent Model*
CPF	Cadastro de Pessoa Física
CRM	Conselho Regional de Medicina
DSL	*Domain-Specific Language*
DREAM	*DRamatically Effective Application development Methodology*
DSML	*Domain-Specific Modeling Languages*
ES	Engenharia de Software
FSML	*Framework-Specific Modeling Languages*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IR	Implementação de Referência
JET	*Java Emitter Templates*
LOCs	*Lines of Code*
M2C	*Model-to-Code*
M2M	*Model-to-Model*

M2T *Model-to-Text*

MDA *Model-Driven Architecture*

MDA-SDLC *MDA-based System Development Lifecycle*

MDASP *MDA Software Process*

MDD *Model-Driven Development*

MDE *Model-Driven Engineering*

MDSD *Model-Driven Software Development*

MDWA *Model-Driven Web Applications*

MIF *Middleware Intergration Framework*

MME-MDD Método para Manutenção e Evolução de sistemas baseados no MDD

Model Driven RichUbi *Model Driven Process to Construct Rich Interfaces for Context-Sensitive Ubiquitous Application*

MP Modelo de Plataforma ou *Model Plataform*

NBC *Non-Breaking Changes*

ODP *Open Distributed Processing*

PDM *Platform-Description Model*

PDS Processo de Desenvolvimento de Sistemas

PHP *PHP: Hypertext Preprocessor*

PIM *Platform Independent Model*

PI-MT *Plataform Independent - Model Transformation*

PL *Product Line*

PLE *Product Line Engineering*

PSM *Platform Specific Mode*

RTE *Round-Trip Engineering*

RTOS *Real-Time Operating System*

SaaS *Software as a Service*

SME	*Situational Method Engineering*
SQL	*Structured Query Language*
SWEBOK	*Software Engineering Body of Knowledge*
UI	*User Interface*
UML	*Unified Modeling Language*
WYSIWYG	*What You See Is What You Get*
XML	*Extensible Markup Language*

Sumário

1 Introdução

1.1 Contexto

A proposta do MDD (*Model-Driven Development* ou Desenvolvimento Orientado a Modelos) é definida como a redução da distância semântica entre os domínios problema e solução/implementação. Esta redução é realizada por meio da utilização de modelos de alto nível e de sucessivas transformações de refinamento dos modelos até a geração do código-fonte (NEIGHBORS 1980; CZARNECKI e HELSEN 2003; HAILPERN e TARR 2006; FRANCE e RUMPE 2007; PHAM et al. 2007; ANGYAL, LENGYEL e CHARAF 2008; LUCRÉDIO 2009; PAPOTTI, PRADO e SOUZA 2012; JÚNIOR 2013).

Essas transformações são classificadas de acordo com o artefato resultante, podendo ser uma transformação *Model-to-Model* (M2M ou Modelo-para-Modelo), uma transformação *Model-to-Code* (M2C ou Modelo-para-Código) ou uma transformação *Model-to-Text* (M2T ou Modelo-para-Texto). Na primeira, o artefato resultante é outro modelo, já no M2C é apenas o código-fonte e, por fim, no M2T os resultados são, além do código-fonte, outros artefatos textuais, como um teste de unidade ou uma documentação (CZARNECKI e HELSEN 2003; PAPOTTI 2013; JÚNIOR 2013).

Para a transformação dos modelos em código-fonte ou texto, são utilizados geradores de código. Porém, a construção desses geradores é de alta complexidade e representa um grande desafio para o desenvolvedor, pois devem ser capazes de se adaptar a todas as variações dos modelos e produzirem código-fonte correspondente com todas as funcionalidades (CLEAVELAND 1988; MUSZYNSKI 2005; FEILKAS 2006; ANGYAL, LENGYEL e CHARAF 2008; PAPOTTI 2013; POSSATTO 2013).

Geradores de código são comumente implementados utilizando *templates*, que é um arquivo de texto padronizado, sendo instrumentado com construções de seleção, responsáveis por realizar consultas em uma entrada (que pode ser um programa ou outro arquivo textual) e expansão de código-fonte (CLEAVELAND 1988; MUSZYNSKI 2005; FEILKAS 2006; ANGYAL, LENGYEL e CHARAF 2008; PAPOTTI 2013; POSSATTO 2013).

Já a construção e manutenção dos *templates* é uma tarefa onerosa visto que um *template* mistura código gerado com código para geração. Devido a essa complexidade, a existência de uma aplicação-exemplo para servir como base para a construção de *templates*, chamada de Implementação de Referência (IR), tem-se mostrado fundamental (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013).

Nos *templates*, assim como em um software, há a necessidade da manutenção e evolução do mesmo ao longo do seu ciclo de vida, de modo que com a utilização de uma IR há a necessidade da migração do código alterado (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013), para que a sincronia seja mantida.

A Implementação de Referência serve como base para as alterações, depurações e testes das modificações e evoluções necessárias (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013). Depois da sua validação, essas modificações são realizadas nos *templates* por um processo chamado de migração de código que consiste na identificação das alterações do código-fonte da IR e sua referência em um ou mais *templates*. Em seguida, eles são atualizados, mantendo assim a sincronia entre ambos.

Em contrapartida, o processo de migração de código necessita ser repetido toda vez que há alteração na Implementação de Referência e estima-se que seu custo a cada execução é de 20 a 25% do tempo de desenvolvimento. Essa estimativa é baseada no estudo realizado por Muszynski (2005) que observou esse custo na utilização da IR na geração de componentes *Graphical User Interface* (GUI) para aplicações *thick-client*. Portanto, supõe-se que este custo seja o mesmo para a utilização de *templates*.

Com o intuito de reduzir esse elevado custo de migração de código, Possatto (2013) desenvolveu um protótipo que realiza de forma parcialmente automática a migração das alterações realizadas no código da Implementação de Referência para os *templates*. Para que a sincronização desses artefatos possa ocorrer, foi necessário o estabelecimento do mapeamento entre a IR e os *templates*, de tal forma que se possa associar os *templates* com o código gerado correspondente.

Para isso, Possatto (2013) modificou o *plug-in Java Emitter Templates* (JET) para que o mesmo gerasse um arquivo de mapeamento, entre os *templates* e a IR, baseado no estudo de Lucrédio e Fortes (2010). Após essa modificação no *plug-in* JET, Possatto (2013) utilizou o *plug-in Fluorite* (YOON e MYERS 2011) que detecta as alterações realizadas na aplicação-exemplo pelo desenvolvedor. Por fim, o desenvolvedor pode optar pela migração das alterações detectadas e, com isso, o protótipo realiza a sincronização com os trechos de código dos *templates* correspondentes e, após isso, é atualizado o arquivo de mapeamento.

Com a finalização do desenvolvimento de seu protótipo, Possatto (2013) realizou um estudo empírico para verificar a efetividade do mesmo e foi observado que o tempo gasto relatado por Muszynski (2005), com a migração de código foi reduzido pela metade em quase todas as situações testadas. No entanto, o estudo realizado por Possatto (2013) foi bastante focado no protótipo desenvolvido, tendo estudado apenas algumas situações de migração de código, onde seus benefícios já eram esperados. Todavia, a evolução e manutenção dos *templates* envolvem uma grande diversidade de tarefas, e um estudo

nesse sentido ainda se faz necessário. Por este motivo, não se sabia exatamente quando a migração automática era possível nem qual a melhor forma de fazê-la.

Além disso, o protótipo desenvolvido por Possatto (2013) realiza a migração em uma única execução, ou seja, não é possível o desenvolvedor testar as diferentes formas de realizar a modificação e visualizar a propagação de cada uma delas. Como resultado, torna-se difícil a possibilidade do teste de impacto das modificações, sendo que esse consiste na análise e na identificação das prováveis consequências da realização dessa modificação no código-fonte e a realização da estimativa do custo de tempo para efetuá-las. Sendo assim, há algumas desvantagens na não realização desse teste, como a não detecção da propagação da alteração em outras partes do sistema e a dificuldade no planejamento e estimativa dos custos das modificações (ARNOLD e BOHNER 1993; BOHNER 2002; HASSINE et al. 2005).

Motivado por essa limitação, Perini (2015) desenvolveu um ambiente especializado para a edição simultânea de *templates* e código gerado. Esse ambiente foi desenvolvido como um *plug-in* para a IDE Eclipse e recebeu o nome de *Split JET Editor*, sendo que sua interface permite ao desenvolvedor trabalhar no código-fonte da IR e visualizar as modificações sendo automaticamente migradas para os *templates*, para tal, foi utilizado o protótipo desenvolvido por Possatto (2013). Além disso, o *Split JET Editor* mostra para o desenvolvedor aonde a migração de código está sendo realizada no *template* e se tal operação é permitida, pois não é permitido alterar uma região de consulta ao modelo ou diretamente um *template*.

Após o término da criação do *plug-in Split JET Editor*, Perini (2015) realizou o mesmo estudo empírico que Possatto (2013) para verificar a efetividade de seu protótipo e comparar seus resultados com o de Possatto (2013). Nesse estudo foi observado que o tempo gasto utilizando o *Split JET Editor* foi reduzido pela metade em relação ao desenvolvimento manual, contudo houve apenas uma pequena melhora em relação ao ganho relatado por Possatto (2013).

1.2 Motivação

Conforme exposto, o processo de migração de código é complexo e exige muita cautela, para que não haja confusão entre o código gerado e o código para a geração, entre outros desafios citados. Outra complicação nesse processo é a necessidade de dois níveis de compilação, um no nível dos *template* e outro no nível de código gerado (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013).

Com os protótipos de Possatto (2013) e Perini (2015), obteve-se ganho no processo de migração de código. No entanto, a melhor forma de se utilizar tais ferramentas ainda necessita de investigações. Como observado e relatado por Possatto (2013), cada tarefa

de manutenção da IR e dos *templates* tem um desempenho diferente com a migração automática de código. Desta forma, o custo/benefício trazido por esta automação depende diretamente da seleção das tarefas que irão passar pelo processo de migração. Por esse motivo, há a necessidade da escolha de quais tarefas devem passar por este processo.

Contudo, a manutenção e a evolução em sistemas utilizando a abordagem MDD podem englobar outros níveis de abstração, tais como o nível de modelo e de metamodelo, sendo assim, é fundamental que exista um método que guie o desenvolvedor por esses níveis de abstração. Outro ponto existente nesse ambiente é que algumas alterações, tais como a inclusão de uma nova consulta ao modelo, exigem a alteração direta de um ou mais *templates* aumentando, dessa forma, a complexidade da modificação.

Conclui-se que é necessário uma melhor investigação do processo de manutenção e evolução de sistemas, para que as ferramentas desenvolvidas por Possatto (2013) e Perini (2015) tenham seus usos efetivos e maximizados, visando à criação de um método que possa nortear o desenvolvedor no uso dessas ferramentas, facilitando, assim, o uso e o manuseio das mesmas e, consequentemente, diminuindo os custos com a utilização dessa abordagem e o tempo gasto nas migrações de código, na manutenção e na evolução de sistemas baseados em MDD.

1.3 Objetivo

Tendo em vista o exposto, foi proposto a investigação do processo de migração automática de código, de manutenção e de evolução, com o objetivo de identificar e descrever quais os tipos de tarefas estão presentes no processo de manutenção e evolução de sistemas, quais os níveis de abstração englobados por cada tipo de tarefa e, dentre esses tipos, quais são passíveis da utilização das ferramentas de Possatto (2013) e Perini (2015). Após esta investigação, foi elaborado um método que conduz o desenvolvedor durante o processo de manutenção e evolução, com o intuito de maximizar a facilidade e o tempo gasto na manutenção e na evolução de sistemas utilizando a abordagem MDD, por meio de uma série de passos que variam de acordo com o tipo de tarefa que será realizada. Para facilitar a compreensão do método e de todo esse ambiente, foi elaborado um Tutorial composto por vídeos e texto explicativo que demonstra cada passo do mesmo. Além disso, foi criada uma ferramenta baseada no Quadro Kanban para facilitar a organização e o controle das tarefas, conforme descrito por Fitzgerald, Musial e Stol (2014).

1.4 Organização do Documento

Neste capítulo foram apresentados o contexto, a motivação e os objetivos desta dissertação. O restante está organizado da seguinte maneira: no Capítulo 2 são apresen-

tados os fundamentos teóricos relacionados a este trabalho, tais quais, geração de código baseado em *templates*, *templates* para geração de código, MDD e processos de Engenharia de Software; no Capítulo 3 são descritos trabalhos relacionados a métodos para a migração de código, para gerador de código ou para o MDD.

O Capítulo 4 é apresentado o método proposto, bem como a ferramenta baseada no Quadro Kanban e o Tutorial para facilitar o entendimento do método; nos Capítulos 5 e 6 estão descritos os experimentos realizados com o intuito de avaliar o método proposto, sendo que o primeiro é um estudo de caso realizado pelo próprio desenvolvedor e o segundo um estudo experimental.

2 Fundamentação Teórica

Com a constante mudança das regras de negócio e a evolução da computação, há a contínua necessidade da adaptação dos sistemas computacionais a essa realidade. Dessa forma, a construção de um software de qualidade, confiável e facilmente adaptado, tem-se mostrado fundamental (LUCRÉDIO 2009). As áreas de Processos de Engenharia de Software e Reutilização de Software têm buscado constantemente essas características.

A área de Processos de Engenharia de Software, que será apresentada na Seção 2.2, tem enfoque, principalmente, na qualidade e na eficiência da construção dos sistemas computacionais, por meio da descrição de atividades, ações e tarefas que são necessárias a essa construção. Usualmente, essa área utiliza ferramentas com o intuito de automatizar ou auxiliar partes desse processo de construção (PRESSMAN 2011).

Já a Reutilização de Software engloba técnicas e ferramentas que enfocam o aproveitamento de artefatos já produzidos em projetos anteriores, tais como, código-fonte, testes e conhecimento, aumentando, dessa forma, a eficiência e o valor agregado do software que está sendo desenvolvido.

Outra vantagem com a Reutilização é a diminuição da necessidade de correções no sistema. Porém, a falta de uma produtividade nesse ambiente de desenvolvimento, poderá acarretar a perda de projetos e clientes, e, consecutivamente, o enfraquecimento do portfólio da empresa.

Uma das técnicas existentes é o Desenvolvimento Dirigido por Modelos, que utiliza modelos de alto nível e sucessivas transformações para a geração de código fonte, será apresentada na Seção 2.1 e, também, os conceitos relacionados, tais como, transformação de modelo (Seção 2.1.1), geração de código (Seção 2.1.2) e geração de código baseado em *templates* (Seção 2.1.3). Já os conceitos relacionados aos processos, metodologias e métodos de Engenharia de Software estão na Seção 2.2. Por fim, a Seção 2.3 contém as considerações finais deste capítulo.

2.1 Desenvolvimento Dirigido por Modelos ou *Model-Driven Development*

Com a indústria cada vez mais automatizada, houve um aumento na procura por sistemas de software mais complexos. Sendo assim, verificou-se a necessidade de utilizar métodos e ferramentas que tornem a produção de sistemas mais ágil e eficaz, otimizando o tempo gasto e melhorando a qualidade dos mesmos (RAUSCHMAYER, KNAPP e

WIRSING 2004; FRANCE e RUMPE 2007; PHAM et al. 2007; LUCRÉDIO 2009).

Deste modo os sistemas computacionais têm que se adaptar constantemente a essa nova realidade na qual um software não deve ser somente confiável, mas também operar em ambientes variados, comunicar-se com diversos paradigmas de comunicação e se adaptar facilmente às mudanças constantes de nosso mundo e às novas tecnologias (PHAM et al. 2007; LUCRÉDIO 2009).

Todavia essas questões ficam ainda mais críticas em virtude do princípio de se construir o software e reutilizá-lo inúmeras vezes, exigindo que os seus artefatos sejam facilmente adaptados a diferentes contextos, características, plataformas e ambientes (PHAM et al. 2007; LUCRÉDIO 2009). Tais necessidades são somente supridas quando artefatos de alto nível são reutilizados também (NEIGHBORS 1980; KRUEGER 1992; GRISS 1995; FRAKES e ISODA 1994).

Para a solução desses problemas, é proposto o MDD (*Model-Driven Development* ou Desenvolvimento Orientado a Modelos), também denominado de MDE (*Model-Driven Engineering*) (SCHMIDT 2006), MDSD (*Model-Driven Software Development*) (VÖELTER e GROHER 2007), MDA (*Model-Driven Architecture*) (MILLER e MUKERJI 2003) ou MD* (VÖELTER 2008). O MDD consiste na combinação entre programação generativa, linguagens específicas de domínio e transformações de software. Essa técnica enfatiza a importância dos modelos no processo de criação e manutenção do software, deixando de ser somente uma atividade que auxilia as atividades de desenvolvimento e manutenção, tornando-se parte constituinte do software (NEIGHBORS 1980; FEILKAS 2006; LUCRÉDIO et al. 2006; FRANCE e RUMPE 2007; PHAM et al. 2007; LUCRÉDIO 2009; POSSATTO 2013).

A proposta do MDD é definida como a redução da distância semântica entre os domínios problema e solução/implementação os quais são realizados por meio do uso de modelos de alto nível para expressar conceitos do domínio e de sucessivas transformações de refinamento até resultarem no código-fonte (NEIGHBORS 1980; CZARNECKI e HELSEN 2003; HAILPERN e TARR 2006; FRANCE e RUMPE 2007; PHAM et al. 2007; ANGYAL, LENGYEL e CHARAF 2008; LUCRÉDIO 2009; PAPOTTI, PRADO e SOUZA 2012; JÚNIOR 2013).

As transformações realizadas pelo MDD podem, de acordo com seu resultado, serem classificadas em: *Model-to-Model* (M2M ou Modelo-para-Modelo), *Model-to-Code* (M2C ou Modelo-para-Código) ou *Model-to-Text* (M2T ou Modelo-para-Texto). Na primeira o artefato-alvo gerado é outro modelo, já no M2C é um código-fonte e no M2T os resultados são, além do código-fonte, outros artefatos textuais, tais como, casos de teste e diversas documentações (CZARNECKI e HELSEN 2003; PAPOTTI 2013; JÚNIOR 2013).

Com a utilização do M2T e, consecutivamente, a geração de vários tipos de artefatos, há um aumento significativo na produtividade (CZARNECKI e HELSEN 2003; HAILPERN e TARR 2006; FRANCE e RUMPE 2007; PHAM et al. 2007; ANGYAL, LENGYEL e CHARAF 2008; LUCRÉDIO 2009; PAPOTTI, PRADO e SOUZA 2012).

A utilização do MDD com o enfoque nos três níveis de abstração, traz duas principais vantagens: a primeira é relacionada aos modelos, pois quando comparados com as linguagens OO, são muito mais próximos dos conceitos do domínio problema e dos requisitos do sistema, além de serem mais fáceis de se compreender e facilitarem o desenvolvimento (PHAM et al. 2007).

A segunda vantagem é relacionada à independência das tecnologias da implementação, uma vez que os conceitos utilizados nos modelos são menos ligados a essa tecnologia, o software é menos suscetível a mudanças tecnológicas, fazendo a manutenção mais fácil e com menor custo (PHAM et al. 2007).

As vantagens citadas anteriormente não necessariamente serão observadas quando o enfoque for em apenas um dos níveis de abstração, como no nível da *Platform Specific Model* isoladamente, por exemplo. Outras vantagens são: a produtividade, pois as tarefas triviais são automatizadas e o reúso do conhecimento adquirido em outros projetos (JÚNIOR 2013).

A Middleware Company (2003) realizou um estudo de caso com o objetivo de verificar a aplicabilidade do MDD no ambiente corporativo. Nesse experimento foram utilizados duas equipes de desenvolvedores, uma utilizando o MDD e outra desenvolvendo na forma tradicional, sendo que ambos os grupos desenvolveram um sistema Web para Petshop em Java. Ao final do desenvolvimento, a equipe MDD gastou 330 horas e a equipe tradicional gastou 507,5 horas, resultando numa economia de 34,98% horas de desenvolvimento utilizando o MDD.

Contudo, o MDD ainda possui dificuldades técnicas como a complexidade da mudança para o paradigma de desenvolvimento orientado a modelos, a redundância dos artefatos, devido à representação dos diferentes níveis de abstração e a necessidade de um maior conhecimento em Engenharia de Software, principalmente, em modelagem e transformadores de código. Contudo, ressalta-se que a sua adoção tem se mostrado vantajosa (HAILPERN e TARR 2006; LUCRÉDIO 2009; PAPOTTI, PRADO e SOUZA 2012; JÚNIOR 2013).

2.1.1 Transformação de Modelo

O desenvolvimento tradicional de um sistema de software passa por vários níveis de abstração, como código de máquina, código *Assembly* e código-fonte (PAPOTTI 2013). Normalmente, o desenvolvedor não precisa se preocupar com esses níveis, pois a tecnologia

atual de compiladores é capaz de automaticamente traduzir código de um nível para outro.

O MDD tenta aplicar a mesma filosofia em outros níveis de abstração adicionais para os modelos. Conforme representado na Figura 1, geralmente são considerados três novos níveis de abstração: *Computacional Independent Model* (CIM ou Modelos Independentes de Computação), *Platform Independent Model* (PIM ou Modelo Independente de Plataforma) e *Platform Specific Model* (PSM ou Modelo Específico de Plataforma) (OMG 2003; BEYDEDA, BOOK e GRUHN 2005; COSTA, GOMES e CAGNIN 2007; PELLEGRINI et al. 2010; JÚNIOR 2013).

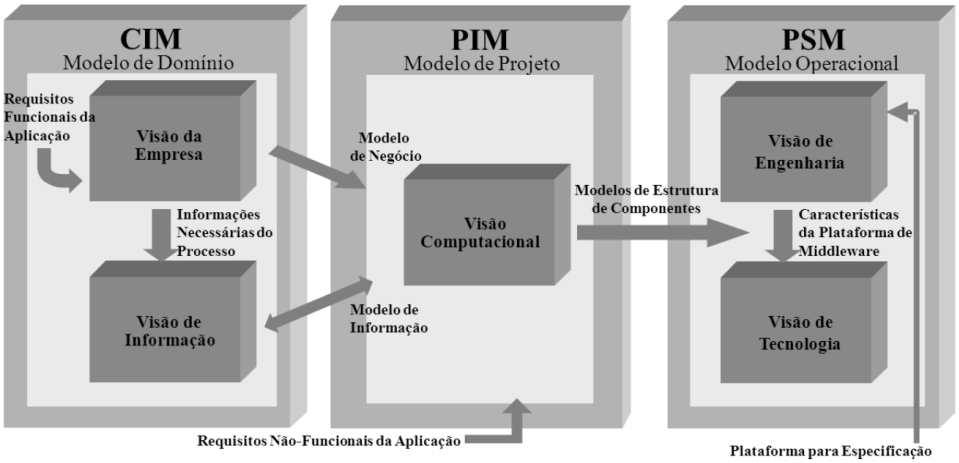


Figura 1: Os níveis de abstração do MDD e seus processos de transformação (PELLEGRINI et al. 2010).

O primeiro nível é o *Computacional Independent Model* (CIM ou Modelos Independentes de Computação) que representa as abstrações de um software de tal forma que não há dependência da computação. Este modelo representa apenas os conceitos do domínio (OMG 2003; BEYDEDA, BOOK e GRUHN 2005; COSTA, GOMES e CAGNIN 2007; PELLEGRINI et al. 2010; JÚNIOR 2013).

Já o segundo nível é chamado de *Platform Independent Model* (PIM ou Modelo Independente de Plataforma) que é um nível de abstração bem detalhado e que tem o enfoque na arquitetura do software que os modelos presentes representam as características independentes de plataforma ou qualquer tecnologia de implementação (OMG 2003; BEYDEDA, BOOK e GRUHN 2005; PARREIRAS e BAX 2005; COSTA, GOMES e CAGNIN 2007; PELLEGRINI et al. 2010; JÚNIOR 2013).

Por fim, o terceiro é chamado de *Platform Specific Model* (PSM ou Modelo Específico de Plataforma) onde há a combinação entre os modelos PIM e as características das tecnologias de implementação (OMG 2003; BEYDEDA, BOOK e GRUHN 2005; PARREIRAS e BAX 2005; COSTA, GOMES e CAGNIN 2007; PELLEGRINI et al. 2010; JÚNIOR 2013). Contudo, mesmo que não sejam utilizados os três níveis de abstração, a adoção do MDD tem-se mostrado efetiva (GAVRAS et al. 2004a; LARRUCEA, DIEZ

e MANSELL 2004; PARREIRAS e BAX 2005; CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

As transformações realizadas pelo MDD podem ocorrer: CIM para CIM (transformação do tipo M2M), CIM para PIM (transformação do tipo M2M), PIM para PIM (transformação do tipo M2M), PIM para PSM (transformação do tipo M2M), PSM para código-fonte (transformação do tipo M2C ou M2T) (KOCH 2006).

Os mapeamentos e as relações entre os níveis de abstração são definidos nas regras das transformações. Um mapeamento é a definição unidirecional do processo de transformação, ao contrário de uma relação que estabelece um relacionamento bidirecional entre os artefatos-fonte e o artefato-alvo (OMG 2003; KOCH 2006).

No MDD, as transformações entre os modelos são fundamentais, pois a partir dessas transformações podemos maximizar o reuso dos artefatos e, com isso, aumentar a qualidade e a rapidez na produção de sistemas. Esse tipo de transformação pode ser classificado em automático, semi-automático e manual. São consideradas automáticas quando não há a necessidade da intervenção humana durante o processo de geração de código; semi-automáticas quando o desenvolvedor escolhe quais elementos do modelo serão transformados e, por fim, manuais quando é o desenvolvedor que realiza a transformação. Em um processo de desenvolvimento de software utilizando MDD é possível a definição de regras de transformação para que este processo ocorra de forma automática (DEURSEN, KLINT e VISSER 2000; KOCH 2006).

Uma das formas de realizar esse tipo de transformação de forma automática é através da utilização de uma DSL (*Domain-Specific Language*). Uma DSL é uma linguagem de programação ou uma linguagem de especificação executável que enfoca em somente um domínio problema. Ela geralmente é uma linguagem declarativa e oferece apenas algumas notações e abstrações, podendo ser textual ou visual (DEURSEN, KLINT e VISSER 2000; LUCRÉDIO 2009). Na literatura são encontradas algumas DSLs que são utilizadas para as transformações entre modelos: ATL (ATLAS Transformation Language) (DEURSEN, KLINT e VISSER 2000), Viatra2 (VARRO e BALOGH 2007), OptimalJ (Compuware 2006), MOF *Query/Views/Transformations* (OMG 2015).

2.1.2 Geração de Código

Em sua essência, um gerador de código é um utilitário que, a partir de uma especificação de alto nível de um problema implementável, transforma automaticamente essa especificação em artefatos-alvo para esse problema. Tais artefatos-alvo podem ser de vários tipos, como códigos-fonte, testes e diversos tipos de documentação (CZARNECKI e EISENECKER 2000; BRAGA 2002; CZARNECKI e HELSEN 2003; HAILPERN e TARR 2006; FRANCE e RUMPE 2007; PHAM et al. 2007; ANGYAL, LENGYEL e CHARAF

2008; LUCRÉDIO 2009; PAPOTTI, PRADO e SOUZA 2012).

Dessa forma, gerador de código é um componente importante do MDD, que tem como objetivo escrever um software a partir de modelos abstratos e refiná-los em código (CLEAVELAND 1988; MUSZYNSKI 2005; FEILKAS 2006; ANGYAL, LENGYEL e CHARAF 2008; POSSATTO 2013). Um Gerador de Código, que é a transformação do tipo M2C ou M2T, pode ser baseado em um desses três tipos de abordagens: *Framework-Specific Modeling Languages* (FSML), padrões de software e *templates* (KOCH 2006; ANGYAL, LENGYEL e CHARAF 2008; PAPOTTI 2013).

O FSML é uma *Domain-Specific Modeling Languages* (DSML) que é projetada para a representação de um *framework*, chamado de *framework*-base. O FSML é composto por uma sintaxe abstrata e um mapeamento entre a sintaxe abstrata e a *Application Programming Interface* (API) do *framework* (ANTKIEWICZ e CZARNECKI 2006; ANGYAL, LENGYEL e CHARAF 2008; ANTKIEWICZ, CZARNECKI e STEPHAN 2009). Ele identifica os conceitos e as características do *framework*-base e os representa em conceitos de linguagem por meio da sintaxe abstrata (ANTKIEWICZ e CZARNECKI 2006; ANTKIEWICZ, CZARNECKI e STEPHAN 2009).

Já o mapeamento entre a sintaxe abstrata e a API do *framework* define como os conceitos e as características do *framework*-base serão implementadas em código-fonte, sendo que este mapeamento é dividido em duas partes. A primeira, chamada de *Forward Mapping*, define como gerar ou atualizar o código-fonte utilizando a sintaxe abstrata. A segunda, chamada de *Reverse Mapping*, define como reconhecer a instância de um conceito no código-fonte. Este mapeamento é definido para cada conceito e característica, permitindo, assim, um aumento no controle sobre o mesmo (ANTKIEWICZ e CZARNECKI 2006; ANTKIEWICZ, CZARNECKI e STEPHAN 2009).

Desse modo, é possível habilitar automaticamente a *Round-Trip Engineering* (RTE) e identificar a origem do código-fonte nos conceitos, qual modelo representa determinado código-fonte e identificar e reconciliar as mudanças, sendo feitas nos conceitos ou no código-fonte (ANTKIEWICZ e CZARNECKI 2006; ANTKIEWICZ, CZARNECKI e STEPHAN 2009).

Outra forma de implementação de geradores de código é utilizando padrões de código. Uma técnica comum é o uso do interpretador lógico que utiliza o padrão Visitor, que foi proposto por Gamma (1995). Com a implementação do interpretador lógico, é possível determinar como e quando interagir com elementos de interesse. As ferramentas GEMS (WHITE, SCHMIDT e GOKHALE 2005), GME (LÉDECZI et al. 2001) e Jamda (BOOCOCK 2003) utilizam esta técnica (HILL e GOKHALE 2012; PAPOTTI 2013).

Há duas técnicas para se implementar o padrão Visitor: interpretação única e interpretação estratégica. A primeira é a mais simplória e, muitas vezes, a mais rápida de

ser implementada. Nela, o núcleo do interpretador lógico interpreta apenas os elementos de interesse, que são chamados de ponto de visitação. Quando cada elemento de interesse é visitado, o interpretador lógico gera artefatos que correspondem ao elemento em particular. Os pontos no interpretador lógico onde um artefato é gerado é chamado de ponto de geração (HILL e GOKHALE 2012).

Já a técnica de interpretação estratégica utiliza o padrão *Strategy*, que também foi proposto por Gamma (1995) para a implementação de uma classe base que contém os pontos para geração como métodos, sendo que o núcleo do interpretador lógico é implementado nessa classe. Por fim, a partir de cada modelo, é gerada uma sub-classe que sobrescreve os pontos de geração correspondente (HILL e GOKHALE 2012).

Por fim, a abordagem baseada em *templates* é a técnica de transformação de modelos de alto nível em código-fonte mais difundida e também é uma maneira prática e robusta de gerar código-fonte utilizando modelos de alto nível (CLEAVELAND 1988; FEILKAS 2006; ANGYAL, LENGYEL e CHARAF 2008; POSSATTO 2013).

2.1.3 Geração de Código baseado em *Templates*

Quando os Geradores de Código são baseados em *templates*, estes contém as regras que são mapeadas nos artefatos-fonte, como por exemplo as ferramentas *Java Emitter Templates* (JET) (VOGEL 2009), AndroMDA (BOHLEN 2003), ArcStyler (ARCSTYLER 2010), Acceleo (ACCELEO 2010) e a proposta por Possatto (2013) e por Perini (2015) (CZARNECKI e HELSEN 2003; PAPOTTI 2013).

Template é um arquivo de texto padronizado qualquer, instrumentado principalmente com construções de seleção e expansão de código, sendo responsável por realizar consultas de parâmetros em uma entrada (um programa ou um arquivo textual), uma *Wizard*, diagramas ou modelos. As informações obtidas por meio destas consultas são utilizadas como parâmetro para produzir código-fonte, conforme exemplificado na Figura 2 (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; ANGYAL, LENGYEL e CHARAF 2008; POSSATTO 2013).

Nessa técnica, primeiramente os *templates* são lidos pelo gerador de código (Figura 2-1) e os parâmetros necessários são consultados nas entradas (Figura 2-2). Por fim, essas informações contidas nos *templates* e nas entradas são processadas pelo gerador de código, resultando no código-fonte da aplicação (Figura 2-3).

Esse processo de geração de código é melhor exemplificado pela Figura 3 e a Figura 4. A partir de um *template* JET e a consulta a um modelo, foi gerado um código-fonte em *Structured Query Language* (SQL).

Ao se executar a transformação JET, ele lê os *templates* (Figura 3-1), e caso necessário, realiza a consulta a uma entrada (Figura 3-2). Após a leitura e interpretação

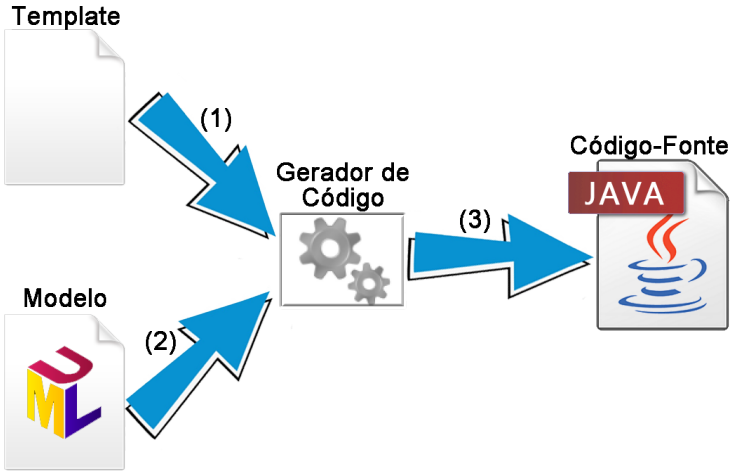


Figura 2: Esquematização da Geração de Código baseado em *Templates* (PERINI 2015).

de todos os *templates* está pronto o código-fonte (Figura 3-3).

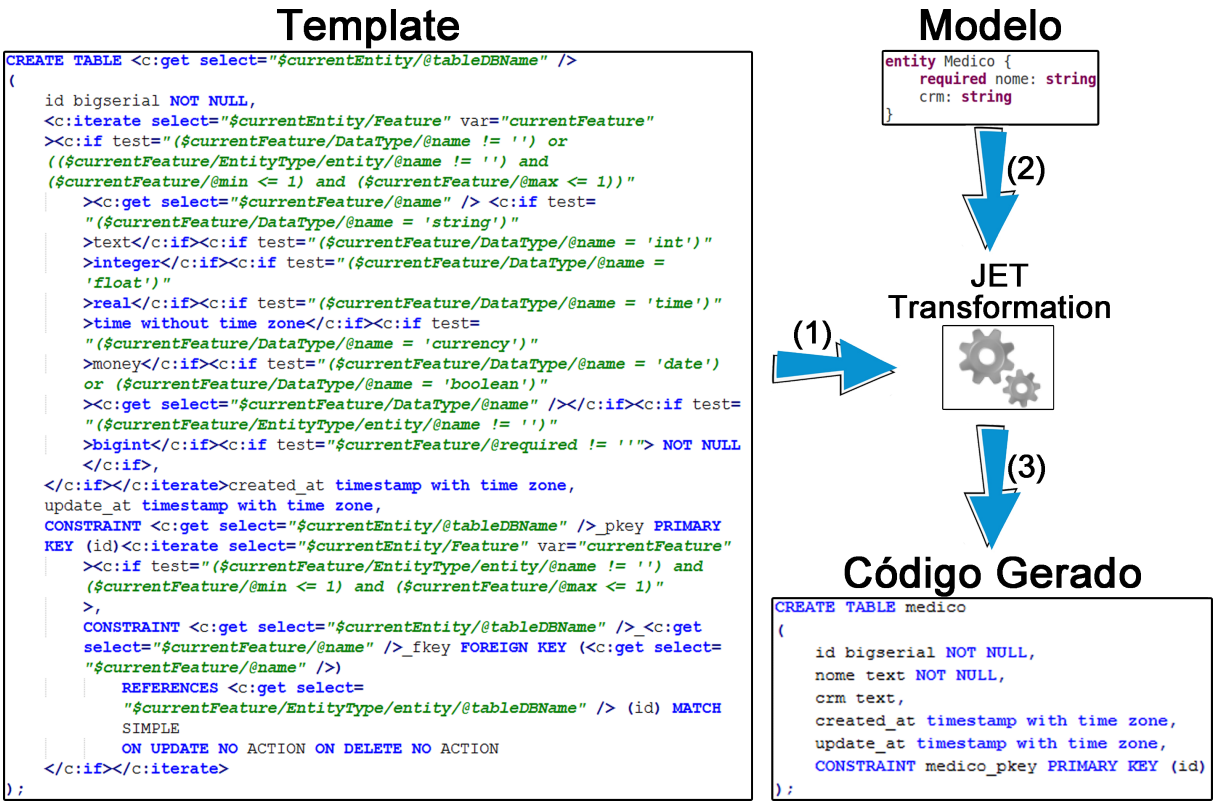


Figura 3: Exemplo da Geração de Código baseado em *Templates*.

Conforme exemplificado na Figura 4, um *template* pode gerar mais de um arquivo, sendo que também é possível que vários *templates* gerem apenas um arquivo de código-fonte e, dessa maneira, há uma dificuldade em identificar qual trecho de um *template* é responsável por determinado trecho no código-fonte resultante, sendo assim há um aumento na complexidade de desenvolver e dar manutenção no mesmo.

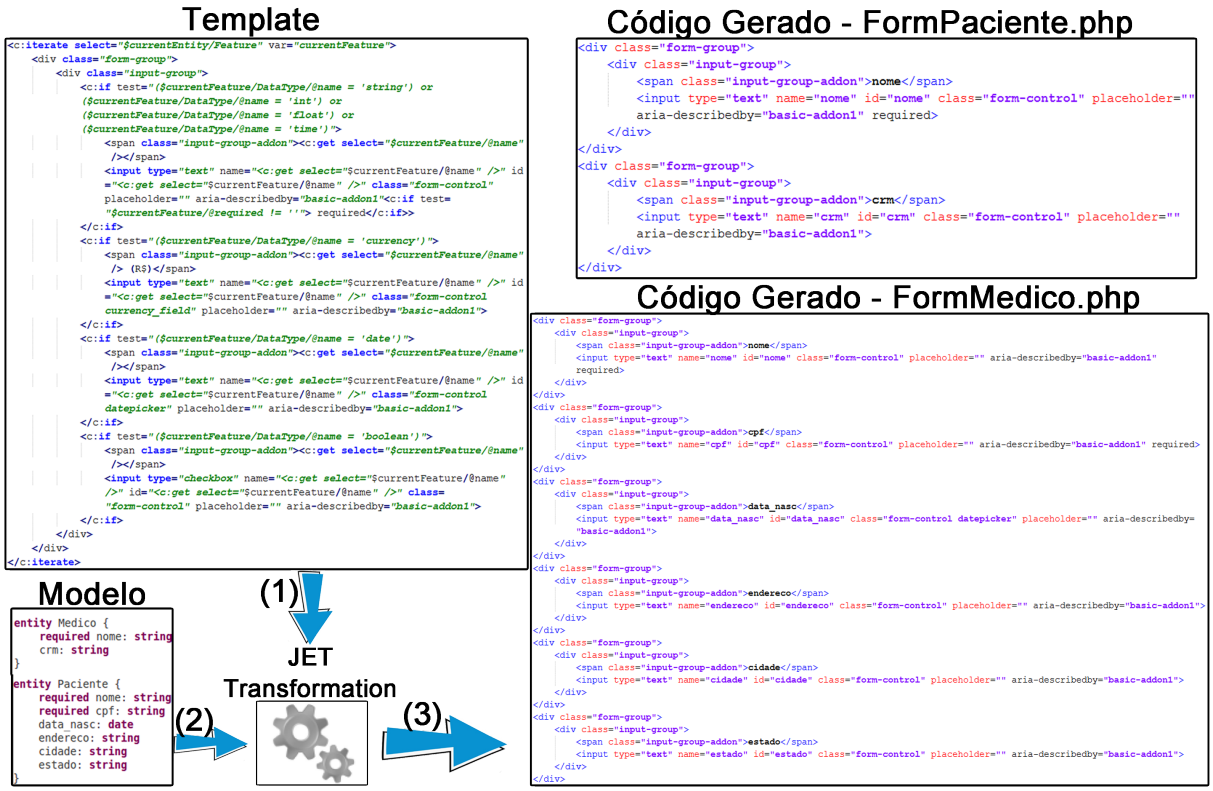


Figura 4: Exemplo da Geração de Código baseado em *Templates* com dois arquivos resultantes.

Além disso, construir *templates* é uma tarefa onerosa, pois um *template* mistura código gerado com código para geração (conforme a Figura 3-1 e a Figura 4-1), aumentando, por sua vez, a complexidade deste processo. Dessa forma, realizar uma edição direta em um *template* mostra-se árdua, o que também limita o desenvolvedor na visualização das modificações realizadas nele (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013).

Tendo em vista a complexidade exposta, a existência de uma aplicação-exemplo, para servir como base para a construção de *templates*, chamada de Implementação de Referência (IR), tem-se mostrado fundamental (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013).

O objetivo dessa é que o desenvolvedor possa trabalhar nesta versão, que utiliza uma forma convencional de desenvolvimento, podendo realizar os testes e a depuração normalmente, sem ter a preocupação com as questões das transformações e geração de código (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; MUSZYNSKI 2005; POSSATTO 2013).

Alguns dos benefícios dessa técnica são: a redução dos custos, o aumento da produtividade, a facilidade da adoção pelos desenvolvedores (devido à possibilidade de gerar código-fonte em qualquer linguagem ou formato) e o código-fonte gerado legível, pois o

template é parecido com a saída desejada e diminuição do índice de erros presentes no software, aumentando, assim, a qualidade do sistema produzido (CLEAVELAND 1988; CZARNECKI e EISENECKER 2000; POSSATTO 2013).

Contudo, a utilização dessa técnica traz a necessidade da sincronização entre a IR e os *templates*. Este processo consiste na identificação das alterações realizadas no código-fonte e a sua referência em um ou mais *templates*. Após essa identificação, eles são atualizados com essas modificações, mantendo, assim, a sincronia entre ambos artefatos.

Muszynski (2005) relata que os problemas da utilização de uma aplicação-exemplo é o custo da migração que fica entre 20 a 25% aproximadamente do custo do tempo de desenvolvimento e as duplicações de código entre a aplicação-exemplo e os *templates* que requerem cautela do desenvolvedor, pois necessitam que a manutenção mantenha a consistência entre ambos.

Uma solução para tal questão é a automatização do processo de migração de código entre a aplicação-exemplo e os *templates*, o que não é uma tarefa trivial. Porém, ainda que essa automação seja parcial, a redução do custo de desenvolvimento é significantemente recompensadora (POSSATTO 2013). No Capítulo 3 estão descritas ferramentas com esse objetivo

2.2 Processos e Métodos de Engenharia de Software

Sabe-se que os sistemas de software são complexos e a sua complexidade está presente desde sua análise até a sua manutenção, podendo apresentar comportamentos não esperados, resultando em consequências negativas para a qualidade do mesmo (FUGGETTA 2000; CHAN 2008).

Encontra-se na literatura uma correlação direta entre qualidade do processo e a do software para que se possam reduzir os erros ou comportamentos inesperados do software e, consequentemente, melhorar a qualidade do mesmo. Para tanto, há a necessidade de cumprir fiel e cuidadosamente todas as fases do processo, verificando assim todas as possibilidades previstas do software, preparando a resposta adequada para cada uma dessas situações (FUGGETTA 2000; CHAN 2008).

Tem-se por definição que processo é uma abordagem adaptável que possibilita à equipe de desenvolvimento de software a escolha das ações e tarefas que serão utilizadas durante todo o ciclo de desenvolvimento. Portanto, processo é um conjunto de atividades, ações, tarefas e ferramentas que são realizadas na criação de algum produto (IEEE 2010; PRESSMAN 2011; BOURQUE e FAIRLEY 2014).

No conjunto de um processo, uma atividade é definida como esforço para atingir um amplo objetivo e é utilizada de forma independente das características do sistema

conforme a interação com o cliente. Por sua vez, uma ação envolve um conjunto de tarefas que resultam num artefato de software assim como o projeto de arquitetura resulta no modelo do projeto. Já uma tarefa foca em um objetivo pequeno e bem definido e produz um resultado tangível bem como a realização de um teste. Por fim, uma ferramenta auxilia o desenvolvedor ou automatiza parte do processo de desenvolvimento (PRESSMAN 2011).

A qualidade de um processo de software é diretamente influenciada pelas pessoas e organizações envolvidas e pela escolha dos procedimentos, métodos e ferramentas que serão utilizados. Porém, a seleção do que deve ser incluído em um processo não é uma decisão trivial, pois há a necessidade da consideração das características das pessoas participantes do desenvolvimento, bem como o nível de conhecimento e domínio das técnicas, dos métodos e das ferramentas da Engenharia de Software, os requisitos, a complexidade e a tecnologia necessária do software, entre outros (FUGGETTA 2000; CHAN 2008).

Em oposição ao processo, que é uma abordagem adaptável, uma metodologia de software estabelece a base completa para o processo de Engenharia de Software através de um pequeno número de atividades que são aplicáveis a todos os projetos (RAUTENBERG et al. 2008; IEEE 2010; PRESSMAN 2011).

Logo, uma metodologia engloba uma série de técnicas e/ou de métodos que tem o intuito de criar uma teoria geral e sistemática para a realização de uma atividade. Além disso, uma metodologia também engloba um conjunto de atividades de apoio, como por exemplo, o controle e acompanhamento do projeto, administração de riscos, garantia da qualidade de software, gerenciamento da configuração de software e gerenciamento da reusabilidade (RAUTENBERG et al. 2008; IEEE 2010; PRESSMAN 2011).

Por sua vez, um método engloba procedimentos sistemáticos que especificam os passos que devem ser executados para alcançar determinados resultados, caracterizando as atividades que devem compor uma atividade mãe. Sendo assim, métodos são orientações sobre como usar a tecnologia e realizar atividades de desenvolvimento de software (FUGGETTA 2000; TEIXEIRA 2014).

Apesar do processo e da metodologia terem o objetivo de nortear o ciclo básico de desenvolvimento e utilizarem métodos e ferramentas, elas diferem quanto à rigorosidade da escolha e do cumprimento dos métodos propostos, visto que uma metodologia cumpre fielmente todos os passos descritos nos métodos. Por sua vez, um processo pode ser personalizado de acordo com as variáveis encontradas nas etapas e com as características da organização (FUGGETTA 2000; TEIXEIRA 2014).

Atualmente, muitos processos de Engenharia de Software são utilizados nas organizações de desenvolvimento de software. Isso monstra uma oportunidade da reutilização da experiência e do conhecimento adquirido e dos artefatos já construídos com o objetivo de aprimorar esses processos, resultando na maximização da qualidade do software e na

diminuição dos custos (HENNINGER 1998; RU-ZHI et al. 2005; COSTA, SALES e REIS 2007; REIS, REIS e NUNES 2001; ROUILLÉ et al. 2013).

Mesmo existindo muitas pesquisas na área de processos de software, ainda há uma grande complexidade na criação de um método que possa ser reutilizado em outros projetos, pois nessa criação há a necessidade da análise dos elementos comuns e variantes de cada projeto. Com isso, é exigido do Engenheiro de Software uma grande experiência e conhecimento de muitos aspectos da Engenharia de Software e da organização (HOLLENBACH e FRAKES 1996; HENNINGER 1998; REIS, REIS e NUNES 2001; ROUILLÉ et al. 2013).

2.3 Considerações Finais

Apesar do OMG (2003) definir diversos padrões e ferramentas para o MDD, ele não define um processo ou uma metodologia de Engenharia de Software para o mesmo. Sendo assim, é necessário que as equipes de desenvolvimento que o utilize, adaptem metodologias e processos tradicionais para a abordagem MDD (ASADI, ESFAHANI e RAMSIN 2010). Contudo, já é possível encontrar na literatura a descrição de processos e metodologias específicas para essa abordagem.

Como concluído por Fuggetta (2000), a área de processos de Engenharia de Software tem papel fundamental na evolução dos sistemas de software, uma vez que a qualidade dos processos, metodologias e métodos influi diretamente na qualidade do software e nos custos de desenvolvimento do mesmo. Por isso é de grande importância o estudo de tais recursos a fim de otimizar o desenvolvimento de software.

Este capítulo apresentou uma revisão da literatura sobre os principais conceitos que serviram de base para o trabalho desenvolvido. Ao longo do capítulo, foram abordados os conceitos fundamentais relacionados ao MDD e a metodologias, processos e métodos em Engenharia de Software. Esta revisão auxilia no entendimento do trabalho elaborado, apresentado no Capítulo 4 e na discussão dos trabalhos correlatos, apresentados no Capítulo 3.

3 Trabalhos Relacionados

A análise dos trabalhos relacionados nesta dissertação foi realizada, primeiramente, por meio de um estudo sobre os projetos anteriores que fazem parte do mesmo projeto de pesquisa desta dissertação. Em seguida, foi feita uma revisão sistemática definida como uma metodologia específica para pesquisa, sendo desenvolvida para coletar e avaliar as evidências disponíveis relacionadas a um tema em questão (KITCHENHAM 2004; BIOLCHINI et al. 2005; BARRETO 2011).

A revisão sistemática consiste na análise dos trabalhos relacionados a este que estão indexados em bases de dados que tem grande impacto na área de Computação e na análise das dissertações e teses indexadas no Banco de Teses da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

3.1 Trabalhos Anteriores

Nesta seção estão descritos os trabalhos anteriores a este, que são as ferramentas desenvolvidas por Possatto (2013) e Perini (2015) que tem como o objetivo a automatização do processo de migração de código.

3.1.1 Protótipo desenvolvido por Possatto (2013)

Possatto (2013) elaborou um protótipo para a sincronização parcialmente automática entre a Implementação de Referência (IR) e os *templates* do gerador de código. Esse protótipo foi desenvolvido como um *plug-in* da *Integrated Development Environment* (IDE) Eclipse, utilizando a ferramenta JET, que é um recurso que permite a geração de código baseada em *templates*.

O objetivo principal de tal trabalho foi a sincronização automática entre IR e *templates*, também conhecida como migração de código. Para que essa sincronização pudesse ocorrer, foi necessário estabelecer o mapeamento físico entre a Implementação de Referência e os *templates*, identificar as alterações realizadas na IR e desenvolver uma ferramenta para realizar a sincronização propriamente dita.

Com o objetivo de estabelecer o mapeamento físico, foi utilizado um arquivo externo de mapeamento e o *plug-in Fluorite* (YOON e MYERS 2011). Na ferramenta JET foi incluído um mecanismo que gera um arquivo com o mapeamento entre a IR e os *templates* o qual contém a relação das posições de cada mapeamento, obtido através da associação dos *templates* com as referências dos modelos de entrada e com o código gerado correspondente, conforme exemplificado na Figura 5.

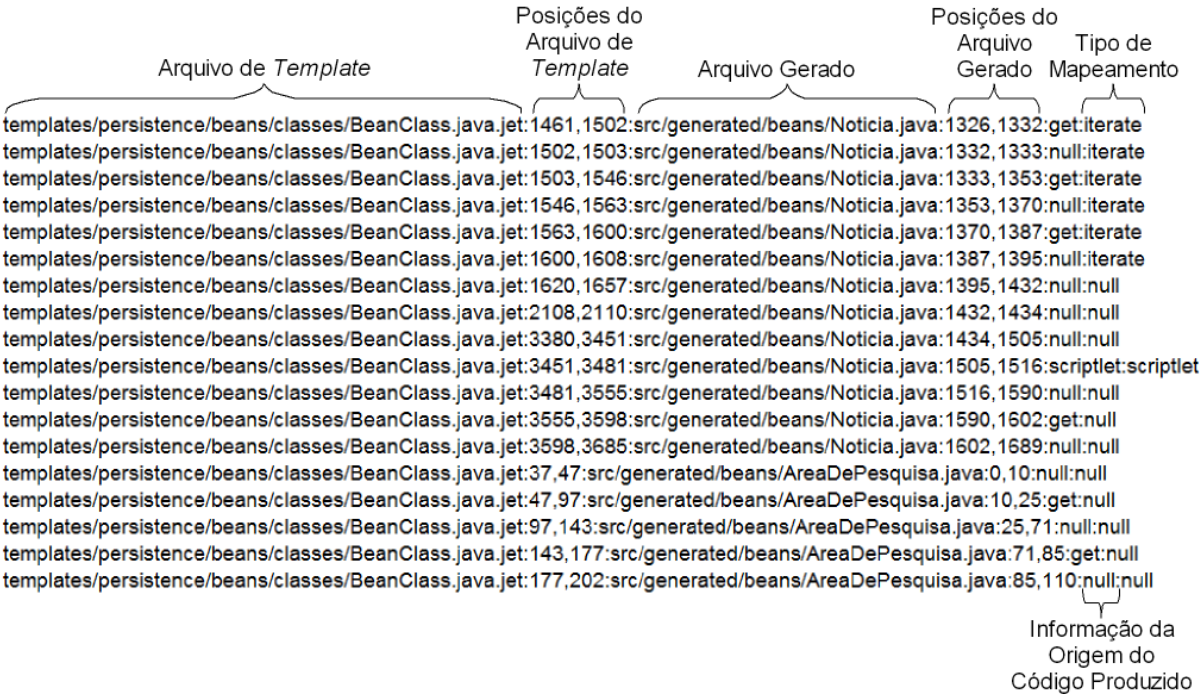


Figura 5: Arquivo de Mapeamento gerado pelo *plug-in* JET (POSSATTO 2013).

Cada linha do arquivo de mapeamento consiste em um mapeamento existente, sendo que em uma linha do arquivo há as seguintes informações e elas são delimitadas pelo caractere ":":

- Caminho e nome do arquivo de *template*;
- Posição de início do mapeamento no arquivo de *template*;
- Posição de término do mapeamento no arquivo de *template*;
- Caminho e nome do arquivo gerado;
- Posição de início do mapeamento no arquivo gerado;
- Posição de término do mapeamento no arquivo gerado;
- Informação da origem do código produzido: Em um *template* pode ocorrer a inclusão de outro *template*, a leitura de uma variável ou a execução de um *scriptlet*; e
- Tipo de Mapeamento: Conforme o mapeamento realizado por Lucredio e Fortes (2010) e descrito no Anexo A, existem 7 tipos de mapeamento, que são Cópia Simples, Substituição Simples, Substituição Indireta, Repetição, Condicional, Inclusão e Novo Arquivo. Contudo, pode não haver um tipo associado ao mapeamento.

O *plug-in Fluorite* detecta todos os eventos ocorridos durante a utilização do editor de código e os armazena em arquivos de log. Por meio da análise desses eventos, é possível estabelecer todas as alterações ocorridas no código-fonte e, dessa forma, foi possível o desenvolvimento do protótipo de migração.

Possatto (2013) realizou um estudo empírico para testar a efetividade de sua ferramenta, sob a hipótese de que qualquer redução dos 25% de tempo gasto na migração relatados por Muszynski (2005) é vantajosa, pois a longo prazo a redução absoluta se mostrará considerável.

As tarefas testadas compreendiam tarefas básicas de manutenção em uma aplicação *Web*, englobando correções, inserções e modificações de seu código-fonte, sendo que cada tarefa foi elaborada por Possatto (2013) de forma específica.

Os participantes desse estudo foram estudantes de graduação e pós-graduação em Ciência da Computação, com envolvimento em projetos de Engenharia de Software e com experiência em abordagens dirigidas a modelos e em manutenção de aplicações *Web*.

Dessa forma, para a realização desse estudo, foram selecionados oito participantes, sendo seis de pós-graduação e dois de graduação. Para a realização do estudo, eles deveriam anotar qual técnica de migração de código foi utilizada (Manual ou Protótipo), qual a tarefa executada (T1, T2, T3 ou T4) e a hora de início e término da execução da tarefa. Em seguida, eles foram divididos em dois grupos balanceados de acordo com suas experiências.

A tarefa T1 consistiu na alteração do tamanho de três campos, a substituição do título de uma página e o *label* de um campo. Já a tarefa T2 foi a modificação do texto do link para adicionar novo cadastro na área administrativa. Por sua vez, a tarefa T3 foi a inclusão de um logo nos formulários de listagem e edição. Por fim, a tarefa T4 foi a alteração do tamanho de um campo.

Esse estudo verificou que, apesar das limitações do protótipo, ele é efetivo na maioria das situações testadas, conseguindo reduzir o tempo gasto na migração pela metade, conforme demonstrado na Tabela 1.

Técnica	Tarefas	Grupo	Média	Soma das Médias	Porcentagem
Manual	T1	G1	00:14	00:51	68,92%
	T2	G2	00:07		
	T3	G1	00:15		
	T4	G2	00:13		
Protótipo	T1	G2	00:06	00:23	31,08%
	T2	G1	00:07		
	T3	G2	00:03		
	T4	G1	00:05		
Total				01:14	100,00%

Tabela 1: Resultados obtidos pelo estudo realizado por Possatto (2013).

O estudo também mostrou que existem tarefas em que a migração automática não traz benefícios, ou seja, em certos casos não há ganho ou perda de tempo utilizando tal abordagem. No entanto, ainda não existiam estudos com o intuito de determiná-las e de caracterizá-las.

Apesar dos benefícios trazidos pelo protótipo desenvolvido por Possatto (2013), a migração segue um processo de execução única. Assim, as modificações realizadas na IR pelo desenvolvedor são propagadas para os *templates*, não possibilitando o teste de impacto das diferentes formas de realizar as modificações, isto é, não é possível identificar as possíveis consequências da realização dessa modificação no software (ARNOLD e BOHNER 1993; BOHNER 2002; HASSINE et al. 2005). Para que isso ocorra, é necessário um processo mais flexível, conforme descrito a seguir.

3.1.2 Um Editor Simultâneo para Facilitar a Evolução de *Templates* de Geração de Códigos (PERINI 2015)

Dando continuidade ao trabalho de Possatto (2013), Perini (2015) desenvolveu um ambiente especializado para a edição simultânea de *templates* e código gerado, possibilitando um maior dinamismo e controle do processo de evolução do software. Esse ambiente especializado recebeu o nome de "Split JET Editor" e foi desenvolvido como uma *view* para a IDE Eclipse. Além disso, ele é integrado ao *plug-in* JET e ao protótipo de Possatto (2013).

A *view* do *Split JET Editor* consiste em duas partes, conforme a Figura 6. A primeira parte, localizada à direita, é onde estão as configurações do mesmo. Já a segunda parte, que está à esquerda, é onde está o status e as informações da edição.

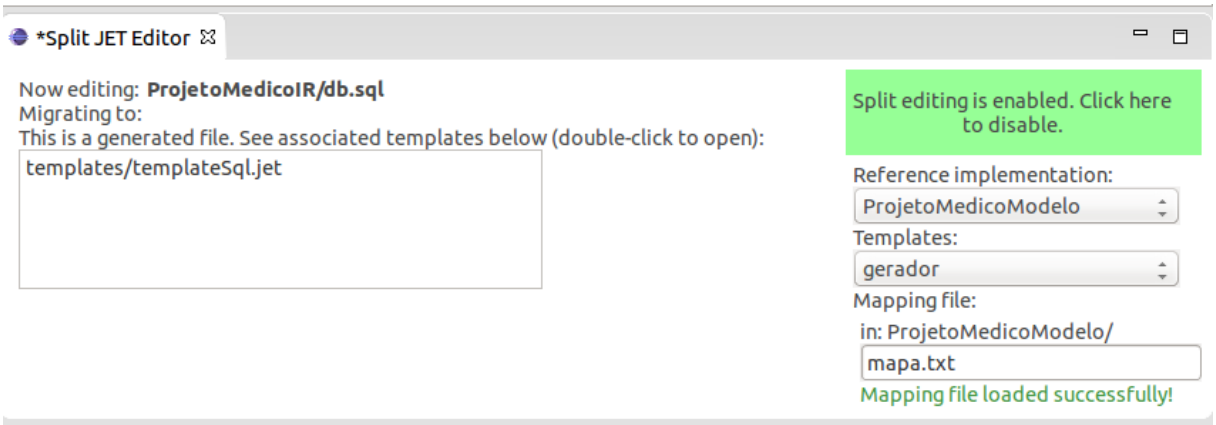


Figura 6: A *view* do *Split JET Editor* desenvolvido por Perini (2015).

Na área de configuração do *Split JET Editor*, conforme a Figura 7, está localizado o botão de ativar/desativar a edição simultânea do mesmo. Ela estará ativa quando o

botão estiver na cor verde e com o texto "*Split editin is enabled. Click here to disable.*" e estará desativada quando o botão estiver na cor vermelha com o texto "*Split editing is disabled. Click here to enable.*".

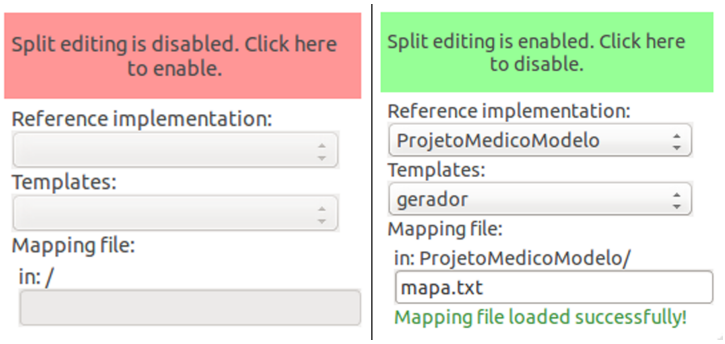


Figura 7: Configuração do *Split JET Editor*.

Quando o *Split JET Editor* está habilitado, é possível configurá-lo. Em "*Reference implementation*" é onde deve ser indicado o projeto que contém o modelo e o arquivo de mapeamento. Já em "*Templates*" é onde deve ser indicado o projeto que contém os *templates*. E, por fim, em "*Mapping file*" é onde deve ser indicado o nome do arquivo de mapeamento. Após ser indicado o arquivo de mapeamento, o *Split JET Editor* tenta carregar o mesmo e apresenta ao usuário se foi possível ou não carregá-lo, sendo que caso o arquivo não possa ser carregado, o *Split JET Editor* não consegue migrar as alterações realizadas e, por esse motivo, não permite que o desenvolvedor realize as modificações na aplicação-exemplo, mas permitindo a realização das modificações nos *templates*.

Já na área de status e informações da edição, conforme a Figura 8, apresenta o nome do arquivo que está sendo editado, o *status* da edição e quais arquivos do código-fonte da IR ou *templates* que são relacionados ao arquivo em edição.

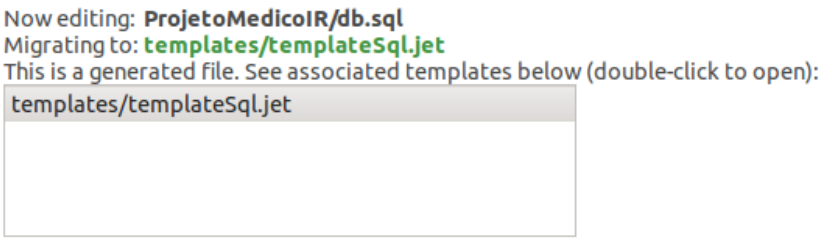


Figura 8: Área de status e informações da edição do *Split JET Editor*.

O *Split JET Editor* tem 5 status possíveis, conforme a Figura 9:

- Edição Habilitada;
- Arquivo de *template* fechado;

- Editando Região de Modelo;
- Editando um arquivo de *template*; e
- Sincronia perdida.

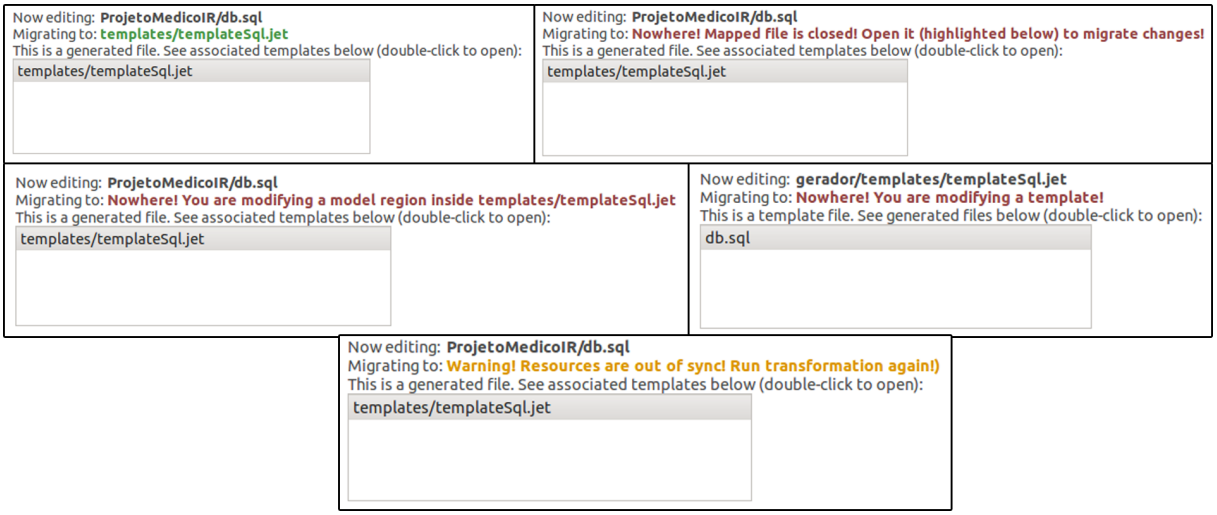


Figura 9: Os 5 status possíveis do *Split JET Editor*.

O *Split JET Editor* tem como status padrão o status **Edição Habilitada**, sendo que nele é permitido somente a edição do código-fonte da IR e é representado pela cor de fundo do texto verde no código-fonte e pela mensagem com o nome do *template* para o qual a migração está sendo realizada em cor do texto verde, conforme a Figura 10.

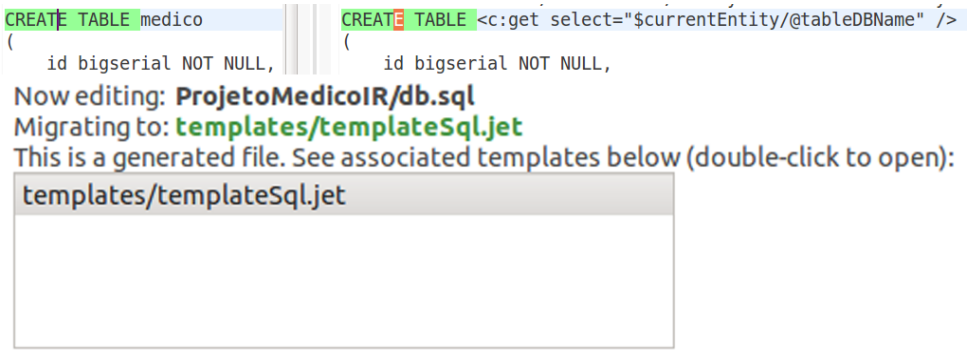


Figura 10: O status "Edição Habilitada" do *Split JET Editor*.

Por sua vez, o status **Arquivo de *template* fechado** ocorre quando o desenvolvedor tenta editar um trecho do código-fonte da IR e seu *template* correspondente não está aberto, impedindo, dessa forma, a migração automática das alterações e, por isso, o *Split JET Editor* impede a edição desse trecho até que o desenvolvedor abra o *template*. Esse status é representado pela cor de fundo do texto vermelho e pela mensagem "*Nowhere!*

Mapped file is closed! Open it (highlighted bellow) to migrate changes!", conforme a Figura 11.

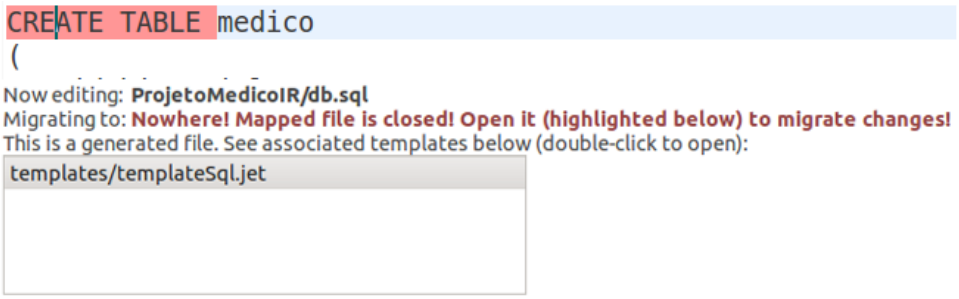


Figura 11: O status "Arquivo de *template* fechado" do *Split JET Editor*.

Já o status do *Split JET Editor* é mudado para o status **Editando Região de Modelo** quando o desenvolvedor tenta alterar uma região do código-fonte da IR que é resultante de uma consulta ao modelo, sendo que esse status é representado pela cor de fundo do texto vermelho e pela mensagem "*Nowhere! You are modifying a model region inside* (nome do template)", conforme a Figura 12.

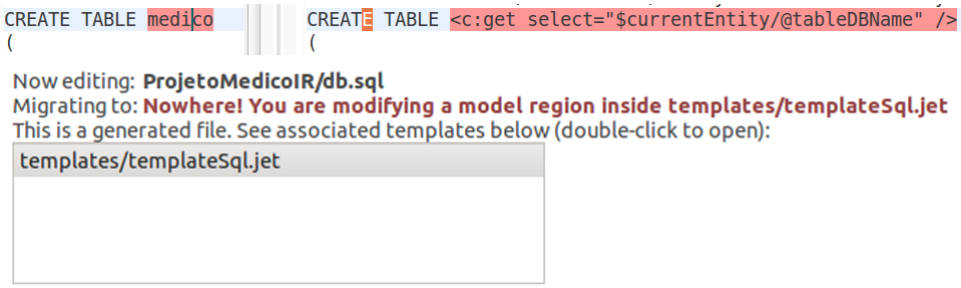


Figura 12: O status "Editando Região de Modelo" do *Split JET Editor*.

O status **Editando um arquivo de *template*** ocorre quando o desenvolvedor tenta editar diretamente um *template* e essa operação não é permitida por esse ambiente, sendo que o mesmo impede a edição e apresenta a mensagem "*Nowhere! You are modifying a template!*", conforme a Figura 13.

Por fim, o status **Sincronia perdida** ocorre quando o desenvolvedor tenta alterar uma região de código-fonte da IR que já teve seu *template* correspondente alterado anteriormente, por outro arquivo ou manualmente pelo desenvolvedor e não houve uma nova execução do *JET Transformation*, sendo assim a sincronia entre o arquivo atual da IR e o *template* foi perdida, mas ainda é permitida a edição. Esse status é representado pela cor de fundo do texto amarelo e pela mensagem "*Warning! Resources are out of sync! Run transformation again!*", conforme a Figura 14.

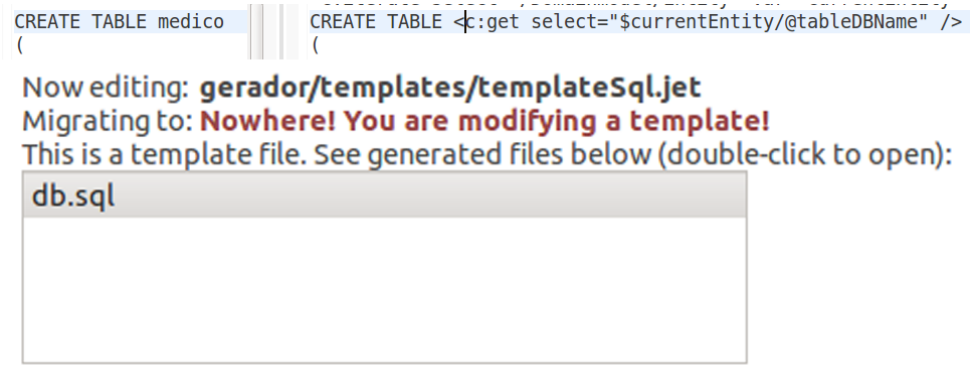


Figura 13: O status "Editando um arquivo de *template*" do *Split JET Editor*.

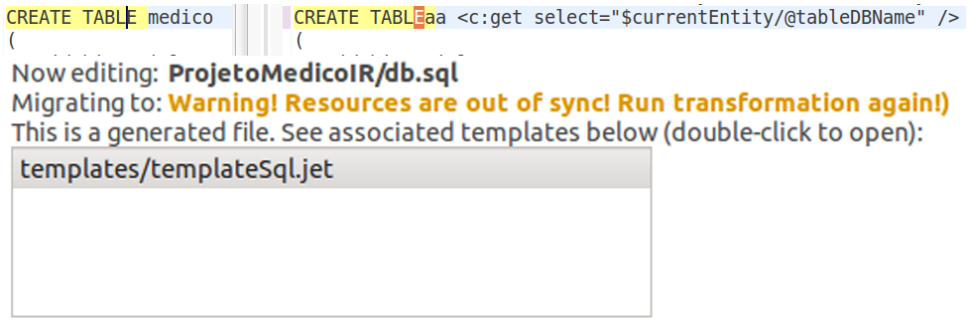


Figura 14: O status "Sincronia perdida" do *Split JET Editor*.

Para a realização da migração automática das alterações realizadas no código-fonte da IR foram utilizadas partes do protótipo desenvolvido por Possatto (2013), sendo que o mesmo é executado a cada tecla pressionada pelo desenvolvedor. Além de migrar automaticamente as alterações, o protótipo desenvolvido por Possatto (2013) atualiza o arquivo de mapeamento modificando apenas as linhas que englobam o trecho de código-fonte alterado. Caso o desenvolvedor precise desfazer uma alteração realizada na IR, o *Split JET Editor* desfaz automaticamente nos *templates* e no arquivo de mapeamento.

Quando a sincronia entre os *templates* e o código-fonte da Implementação de Referência foi perdida ou quando o desenvolvedor altera manualmente um ou mais *templates*, é necessário que o mesmo execute o *JET Transformation* para que a sincronia seja restabelecida. O *JET Transformation* consiste na leitura dos *templates* e a resolução de suas *TAGs* pelo *plug-in JET*, sendo que o JET utilizado foi modificado por Possatto (2013) para que o mesmo gere o arquivo de mapeamento toda vez que for executado.

Com o intuito de comparar seu ambiente com o protótipo de Possatto (2013), Perini (2015) realizou o mesmo experimento que Possatto (2013). Nesse estudo, verificou-se que houve uma melhora em relação aos tempos obtidos por Possatto (2013) em quase todas as tarefas, conforme demonstrado na Tabela 2.

Técnica	Tarefas	Grupo	Média	Soma das Médias	Razão entre Protótipo e Manual
Manual de Possatto	T1	G1	00:14:30	00:51:00	54,90%
	T2	G2	00:07:30		
	T3	G1	00:15:30		
	T4	G2	00:13:30		
Protótipo de Possatto	T1	G2	00:06:45	00:23:00	
	T2	G1	00:07:30		
	T3	G2	**00:03:45**		
	T4	G1	00:05:00		
Manual de Perini	T1	G3	00:09:15	00:33:45	34,81%
	T2	G4	00:06:00		
	T3	G3	00:09:00		
	T4	G4	00:09:30		
Protótipo de Perini	T1	G4	**00:05:15**	00:22:00	
	T2	G3	**00:03:00**		
	T3	G4	00:09:45		
	T4	G3	**00:04:00**		
Total				02:09:45	100,00%

Tabela 2: Resultados obtidos pelo estudo realizado por Possatto (2013) e Perini (2015).

O estudo mostrou que na tarefa T3 não houve ganho com a utilização do protótipo de Perini (PERINI 2015), contudo o mesmo relata que o enunciado da tarefa apresentava um erro e isso dificultou a execução por parte dos participantes. Porém, o erro estava presente tanto no enunciado da T3 manual quanto no enunciado do protótipo. Com isso, esse estudo também demonstra que existem tarefas em que a utilização do ambiente automatizado não traz benefícios e, com isso, um estudo que identifique essas tarefas se faz necessário.

3.2 Revisão Sistemática

3.2.1 Metodologia da Pesquisa

Conforme apresentado anteriormente, há um grande custo no processo de geração de código, principalmente na migração de código. Desta forma, pretende-se encontrar métodos, abordagens ou processos em MDD (*Model-Driven Development*), sendo o enfoque na geração de código e, principalmente, na migração de código.

O objetivo desta revisão é a descrição de métodos para a migração de código, para a geração de código e para MDD e metodologias e processos para MDD que englobem a manutenção de sistemas descritos na literatura

Com o intuito de definir o escopo desta revisão, foram estabelecidos alguns critérios para garantir a viabilidade da execução (custo, esforço e tempo), acessibilidade aos dados

e abrangência do mesmo. O primeiro critério foi considerar apenas as publicações escritas na língua inglesa, pois a mesma é a língua-padrão do meio acadêmico. Além disso, não houve restrição de ano na seleção dos artigos.

Já o segundo critério, foi a seleção das bases de dados, sendo que foram escolhidas as seguintes bases:

- *ACM Digital Library*: <http://dl.acm.org>;
- *Compendex*: <http://www.engineeringvillage.com>;
- *IEEE Xplore*: <http://ieeexplore.ieee.org>;
- *Scopus*: <http://www.scopus.com>;
- *Web of Science*: <http://apps.webofknowledge.com>; e
- Banco de Teses da CAPES: <http://bancodeteses.capes.gov.br/>.

3.2.1.1 Método de Busca

Inicialmente, foi feita uma busca automática, com base em uma expressão lógica, visando levantar um conjunto de estudos primários. Os resultados foram então analisados na íntegra. Em seguida, as listas de referências dos primeiros estudos encontrados foram analisadas em busca de mais estudos, em um processo também conhecido como "*Snowballing*" o qual se repetiu até que se chegasse a um nível de saturação, onde novos estudos não puderam mais ser identificados.

Para se chegar à expressão lógica, também chamada de *string de busca*, adotada nas bases de dados, foi realizado um processo de teste e refinamento da mesma, que seguiu os seguintes passos: Primeiramente, foi estudada a expressão lógica usada por Barreto (2011). Depois disso, foram analisados os artigos já utilizados no desenvolvimento desta dissertação, com o objetivo de se estabelecer as *keywords* mais utilizadas. O terceiro passo, foi a adaptação dos parâmetros adotados por Barreto (2011), que seguem a classificação de população, intervenção, comparação e saída.

- **População**: Pesquisas relacionadas à definição de processos de software e Técnicas de reutilização de processos;
- **Intervenção**: Migração de código, geração de código, MDD, *Framework*, Engenharia Reversa, Engenharia Ida-e-Volta;
- **Comparação**: Não há comparação entre os resultados, pois o objetivo desta pesquisa é a descrição dos métodos; e

- **Saída**: Métodos, Abordagens ou Processos de Engenharia de Software.

Desta forma, as expressões escolhidas e utilizadas foram:

- **População**:

```
("software process" OR "software architecture" OR "software
engineering" OR "reusable software processes" OR "software
reusability" OR "process reengineering" OR "reusing software
processes" OR "process pattern")
```

- **Intervenção**:

```
("model-driven" OR "model software" OR "model driven
engineering" OR "model driven development" OR "model-driven
development" OR "model driven" OR "model driven architecture"
OR "automatic model development" OR "automatic source code
generation" OR "framework" OR "reverse engineering" OR
"round-trip engineering")
```

- **Comparação**: Não aplicável; e
- **Saída**:

```
("method engineering" OR "situational method engineering" OR
"model-driven software development methodology" OR "model
driven architecture methodology" OR "dramatically effective
application development methodology" OR "conceptual tools" OR
"agile methodology")
```

Com isso, a expressão lógica foi adaptada a cada base de dados. Na base de dados *ACM Digital Library*, foi necessária a inclusão do filtro *ACM Computing Classification System* (CCS) com o valor I.6.5 que corresponde ao assunto *Model Development* para que apenas artigos da área do MDD fossem retornados. Desta forma, a expressão utilizada nesta base foi a seguinte:

```
((("software process" OR "software architecture" OR "software
engineering" OR "reusable software processes" OR "software
reusability" OR "process reengineering" OR "reusing software
processes" OR "process pattern") AND ("model-driven" OR "model software"
OR "model driven engineering" OR "model driven development" OR
```

```
"model-driven development" OR "model driven" OR "model driven
architecture" OR "automatic model development" OR "automatic source code
generation" OR "framework" OR "reverse engineering" OR "round-trip
engineering") AND ("method engineering" OR "situational method
engineering" OR "model-driven software development methodology" OR "model
driven architecture methodology" OR "dramatically effective
application development methodology" OR "conceptual tools" OR
"agile methodology")) AND (CCS:I65))
```

Já na base de dados *Compendex*, foram necessários alguns outros ajustes, tais como a inclusão do código relacionado à Computação "(72* wn CL)". Sendo assim, a expressão usada foi:

```
(("software process" OR "software architecture" OR "software
engineering" OR "reusable software processes" OR "software
reusability" OR "process reengineering" OR "reusing software
processes" OR "process pattern") AND ("model-driven" OR "model software"
OR "model driven engineering" OR "model driven development" OR
"model-driven development" OR "model driven" OR "model driven
architecture" OR "automatic model development" OR "automatic source code
generation" OR "framework" OR "reverse engineering" OR "round-trip
engineering") AND ("method engineering" OR "situational method
engineering" OR "model-driven software development methodology" OR "model
driven architecture methodology" OR "dramatically effective
application development methodology" OR "conceptual tools" OR
"agile methodology")) wn KY and ({english} wn LA) and (72* wn CL)
```

Por sua vez, a base de dados *Scopus* também necessitou de alguns ajustes, tais como, a especificação da área da Computação. Abaixo a expressão lógica utilizada na referida base.

```
TITLE-ABS-KEY(("software process" OR "software architecture" OR "software
engineering" OR "reusable software processes" OR "software
reusability" OR "process reengineering" OR "reusing software
processes" OR "process pattern") AND ("model-driven" OR "model software"
OR "model driven engineering" OR "model driven development" OR
"model-driven development" OR "model driven" OR "model driven
architecture" OR "automatic model development" OR "automatic source code
generation" OR "framework" OR "reverse engineering" OR "round-trip
engineering") AND ("method engineering" OR "situational method
```

```
engineering" OR "model-driven software development methodology" OR "model
driven architecture methodology" OR "dramatically effective
application development methodology" OR "conceptual tools" OR
"agile methodology")) AND (LIMIT-TO(SUBJAREA, "COMP"))
```

Enquanto que as bases de dados *IEEE Xplore* e *Web of Science* não necessitaram de adaptação da expressão lógica. Com isso, a expressão utilizada foi a seguinte:

```
(("software process" OR "software architecture" OR "software
engineering" OR "reusable software processes" OR "software
reusability" OR "process reengineering" OR "reusing software
processes" OR "process pattern") AND ("model-driven" OR "model software"
OR "model driven engineering" OR "model driven development" OR
"model-driven development" OR "model driven" OR "model driven
architecture" OR "automatic model development" OR "automatic source code
generation" OR "framework" OR "reverse engineering" OR "round-trip
engineering") AND ("method engineering" OR "situational method
engineering" OR "model-driven software development methodology" OR "model
driven architecture methodology" OR "dramatically effective
application development methodology" OR "conceptual tools" OR
"agile methodology"))
```

Por fim, a base de dados Banco de Teses da CAPES também necessitou de alguns ajustes, tais como, a redução dos termos, devido ao limite de caracteres da pesquisa, e a tradução dos termos para o português brasileiro. Dessa forma, a seguinte expressão foi executada nessa base:

```
(("model-driven" OR "dirigido-a-modelo*" OR "md*") AND
("method" OR "método" OR "methodology" OR "metodologia" OR "process*"))
```

3.2.1.2 Procedimentos de Seleção e Critérios

Os procedimentos de seleção dos artigos foram realizados em quatro etapas:

1ª Etapa - Seleção e Catalogação Preliminar: Esta etapa foi realizada a partir da utilização da expressão lógica nas bases de dados selecionadas. Todas as publicações retornadas pelas bases foram catalogadas para análise posterior e aplicação dos filtros de exclusão.

2ª Etapa - Remoção das publicações duplicadas: Com a utilização de diversas bases de dados, pode ocorrer duplicações das publicações, pois revistas, jornais e

conferências podem indexar suas publicações em diversas bases. Logo, é necessário verificar a existência de duplicações existentes nas publicações encontradas.

3ª Etapa - Seleção das Publicações Relevantes pelo *Abstract*: A seleção das publicações através da utilização de expressões lógicas, não garantem que todas as publicações são relevantes para este contexto, pois a aplicação dos critérios são restritos ao aspecto sintático. Sendo assim, todos os resumos foram lidos e analisados de acordo com os seguintes critérios de inclusão:

- **Critério de Inclusão 1**: Publicações que tratavam da definição de métodos ou afins para a migração de código, geração de código ou MDD;
- **Critério de Inclusão 2**: Publicações escritas em língua inglesa ou português; e
- **Critério de Inclusão 3**: Publicações que tenham seu manuscrito completo disponível para download.

Cada publicação foi selecionada para a próxima etapa apenas se cumprisse com todos os critérios de inclusão descritos. Para diminuir o risco de que uma publicação fosse excluída prematuramente, em caso de dúvida ou não existência de *abstract*, a publicação foi considerada como válida.

4ª Etapa - Seleção das Publicações Relevantes pelo Texto Completo: Apesar de toda a verificação da 2ª etapa, pode ocorrer que algumas publicações não sejam excluídas na referida etapa, portanto, após a leitura completa do texto, foram aplicados todos os critérios de inclusão apresentados na referida etapa.

Para facilitar a organização da revisão sistemática, a seleção dos artigos e a detecção dos artigos duplicados, foi utilizada a ferramenta StArt, que é uma ferramenta de apoio a revisões sistemáticas e mapeamentos sistemáticos. Essa ferramenta segue as etapas de seleções descritas anteriormente e detecta automaticamente os artigos duplicados. Além disso, a ferramente permite a importação dos resultados das bases de dados e, também, permite a classificação da prioridade de leitura dos artigos (ZAMBONI et al. 2010).

3.2.2 Resultados da Pesquisa

Após a definição de todo os critérios da Pesquisa, este estudo foi executado entre maio e dezembro de 2014 com a utilização da expressão lógica apresentada e executada nas bases de dados *ACM Digital Library*, *Compendex*, *IEEE Xplore*, *Scopus*, *Web of Science* e o Banco de Teses da CAPES. Inicialmente, foram retornadas 517 publicações, sendo 262 publicações encontradas na *ACM Digital Library*, 63 na *Compendex*, 34 na *IEEE Xplore*, 120 na *Scopus*, 29 *Web of Science* e nove no Banco de Teses da CAPES.

Após a execução da 2ª etapa, que compreende a remoção das duplicações existentes, restaram 403 publicações, pois 114 publicações foram removidas, devido à duplicação. Com a execução da 3ª etapa dos procedimentos de seleção, que envolvia a leitura do título e do *abstract* de todas as publicações, seguindo os critérios de inclusão, foram selecionadas 174 publicações, sendo 82 publicações encontradas na *ACM Digital Library*, 38 na *Compendex*, 17 na *IEEE Xplore*, 18 na *Scopus*, sete *Web of Science*, seis no Banco de Teses da CAPES e seis foram adicionadas manualmente, pois foram encontradas no processo de "*Snowballing*", contudo não estão indexados nas bases consultadas.

Finalizando a 4ª etapa de seleção, que envolvia a leitura completa de todas as publicações, seguindo os critérios de inclusão, foram selecionadas 22 publicações, de modo que as publicações P1 até P14 são metodologias e processos para MDD (Seção 3.2.2.1), já as publicações P15 até P20 são métodos baseados em MDD (Seção 3.2.2.2) e, por fim, as publicações P21 e P22 são processos de apoio ao desenvolvimento utilizando MDD (Seção 3.2.2.3). A Tabela 3 resume os achados desta revisão.

3.2.2.1 Metodologias e Processos para MDD

Gervais (2002) (P1) criou uma metodologia chamada ODAC que é baseada no padrão *Open Distributed Processing* (ODP). O ODP fornece uma série de conceitos e suas regras de estruturação, sendo que o principal conceito é o *Viewpoint* que permite modelar atividades. Um *viewpoint* é uma subdivisão das especificações de um sistema complexo, correspondendo a uma perspectiva particular, que permite que o sistema seja "visto" de um ângulo particular com enfoque em conceitos específicos. Existem 5 *viewpoints* que são *Enterprise*, *Information*, *Computational*, *Engineering* e *Technology*.

Os conceitos definidos na metodologia ODAC são os mesmo definidos no ODP que são utilizados para definir os passos da ODAC, identificando um associação entre as atividades de análise, design e implementação com os *viewpoints* do ODP, sendo representado na Figura 15 com isso os *viewpoints Enterprise*, *Information* e *Computational* são associados a atividade de análise, já o *viewpoint* de *Engineering* é associado com a atividade de Design e, por fim, o *viewpoint* de *Technology* é associado com a atividade de Implementação (GERVAIS 2002).

Na metodologia ODAC há três categorias de especificação: Comportamento do Sistema, Operacional e Engenharia do Sistema, sendo que a relação entre elas está representada na Figura 16. A primeira, especificação do comportamento do Sistema, é a saída da atividade de Análise que corresponde as especificações estabelecidas nos *viewpoints Enterprise*, *Information* e *Computational*, sendo que descreve o sistema de acordo com o seu objetivo e sua posição no processo organizacional do cliente, com essa especificação é criado um modelo PIM (GERVAIS 2002).

A segunda, especificação da engenharia do sistema, é estabelecida no *viewpoint* de

ID	Autores	Tipo	Nome	Engloba Manutenção / Evolução?
P1	Gervais (2002)	Metodologia	ODAC	Não
P2 e P3	Gavras *et al.* (2004a)	Metodologia	MODA-TEL	Não
P4	Larrucea, Diez e Mansell (2004)	Metodologia	MASTER	Sim, porém sem especificação da atividade.
P5	Chitforoush, Yazdandoost e Ramsin (2007)	*Framework*		Não
P6	Asadi, Esfahani e Ramsin (2010)	Metodologia	MDASP	Não
P7	Alegria *et al.* (2011)	Processo		Não
P8	Ruiz, Ramón e Molina (2013)	Processo		Não
P9	Hildenbrand e Korthaus (2004)	Processo	C^3	Não
P10	Kulkarni, Barat e Ramteerthkar (2011a)	Metodologia		Sim, porém sem especificação da atividade.
P11	Kim *et al.* (2005)	Metodologia	DREAM	Não
P12	Hebig e Bendraou (2014)	Revisão Sistemática		Não
P13	Asadi, Ravakhah, Ramsin (2008)	Metodologia	MDA-SDLC	Sim, porém sem especificação da atividade.
P14	Karagiannis, Hrgovcic e Woitsch (2011)	Processo		Não
P15	Céret, Calvary e Dupuy-Chessa (2011)	Método	UsiXML	Não
P16	Guelfi *et al.* (2004)	Método	DRIP Catalyst	Não
P17	Garro e Tundis (2012) (P17)	Método		Não
P18	Júnior (2012)	Método	MDWA	Não
P19	Soares (2012)	Método	PM-MDA	Não
P20	Agner (2012)	Método	PI-MT	Não
P21	Schramm *et al.* (2010)	Processo		Não
P22	Cirilo (2011)	Processo	Model Driven RichUbi	Não

Tabela 3: Resumo das publicações encontradas na revisão sistemática.

Engineering que descreve o ambiente que o sistema será executado, gerando o modelo *Platform-Description Model* (PDM), sendo que para cada tipo de ambiente que o sistema

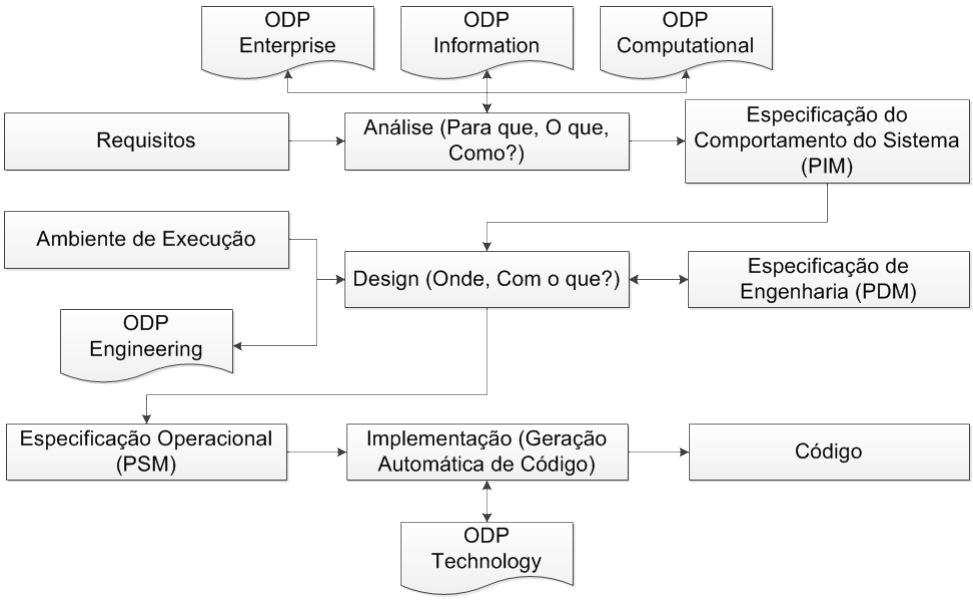


Figura 15: Proposta da metodologia ODAC (GERVAIS 2002).

for executado, a metodologia ODAC fornece diretrizes para auxiliar na descrição do novo ambiente (GERVAIS 2002).

Ao final, a terceira, especificação operacional, é resultado da especificação do comportamento do sistema (modelo PIM) com a especificação da engenharia do sistema (modelo PDM) para determinado ambiente de execução, gerando, dessa forma, o modelo PSM específico para aquele ambiente (GERVAIS 2002).

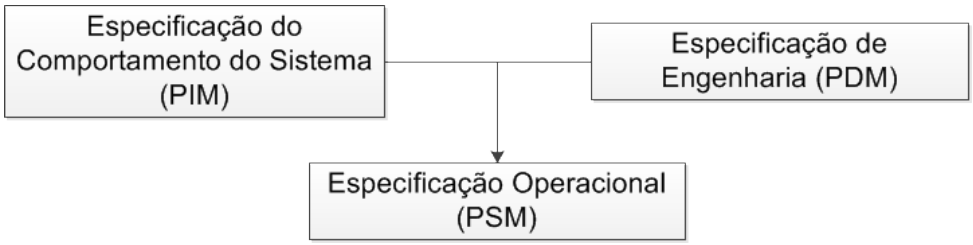


Figura 16: Relação entre as categorias de especificação da metodologia ODAC (GERVAIS 2002).

Gavras *et al.* (2004a) (P2) propõem a metodologia MODA-TEL. Nessa metodologia, no inicio do projeto, é realizada a classificação da equipe de desenvolvimento em três categorias: Construtores do Conhecimento, Facilitadores do Conhecimento e Usuários do Conhecimento. Com a utilização dessa categorização, é possível determinar as diferentes regras e habilidades necessárias de cada categoria, sendo que cada uma dessas regras são diferentes atividades e necessitam de diferentes ferramentas.

A categoria Construtores do Conhecimento engloba as pessoas que constroem o conhecimento (repositórios) para serem utilizados em vários projetos MDA. Essa categoria

inclui os Arquitetos de Software, Experts da Plataforma, Engenheiros de Controle de Qualidade e Experts em Metodologias (GAVRAS et al. 2004a).

Na categoria Facilitadores do Conhecimento estão as pessoas que montam, combinam, customizam e implantam o conhecimento específico para cada projeto MDA. Nessa categoria são incluídos os Gerentes de Projetos e Engenheiros de Controle de Qualidade (GAVRAS et al. 2004a).

Por fim, na categoria Usuários do Conhecimento são as pessoas que aplicam o conhecimento nos projetos MDA. São incluídos nessa categoria os Designers e Engenheiros de Software (GAVRAS et al. 2004a).

Outro ponto abordado por essa metodologia é a distinção entre as atividades de preparação e execução. As atividades de preparação são os esforços para a estruturação e planejamento do projeto, onde é possível à reutilização de conhecimento, que é um dos principais benefícios do MDD. Esse tipo de atividade é principalmente executada pelos Construtores de Conhecimento e elas devem ser realizadas antes das atividades de execução (GAVRAS et al. 2004a).

Contudo, nessa metodologia é possível a intercalação entre os dois tipos de atividades, permitindo, dessa forma, que as atividades de preparação sejam revisadas enquanto as atividades de execução são realizadas. Isso é necessário devido às possíveis mudanças dos requisitos dos projetos (por exemplo, a mudança de plataforma), ao melhor detalhamento de um requisito (por exemplo, alguns requisitos não foram suficientemente detalhados), aos problemas que podem ocorrer na fase de execução (por exemplo, a limitação ou falta de expressividade da linguagem de modelagem escolhida) entre outros. Sendo assim, a metodologia MODA-TEL consiste nas seguintes cinco fases e está representada na Figura 17 (GAVRAS et al. 2004a).

A primeira fase é a de Gerenciamento do Projeto que tem como objetivo organizar e monitorar o projeto (Figura 17-1), sendo executada pelos Facilitadores do Conhecimento. Nessa fase há a realização das atividades de Seleção do Processo de Desenvolvimento de Software, Organização do Projeto e Gerenciamento da Qualidade (GAVRAS et al. 2004a).

A segunda fase é a de Preparação Preliminar que tem como o objetivo de identificar as necessidades da modelagem e das transformações (17-2), sendo que normalmente é executada pelos Construtores de Conhecimento. Nessa fase há a realização das atividades de Identificação da Plataforma, Identificação da Linguagem de Modelagem, Identificação das Transformações e Definição das Estratégias de Propagação (GAVRAS et al. 2004a).

A fase 3 é a fase de Preparação Detalhada, que tem como objetivo determinar as especificações da modelagem e das transformações (17-3) e é realizada pelos Construtores de Conhecimento. Nessa fase há a realização das atividades de Especificação das Linguagens de Modelagem e Especificação das Transformações (GAVRAS et al. 2004a).

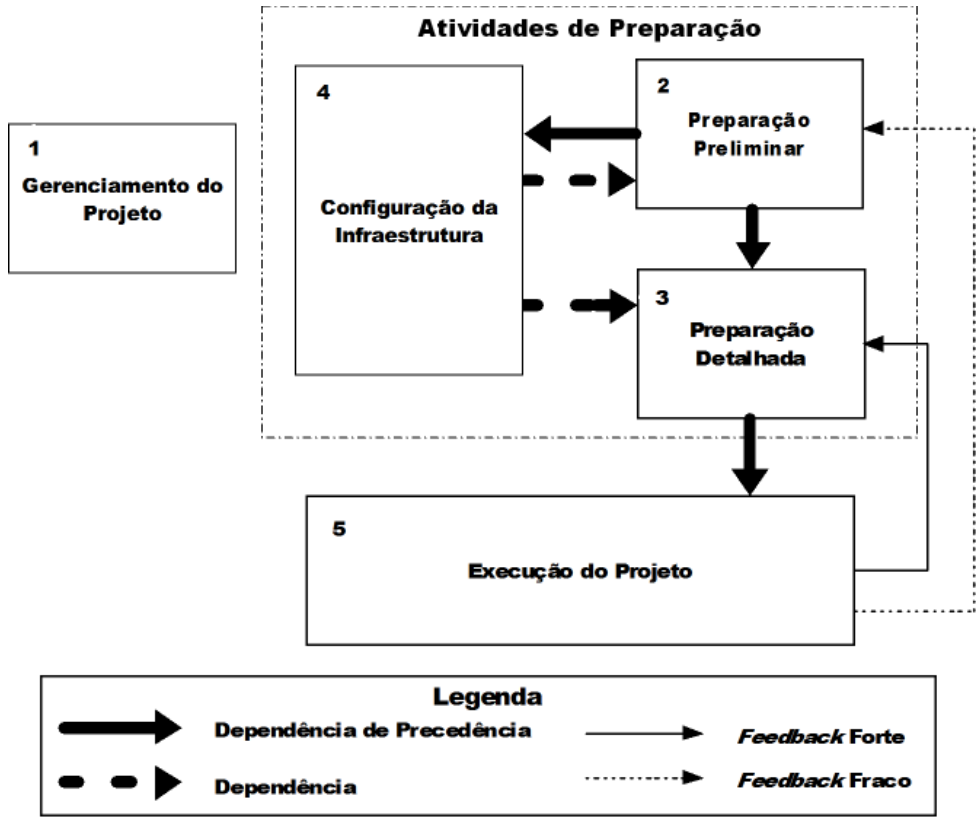


Figura 17: Proposta da metodologia MODA-TEL (GAVRAS et al. 2004a).

A Configuração da Infraestrutura é a fase 4, nela há a criação de ferramentas de suporte e gerenciamento dos metadados para a facilitação do uso (17-4) e é executada pelos Construtores de Conhecimento. Nessa fase há a realização das atividades de Seleção de Ferramentas e Gerenciamento de Metadados (GAVRAS et al. 2004a).

Por fim, a fase 5 é a de Execução do Projeto, sendo que nessa fase é produzido os artefatos necessários e o produto final (17-5), sendo realizada pelos Usuários do Conhecimento. Nessa fase há a realização das atividades de Análise de Requisitos, Modelagem, Verificação/Validação, Transformações, Codificação/Testagem, Integração/Implantação e Operação/Manutenção (GAVRAS et al. 2004a).

Gavras *et al.* (2004b) (P3) descrevem um estudo de caso da aplicação do MODA-TEL no desenvolvimento de uma aplicação baseada em VoiceXML. Nesse estudo era esperado que todo o código-fonte da aplicação fosse gerado automaticamente e seu objetivo foi alcançado. Além disso, o autor relata que os processos, metodologias e métodos baseados em MDD devem ser extremamente bem descritos e bem documentados, no que diz respeito aos produtos, definições e atividades necessárias no desenvolvimento.

Larrucea, Diez e Mansell (2004) (P4) descrevem uma metodologia ágil para MDD, sendo baseado na definição de diferentes camadas de arquiteturas e modelo como domínio. A Figura 18 apresenta as oito fases constituintes dessa metodologia.

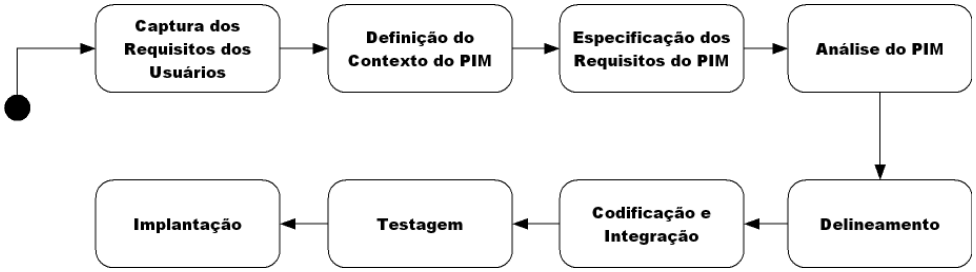


Figura 18: Proposta da metodologia MASTER (LARRUCEA, DIEZ e MANSELL 2004).

A primeira fase dessa metodologia é chamada de Captura dos Requisitos dos Usuários. Seu objetivo é extrair, combinar e documentar os requisitos do cliente que são necessários para o funcionamento adequado do sistema. Nessa fase está incluso o estabelecimento junto ao cliente um consentimento dos requisitos funcionais e não-funcionais. As atividades pertencentes a essa fase são: Formalizar os Requisitos do Cliente em um Modelo de Aplicação, Derivar um PIM de Aplicação Inicial e Especificação Inicial dos Requisitos Funcionais da Infraestrutura em Comum com outros projetos (LARRUCEA, DIEZ e MANSELL 2004).

A segunda fase é chamada de Definição do Contexto do PIM, sendo seu objetivo definir o escopo do sistema. Os resultados são uma definição não-ambígua do sistema, seus objetivos e seu escopo, sendo aplicada a abordagem de caixa-preta. As atividades principais são: Estabelecer os objetivos do sistema e os princípios de negócios, Descrever os atores externos que interagem com o sistema, Identificar os serviços de alto-nível oferecidos pelo sistema e seus comportamento-chave e Definir os eventos e os objetos de negócios (LARRUCEA, DIEZ e MANSELL 2004).

A Especificação dos Requisitos do PIM é a terceira fase, com o objetivo da construção de um modelo de requisitos do cliente claro e completo e ter a descrição de requisitos únicos nos modelos subsequentes. Com o intuito de organizar os requisitos em funcionais e não-funcionais, sendo as atividades principais dessa fase: Refinar o Contexto do PIM, Identificar serviços, eventos e objetos de negócio produzidos e consumidos pelo sistema e os atores que interagem com o mesmo, Especificar capacidades (casos de uso), forças (requisitos não-funcionais) e requisitos atômicos e Identificar e modelar os relacionamentos entre os requisitos funcionais e não-funcionais (LARRUCEA, DIEZ e MANSELL 2004).

Análise do PIM é a quarta fase e tem como objetivo a modelagem das funcionalidades do sistema, sem nenhuma interferência das tecnologias de implementação, mantendo a separação dos requisitos em funcionais e não-funcionais. As atividades principais dessa fase são três. A primeira é descrever as funcionalidades do sistema, definindo os objetos (com as classes, atributos, pacotes, etc), funções (com as operações), interfaces, comportamentos (com os diagramas de sequência), entre outros. Já a segunda fase é descrever os

aspectos de qualidade do sistema (refinamento das classes) e sua aplicação nos elementos funcionais do modelo e, por fim, a atividade de manutenção da rastreabilidade dos requisitos do PIM (LARRUCEA, DIEZ e MANSELL 2004).

A atividade de manutenção da rastreabilidade dos requisitos do PIM consiste na definição de regras das transformações que permite que quando houver uma alteração em um modelo de outro nível, essas alterações possam ser migradas para o modelo PIM. Contudo, os autores não definem um método para a realização dessa migração (LARRUCEA, DIEZ e MANSELL 2004).

Por sua vez, a quinta fase é chamada de Delineamento e seu objetivo é modelar os detalhes estruturais e comportamentos do sistema que cumprirão os requisitos funcionais e não-funcionais do sistema. Isso implica na realização de decisões de como o sistema será implementado e quais arquiteturas, padrões e plataformas serão utilizadas. Seguindo uma abordagem MDA, essa fase é executada em dois passos: Especificar e delinear o PIM para todos os requisitos. O PIM será definido com diferentes elementos dependendo da arquitetura escolhida e Especificar e delinear o PSM pelo refinamento do PIM. O PSM é derivado automaticamente do PIM, através das transformações. O PSM contém os modelos específicos da plataforma e é detalhado e completo o suficiente para permitir a codificação e o desenvolvimento da solução (LARRUCEA, DIEZ e MANSELL 2004).

A Codificação e a Integração constituem a sexta fase, tendo como objetivo o desenvolvimento e verificação do código-fonte do sistema, que implementa os requisitos. Essa fase inclui atividades como Desenvolvimento dos Componentes e das Classes (baseado nos modelos utilizados como entrada), Definição da Organização do Código-Fonte, Execução de Testes Unitários e Integração de Componentes e Subsistemas. De acordo com a abordagem MDA, o código-fonte é automaticamente gerado pelos transformadores, que se baseiam no PSM (LARRUCEA, DIEZ e MANSELL 2004).

A Testagem é a sétima fase, na qual o objetivo é demonstrar que a versão final do sistema cumpre os requisitos. Essa fase inclui as atividades de Planejamento dos Testes, Preparação dos Modelos de Teste, Scripts e Casos de Teste, Execução dos Testes, Correção dos Defeitos e Documentação dos Resultados dos Testes. Testes de Modelos são rastreáveis até o PIM e, segundo a abordagem MDA, os Testes de Modelos são refinados a partir do PIM e os scripts e casos de testes são gerados automaticamente a partir dos Testes de Modelos (LARRUCEA, DIEZ e MANSELL 2004).

Por fim, a oitava e última fase é chamada de Implantação, com o objetivo de assegurar o sucesso da transição entre o ambiente de desenvolvimento para o de produção. Essa fase inclui as atividades de Criação do Planejamento de Implantação (datas de instalação, recursos, etc.), Criação do Modelo de Implantação, Criação dos Manuais do Sistema, Manutenção dos Registros iniciais do Sistema e Fornecimento da Instalação para o Cliente (LARRUCEA, DIEZ e MANSELL 2004).

Chitforoush, Yazdandoost e Ramsin (2007) (P5) propõem um *framework* de metodologia de desenvolvimento de sistemas para MDD que é composto por 4 fases, sendo que cada uma fase consiste em várias etapas e suas atividades, e está representado na Figura 19. Além disso, esse *framework* foi criado com a possibilidade de reutilizar o processo ou metodologia já existente na organização ou ser utilizado como uma metodologia de desenvolvimento.

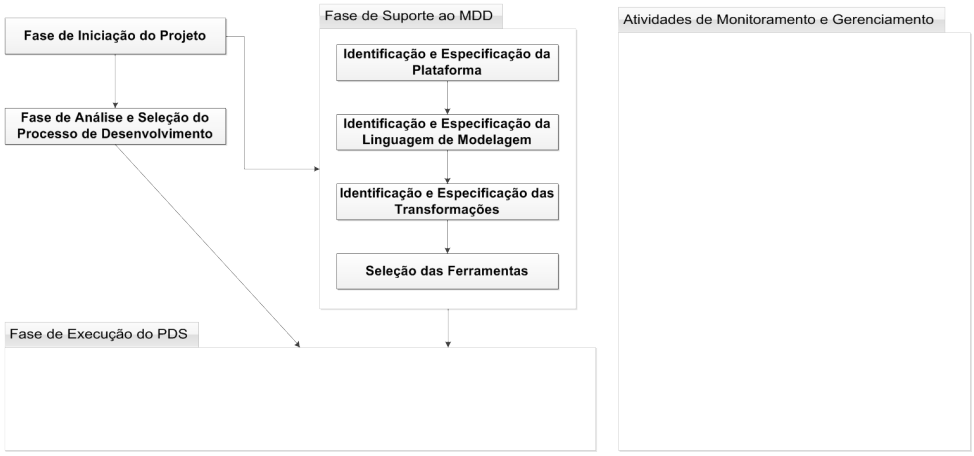


Figura 19: Proposta de um *framework* de suporte ao desenvolvimento de sistemas utilizando MDD (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

A primeira fase do *framework* é a Iniciação do Projeto, cuja função é a identificação dos objetivos do projeto e a estimação do tamanho e do escopo. Todas as variáveis e riscos do projeto são analisados e as pessoas importantes, organizações e sistemas externos que irão interagir com o sistema são identificados. Além disso, a equipe é determinada (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

Já a segunda fase é a de Análise e Seleção do Processo de Desenvolvimento que será utilizado no projeto. Os requisitos necessários do processo de desenvolvimento de sistemas são analisados nessa fase. Os requisitos especificados são utilizados como base para a seleção de um Processo de Desenvolvimento de Sistemas (PDS) que serão usados na fase de Execução do PDS. Se um PDS já existente for selecionado, é necessária a adaptação de seus requisitos para os do projeto, uma vez que todas as modificações necessárias são realizadas nessa fase e na próxima, tendo como resultado a produção precisa da descrição do processo para cada futura atividade desse projeto (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

A terceira fase é a de Suporte ao MDD, a qual pode ser executada em paralelo com a fase de Análise e Seleção do Processo de Desenvolvimento. Nessa fase são aplicadas as decisões sobre o MDD e sua arquitetura. Essa fase inclui as seguintes atividades (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007):

- **Identificação e Especificação da Plataforma**: A plataforma de produção que o sistema será implementado é identificado por um Arquiteto de Sistemas. As características da plataforma incluem todos os aspectos de hardware, software e tecnologia, visto que essa identificação deve ser especificada em um Modelo de Plataforma (MP). Esse modelo será usado na transformação do modelo PIM para o modelo PSM;

- **Identificação e Especificação da Linguagem de Modelagem**: Nessa atividade, são identificadas as diferentes perspectivas que o sistema deverá ser modelado, sendo assim, a linguagem de modelagem que será utilizada deve suportar todas as perspectivas necessárias. Além disso, essa linguagem será utilizada para a criação dos modelos PIM e PSM;

- **Identificação e Especificação das Transformações**: As transformações entre modelos necessários ou possíveis são identificadas, pois o principal foco das transformações é a transformação dos modelos PIM para modelos PSM, mas pode ser necessária a transformação entre modelos PIM ou modelos PSM, que devem ser planejadas nessa atividade. As transformações devem considerar as particularidades das linguagens de modelagem selecionadas. Após a decisão de quais transformações são necessárias, elas devem ser especificadas em detalhes, contendo as regras e características das transformações bem descritas;

- **Seleção das Ferramentas**: As ferramentas são fundamentais em um ambiente de desenvolvimento MDD. O número de atividades deve ser controlado por ferramentas, tanto como as especificações das linguagens de modelagem, os modelos, as transformações de modelos, geração de código e a definição das regras e características das transformações. Nessa atividade, as ferramentas necessárias são selecionadas para serem utilizadas na próxima fase.

A última fase é a Execução do PDS, que é a principal fase do projeto, pois todos os produtos do projeto são desenvolvidos. As atividades específicas dessa fase dependem da metodologia/processo selecionado. Como resultado da natureza MDD desse *framework*, os modelos são os principais produtos do processo de desenvolvimento e são refinados, de forma interativa e incremental, do modelo PIM até o código-fonte final. Como os modelos guiam toda a fase de execução, em caso de erros, falhas, defeitos ou outros problemas detectados em qualquer atividade dessa fase, o processo deve facilitar a resolução desse problema ao nível de modelagem. Todas as atividades dessa fase podem gerar *feedback* que será utilizado para refinar e melhorar o processo de desenvolvimento e podem influenciar os resultados das fases anteriores. Com isso, é possível enviar *feedback* para a fase de Suporte ao MDD, caso haja alguma mudança a ser feita na linguagem de modelagem ou no MP (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

Além das fases descritas, o *framework* proposto contém as atividades de Monitoramento e Gerenciamento que são executadas durante todo o ciclo de desenvolvimento do projeto. Essas atividades incluem o Gerenciamento do Projeto, Controle de Qualidade, Gerenciamento de Riscos, Treinamento e Gerenciamento de Reuso. Essa última é a mais importante atividade dessa categoria, pois irá gerenciar a reutilização dos modelos, as linguagens de modelagens, as especificações das transformações de modelos (CHITFOROUSH, YAZDANDOOST e RAMSIN 2007).

No estudo de Asadi, Esfahani e Ramsin (2010) (P6) é proposta a metodologia *MDA Software Process* (MDASP) baseada em padrões de processo e inclui três fases. Cada uma contém várias etapas, conforme demonstrado na Figura 20.

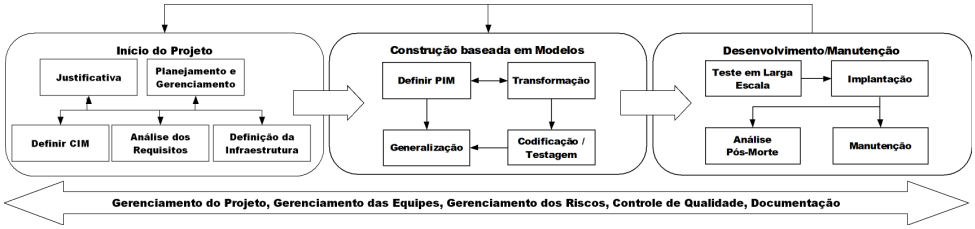


Figura 20: Proposta da metodologia MDA Software Process (MDASP) (ASADI, ESFAHANI e RAMSIN 2010).

A primeira fase da metodologia MDASP é chamada de Início de Projeto. O objetivo dessa fase é fornecer os fundamentos para o desenvolvimento do software. Esse fornece a justificativa do projeto e produz os modelos de requisitos, o planejamento da infraestrutura necessária, o planejamento inicial e o gerenciamento da documentação. As etapas constituintes dessa fase são: Justificativa, Definir CIM, Análise dos Requisitos, Planejamento e Gerenciamento e Definição da Infraestrutura (ASADI, ESFAHANI e RAMSIN 2010).

A segunda fase, chamada de Construção Baseada em Modelos, produz o software de acordo com a abordagem MDD. Um modelo PIM completo e preciso é criado a partir dos modelos de requisitos, representando a estrutura e o comportamento do sistema. O modelo PIM é transformado em um modelo PSM e, por fim, em código-fonte. Nessa fase, as etapas constituintes são: Definir PIM, Transformação, Generalização e Codificação/Testagem (ASADI, ESFAHANI e RAMSIN 2010).

A terceira fase é chamada de Desenvolvimento/Manutenção, tendo como objetivo o sucesso da entrega do sistema para o cliente e a manutenção desse sistema em produção. Isto é realizado através dos testes da aplicação em larga escala e sua constante manutenção. As etapas constituintes dessa fase são: Teste em Larga Escala, Implantação, Manutenção e Análise Pós-Morte (ASADI, ESFAHANI e RAMSIN 2010).

Alegria *et al.* (2011) (P7) descrevem uma abordagem automática de adaptação dos

processos de Engenharia de Software para o contexto particular de cada projeto, sendo baseado nas técnicas do MDE. As adaptações são implementadas por transformações de modelos que tem como entrada o modelo do processo organizacional, incluindo suas variâncias, e o modelo do contexto do projeto. O resultado dessa transformação é um processo adaptado ao contexto, conforme exemplificado na Figura 21. As formalizações dos metamodelos e a implementação das regras de transformações são realizadas utilizando a linguagem ATL (*Atlas Transformation Language*).

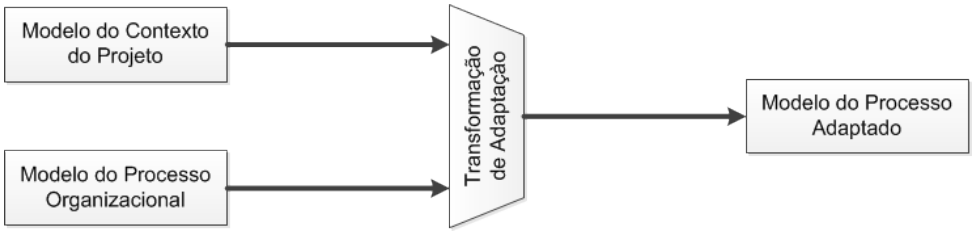


Figura 21: Proposta de abordagem automática de adaptação dos processos de ES para o contexto particular de cada projeto (ALEGRIA et al. 2011).

No Modelo do Contexto do Projeto são incluídas diversas características, tais como, conhecimento sobre o domínio da aplicação (alto, médio ou baixo), o tipo do projeto (desenvolvimento, extensão ou reengenharia) e seu tamanho (pequeno, médio ou grande) (ALEGRIA et al. 2011).

O Modelo do Processo Organizacional representa o processo utilizado na organização e suas variações já existentes. O resultado da transformação desses dois modelos é o Modelo do Processo Adaptado, que é um processo específico para o projeto em questão, incluindo todas as variações possíveis existentes no projeto (ALEGRIA et al. 2011).

Em seu estudo, Alegria *et al.* (2011), relata, através de um estudo de caso, a aplicação de sua abordagem em um projeto de uma empresa chilena de médio porte, sendo que essa abordagem obteve sucesso na sua aplicação, mostrando-se uma abordagem efetiva, prática e usual.

Ruiz, Ramón e Molina (2013) (P8) descrevem um processo para a migração de sistemas baseada na abordagem MDD, representado na Figura 22.

Esse processo tem como principal artefato o Modelo de Migração, que define o processo de migração como um todo. Nesse modelo está contido uma série de tarefas que devem ser cumpridas ao longo da realização da migração, descrevendo, também, quais ferramentas e artefatos serão utilizados. O Modelo de Migração é criado apenas uma vez e é reutilizado a cada nova migração. As mudanças num processo são raramente esperadas, mas caso isso ocorra, esse modelo pode ser facilmente modificado para os novos requisitos. Esse modelo não pode ser definido automaticamente, mas deve ser definido por um desenvolvedor especialista em migração utilizando uma *Domain-Specific Language*

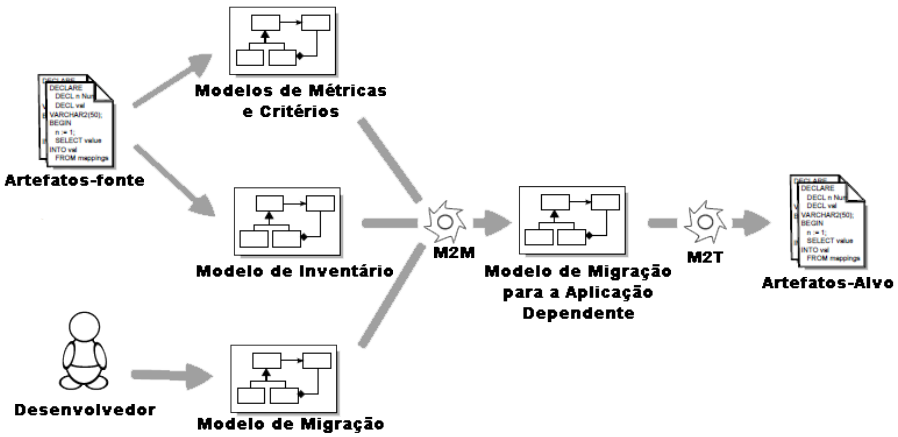


Figura 22: Proposta de processo para a Migração de Sistemas baseada em MDD (RUIZ, RAMÓN e MOLINA 2013).

(DSL) (RUIZ, RAMÓN e MOLINA 2013).

Além do Modelo de Migração, os modelos de Métricas e Critérios e de Inventário são necessários para a geração do Modelo de Migração para a Aplicação Dependente. O Modelo de Métricas e Critérios contém algumas informações sobre o sistema-fonte, o qual pode ser usado como um guia para a geração do modelo final, além de estabelecer e ordenar as tarefas necessárias para o cumprimento desse processo. Como por exemplo, a complexidade dos arquivos do código-fonte pode ser mensurado por *Lines of Code* (LOCs) e serem ordenados de forma crescente. Esse modelo pode ser gerado de forma automático por algumas ferramentas que analisam os artefatos-fonte (RUIZ, RAMÓN e MOLINA 2013).

O Modelo de Inventário é um conjunto de recursos necessários para a realização do processo de migração, como, por exemplo, o código-fonte e as conexões com o Banco de Dados. Para cada arquivo, são armazenadas as informações necessárias para a migração, tais como o caminho para o arquivo, o tipo do arquivo, seu tamanho e a quantidade de linhas (RUIZ, RAMÓN e MOLINA 2013).

Os três modelos (Modelo de Métricas e Critérios, Modelo de Inventário e Modelo de Migração) são transformados no Modelo de Migração para a Aplicação Dependente, que representa um processo de migração específico para uma determinada aplicação-fonte. Esse modelo contém detalhes suficientes para se realizar a migração do sistema-fonte, desde quais artefatos devem ser migrados manualmente e quais serão migrados por ferramentas. Por fim, os artefatos-alvos são gerados a partir do Modelo de Migração para a Aplicação Dependente, podendo ser realizado de forma automática, semi-automática ou manual (RUIZ, RAMÓN e MOLINA 2013).

Hildenbrand e Korthaus (2004) (P9) propõe o processo C^3 para a aplicação dos

conceitos do MDD no desenvolvimento de sistemas de negócios. Os autores também adotam vários conceitos da metodologia *Business Object Oriented Software Technology for Enterprise Reengineering* (BOOSTER), sendo que essa metodologia é baseada no desenvolvimento de sistemas que utilizam componentes. Juntamente com a aplicação das técnicas e padrões do MDD, os autores adotaram três "Cs" como pilares de seu processo: Concorrência, Colaboração e Componentes.

A Figura 23 apresenta o ambiente de desenvolvimento proposto por Hildenbrand e Korthaus (2004), tendo como a principal *feature* desse ambiente o repositório de domínio que contém os metadados específicos do domínio e os componentes desenvolvidos para esse domínio pelos engenheiros de negócios e os desenvolvedores. O processo C^3 é constituído de 3 partes principais: Situação-problema (contexto do processo), Definição dos papéis e Avaliação e Métricas do Processo.

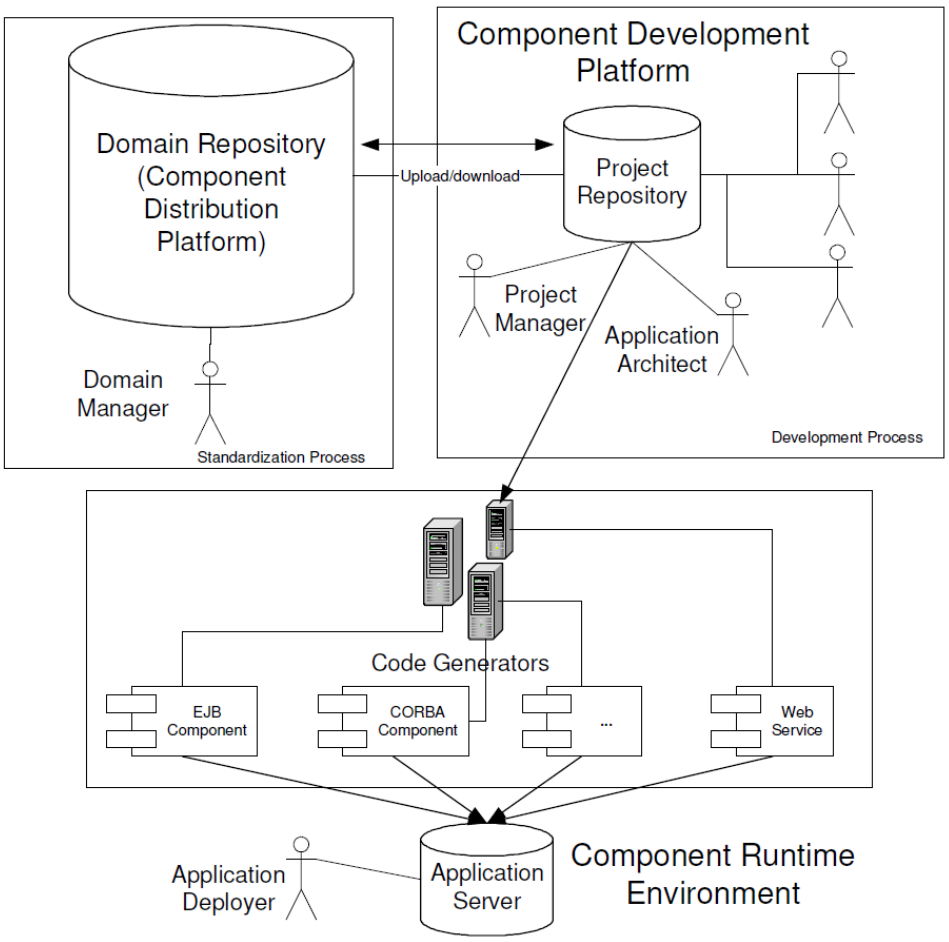


Figura 23: Ambiente Colaborativo de Desenvolvimento do processo C^3 (HILDENBRAND e KORTHAUS 2004).

A primeira parte é a Situação-problema, que consiste no levantamento dos requisitos e especificação dos casos de usos para os domínios que devem ser suportados pelo sistema. Já na parte de Definição dos papéis é onde cada participante do desenvolvimento

recebe seu papel no processo, conforme apresentado na Figura 23, há os seguintes papéis (HILDENBRAND e KORTHAUS 2004):

- ***Domain Manager***: É o responsável pelo gerenciamento das informações dos domínios, sendo sua responsabilidade o armazenamento, arquivamento e publicação dos componentes e padrões no repositório do domínio;
- ***Project Manager***: É o responsável pela integração no sistema, pois é de sua responsabilidade a verificação da consistência entre os produtos de cada time de desenvolvimento;
- ***Application Architect***: É o responsável por transformar os requisitos e as análises em estruturas básicas para a integração com os componentes;
- ***Component Developer***: É a função de apenas um desenvolvedor que deve elaborar mais sobre a especificação semântica de um componente no contexto de uma certa arquitetura e suas dependências em relação a outros componentes; e
- ***Application Deployer***: Sua função é implantar os componentes gerados para uma plataforma específica.

Por fim, na parte de Avaliação e Métricas do Processo são definidas três métricas para a avaliação da instância utilizada do processo de desenvolvimento, sendo elas: o Nível de reutilização de componentes, o Compartilhamento de código gerado automaticamente e o Tempo de desenvolvimento. Essas métricas devem ser implementadas por cada Engenheiro de Software, pois não há mínimo definido no artigo de Hildenbrand e Korthaus (2004).

O processo C^3 é composto por cinco fases de desenvolvimento: Padronização, Desenvolvimento do Sistema, Design dos Modelos, Geração de Código e Implantação do Sistema. A primeira fase é a de Padronização, na qual se deve escolher quais repositórios de domínio serão utilizados. Esse repositório fornece uma interface para pesquisa sobre os projetos anteriores que englobam os domínios em questão. Depois disso, é possível adicioná-los ao novo projeto de sistema e as novas especificações e padrões podem ser adicionados ao repositório (HILDENBRAND e KORTHAUS 2004).

A segunda fase é a do Desenvolvimento do Sistema onde as tarefas de modelagem dos novos componentes devem ser divididas entre as equipes, sendo que esses modelos devem ser independentes de plataforma, ou seja, devem ser modelos do tipo PIM. Além disso, são definidas as arquiteturas principais do sistema e os testes de integração dos componentes. A terceira fase é a fase de Design dos Modelos, na qual os componentes são integrados e sua integração é verificada (HILDENBRAND e KORTHAUS 2004).

Na quarta fase, ocorre a Geração de Código, que a partir dos modelos de componentes, gera-se o código-fonte da aplicação para cada plataforma necessária, sendo que essa geração de código pode ser automatizada com a utilização de ferramentas como o Enterprise Java Beans ou CORBA. Além disso, caso haja necessidade de alteração de código manualmente, ela deve ser realizada nessa fase. Por fim, na última fase, a de Implantação do Sistema, os testes são realizados e a aplicação é implantada (HILDENBRAND e KORTHAUS 2004).

Kulkarni, Barat e Ramteerthkar (2011a) (P10) descrevem a necessidade de adoção de uma metodologia mais dinâmica para a utilização de seu conjunto de ferramentas MDD, o MasterCraft (KULKARNI, BARAT e RAMTEERTHKAR 2011b). O MasterCraft é um *plug-in* para a IDE Eclipse que engloba cinco ferramentas principais. A primeira ferramenta é para a modelagem do metamodelo, com o intuito de especificar uma *view* abstrata. Já a segunda ferramenta é um conjunto de modeladores para popular a instância da *view* abstrata. A terceira é um conjunto de geradores de código que transformam cada instância da *view* no artefato desejado. A quarta é um conjunto de utilitários para compilação automatizada. A quinta é um repositório para o desenvolvimento de componentes (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

Apesar da simplicidade na utilização do MasterCraft, houve um grande aumento na demanda dos usuários dos sistemas desenvolvidos por suporte a novas tecnologias e a novas plataformas. Além disso, Kulkarni, Barat e Ramteerthkar (2011a) descrevem que duas aplicações geradas nunca irão ter a mesma decisão de design, estratégias arquiteturais e plataformas. Com isso, era necessário identificar a versão mais próxima do MasterCraft e alterá-lo para as novas configurações.

Dessa maneira, Kulkarni, Barat e Ramteerthkar (2011a) decidiram reestruturar o MasterCraft, apesar dele ainda não estar no tempo de entrega ideal. A partir disso, os autores analisaram o processo de desenvolvimento que estavam utilizando, o processo Cascata, além de usar uma estrutura tradicional de times para a evolução do MasterCraft. As atividades de desenvolvimento eram caracterizadas pelo grande detalhamento do planejamento e uma rigorosa revisão do processo, conforme descrito no processo Cascata.

Sendo assim, levava-se um grande tempo no processo de manutenção e evolução do conjunto de ferramentas, porém os usuários do MasterCraft exigiam um tempo menor de resposta a cada solicitação. Além disso, era difícil o planejamento para desenvolver rapidamente uma solução para as requisições pequenas e, também, da dificuldade de acompanhar as constantes mudanças de requisitos. Portanto, os autores decidiram mudar do processo Cascata para a metodologia Scrum (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

O Scrum foi escolhido por estabelecer um melhor mecanismo de *feedback* para todos os *stakeholders* e permitir uma rápida evolução no MasterCraft. Além disso, com as metodologias ágeis há uma diminuição na necessidade de documentação, com um aumento

da interação com os usuários. Outra vantagem, foi a necessidade de retrabalho devido à alteração do mesmo requisito, pois foi possível uma entrega rápida de cada alteração solicitada. Contudo, os autores encontraram limitações na metodologia ágil em todas as atividades de desenvolvimento do MasterCraft e, portanto, decidiram criar uma nova metodologia, mesclando o Scrum com o MDD. O resultado está representado na Figura 24 (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

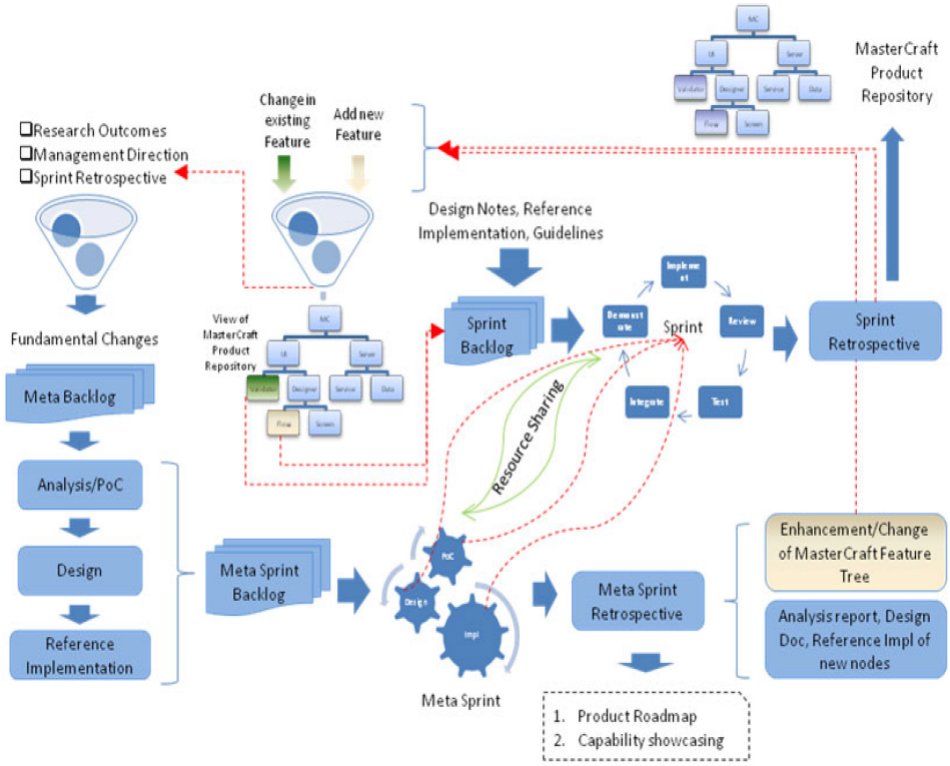


Figura 24: Metodologia Scrum adaptada as necessidades do desenvolvimento do MasterCraft (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

Kulkarni, Barat e Ramteerthkar (2011a) realizaram várias modificações na metodologia Scrum, sendo, a primeira, a melhoria no suporte a grande equipes de desenvolvimento com uma ampla variação de experiência e distribuídos geograficamente. Outra modificação foi que algumas atividades exigem uma extensa análise, experimentação e documentação detalhada e, com isso, não era possível realizá-las em pequenas *sprints*, sendo assim foram criadas as *meta-sprints* que possuem uma duração bem maior.

As *meta-sprints* são utilizadas para a realização de mudanças que necessitam uma mudança mais profunda no MasterCraft, como uma mudança de conceito, sendo que elas seguem um processo de desenvolvimento baseado no modelo Cascata. Uma *meta-sprints* começa com a análise da modificação. Após isso, há o Design da modificação e, por fim, é criado uma Implementação de Referência de cada *feature* (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

A Implementação de Referência utilizada por Kulkarni, Barat e Ramteerthkar (2011a) tem a mesma funcionalidade que a IR utilizada neste trabalho, pois ela é uma versão atual do MasterCraft e nela são implementadas as modificações e, caso essas modificações forem aprovadas, elas serão migradas para o repositório principal do conjunto de ferramentas.

Os clientes devem saber de todas as decisões nas metodologias ágeis, criando uma pressão desnecessária durante todo o estágio de estudo das modificações, além de ocasionar essa pressão na fase de testes e no controle de qualidade. Outro ponto é que as metodologias ágeis quase não produzem documentação e isso pode causar um grande problema para futuras manutenções, uma vez que existe a necessidade do cliente visualizar os *backlogs* das *sprints* e os meios de planejamentos o que atrapalha o desenvolvimento das atividades maiores. Por fim, nem todas as modificações são baseadas nas necessidades do cliente, portanto algumas *sprints* não geram *backlogs* (KULKARNI, BARAT e RAMTEERTHKAR 2011a).

Com a utilização da metodologia proposta, Kulkarni, Barat e Ramteerthkar (2011a) observaram que num primeiro momento conseguiram entregar apenas 50% das alterações prometidas com um atraso de duas semanas. Contudo, houve um aumento significativo na produção nas *sprints* subsequentes, conseguindo alcançar os prazos corretos com três *sprints*. Ao final de seu estudo, os autores relataram que conseguiram diminuir o tempo de entrega de seis meses para quatro semanas nas alterações utilizando *sprints* e de seis meses para três meses nas alterações utilizando *meta-sprints*.

Kim *et al.* (2005) (P11) propõem a metodologia *DRamatically Effective Application development Methodology* (DREAM) que é a aplicação dos conceitos do MDD com os conceitos da *Product Line Engineering* (PLE). O processo de PLE é constituído de dois sub-processos: *framework engineering* e *application engineering*. Cada sub-processo é composto por fases, conforme a Figura 25.

O primeiro sub-processo é o *framework engineering* que é composto por três fases de Análise de Domínio, Escopo do *Product Line*, Modelagem do *framework*. A primeira fase é a de Análise de Domínio que consiste na análise das *features* de várias organizações em um domínio particular e analisar seus aspectos em comum e sua variabilidade. Já a segunda fase é a de Escopo do *Product Line* (PL) que consiste em determinar o escopo do PL e quais *features* comuns e quais variações devem ser implementadas. Por fim, a terceira fase é a de Modelagem do *framework* que consiste na modelagem do *framework* independente da plataforma, gerando um modelo PIM (KIM et al. 2005).

Já o segundo sub-processo é o *application engineering* que é composto por seis fases: Análise de Requisitos da Aplicação, Especificação do Design da Aplicação, Instanciação do *framework*, Integração do Modelo, Detalhamento do Design da Aplicação e Implementação da Aplicação (KIM et al. 2005). A primeira fase é a Análise de Requisitos da Aplicação

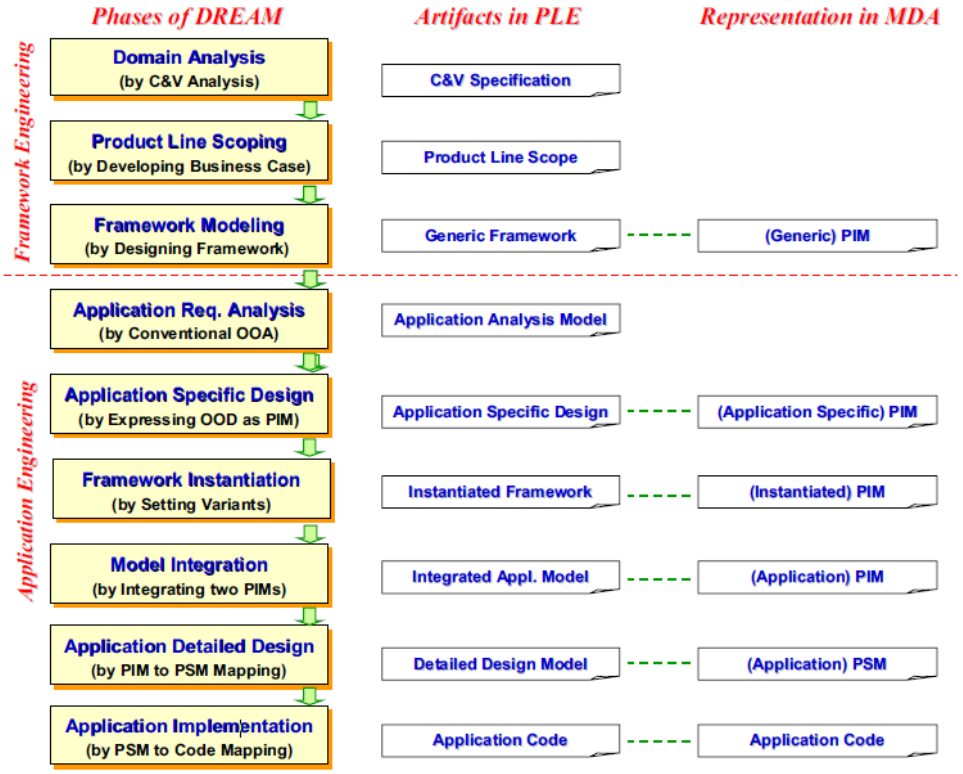


Figura 25: Metodologia DREAM (KIM et al. 2005).

que consiste na escolha de quais *features* do *framework* serão utilizadas e, a partir dessa decisão, a aplicação é modelada independente de plataforma, gerando um modelo PIM. A segunda fase é a de Especificação do Design da Aplicação que constitui-se na inclusão do modelo PIM da aplicação dos componentes e interfaces que serão utilizadas e, com isso, é possível integrar corretamente o *framework* na próxima fase. A terceira fase é a de Instanciação do *framework* na qual a transformação do modelo genérico PIM do *framework* para um modelo específico para a aplicação, sendo que esse modelo ainda é um modelo PIM. A quarta fase é a de Integração do Modelo, ou seja, a integração entre o modelo PIM do *framework* e da aplicação, resultando um outro modelo PIM. A quinta fase é a de Detalhamento do Design da Aplicação que se compõe na transformação do modelo PIM para o modelo PSM com a inclusão das características das plataformas que serão utilizadas pela aplicação. Por fim, a sexta fase é a de Implementação da Aplicação que compreende na geração do código, na sua finalização e seu teste (KIM et al. 2005).

Hebig e Bendraou (2014) (P12) realizaram uma revisão sistemática da literatura, com o objetivo de identificar os processos de Engenharia de Software combinados com MDD. Os termos utilizados nessa busca foram “agile”, “agility”, “SCRUM”, “V Modell”, “XP” e “RUP”, sendo esses termos combinados com o termo “model driven”. A string resultante foi executada nas bases ACM, IEEE, DBLP Computer Science Bibliography e Google Scholar.

Ao final de sua revisão, Hebig e Bendraou (2014) consideraram dez artigos, que estão descritos na Tabela 4. Nessa tabela está descrito o processo de Engenharia de Software (ES) utilizado, se a utilização do MDD segue os padrões definidos pelo OMG (2003) e se houve mudança no processo.

Artigo	Processo de ES utilizado	Utilização padrão do MDD?	Houve mudança no processo?
Loniewski, Armesto e Insfran (2011)	OpenUP	Sim	Sim
Yaghoobi, Torkamani e Yaghoobi (2011)	RUP	Sim	Sim
Liu *et al.* (2004)	RUP	Utilização específica de alguns padrões do MDD	Não
Grigera *et al.* (2012)	SCRUM e TDD	Utilização específica de alguns padrões do MDD	Não
Kulkarni, Barat e Ramteerthkar (2011a)	SCRUM	Utilização específica de alguns padrões do MDD	Não
Bostrom Nakicenovic (2012)	SCRUM e Kanban	Sim	Não
Zhang e Patel (2011) e Matinnejad (2011)	SLAP	Sim	Não
Fieber e Petrasch (2005)	V-Modell XT	Sim	Sim
Rausch *et al.* (2005)	V-Modell XT	Sim	Não

Tabela 4: Publicações encontradas por Hebig e Bendraou (2014)

Hebig e Bendraou (2014) concluem que há várias formas de se adaptar um processo convencional para o MDD. Todas as publicações encontradas demonstraram que os princípios do MDD são uma forte influência no processo de desenvolvimento de software. Além disso, parte dos estudos mostraram que devido à grande extensão das adaptações necessárias no processo de desenvolvimento, há a necessidade de um conhecimento prévio para se prever as influências do MDD sobre o processo.

Por outro lado, há variações na adoção dos padrões do MDD, uma vez que isso é observado tanto em metodologias e processos ágeis e tradicionais. Em alguns casos, é possível a utilização dos padrões estabelecidos para o MDD pelo OMG (2003), sem que haja a necessidade da adaptação do processo (HEBIG e BENDRAOU 2014).

A variância na adaptação da adoção do MDD existe devido à falta de um método que pode ou deve ser seguido nessas situações. Desse modo, ainda há uma necessidade de um conhecimento sistemático do impacto do MDD sobre os processos de Engenharia de Software e os passos para a realização das adaptações necessárias para que haja essa combinação (HEBIG e BENDRAOU 2014).

Por sua vez, Asadi, Ravakhah, Ramsin (2008) (P13) propõem um ciclo de vida de desenvolvimento de sistemas baseado em MDD, que pode ser instanciado e utilizado como uma metodologia de desenvolvimento de sistemas baseado em MDD que aplica os conceitos do *Situational Method Engineering* (SME). Essa metodologia proposta é cha-

mada de MDA-SDLC (*MDA-based System Development Lifecycle*), a qual foi desenvolvida adotando as ideias de ciclo de vida de desenvolvimento de sistema genérico e a abordagem MDD. Como a metodologia MDA-SDLC não é uma metodologia concreta, mas um processo geral que define fases e atividades esperadas em uma metodologia MDD. Ela foi dividida em cinco fases (Figura 26): Iniciação do Projeto, Desenvolvimento do PIM, Desenvolvimento do PSM e Código, Implantação e Manutenção.

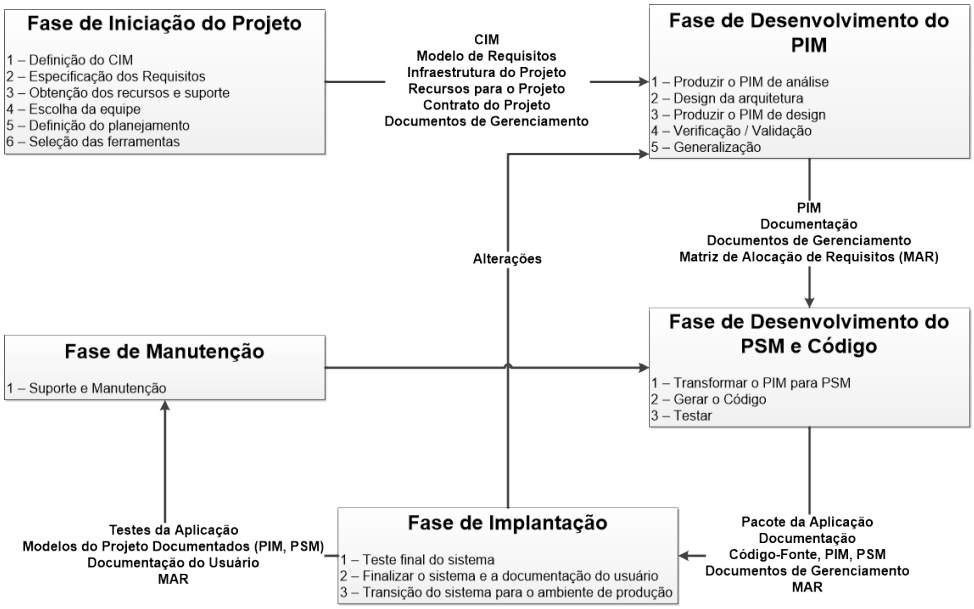


Figura 26: Proposta da metodologia *MDA-based System Development Lifecycle* (MDA-SDLC) (ASADI, RAVAKHAH e RAMSIN 2008).

A primeira fase da metodologia MDA-SDLC é a fase de Iniciação do Projeto, que é composta pelas atividades abaixo, tendo como saída o modelo CIM, o modelo de requisitos, a infraestrutura do projeto, os recursos para o projeto, o contrato do projeto e os documentos de gerenciamento (ASADI, RAVAKHAH e RAMSIN 2008).

- **Definição do CIM**: O escopo do sistema é bem definido dentro do domínio. Ao final dessa atividade será produzida uma desambígua definição do sistema, de seus objetivos e seu escopo;
- **Especificação dos Requisitos**: Os requisitos do sistemas são especificados utilizando a Engenharia de Requisitos e com técnicas de Coleta de Requisitos, tendo como resultado o Modelo de Requisitos;
- **Obtenção dos recursos e suporte**: Análise da viabilidade do projeto e a obtenção dos recursos necessários para a realização do mesmo;
- **Escolha da equipe**: O time de desenvolvimento é organizado;

- **Definição do planejamento**: Análise e estimativa do risco e o esforço necessário para o cumprimento de cada requisito, baseando-se na prioridade de cada um. Ao final dessa atividade, é criado o planejamento do sistema que será revisado e completado ao longo do desenvolvimento;

- **Seleção das ferramentas**: a ferramenta geralmente gerencia as atividades baseadas em MDD, como a definição de modelos e metamodelos, transformação dos modelos, geração de código e definição das constantes e regras para verificar os modelos.

A segunda fase da MDA-SDLC é a do Desenvolvimento do PIM, que é composta pelas atividades abaixo, tendo como saída o modelo PIM, a documentação, documentos de gerenciamento e a Matriz de Alocação de Requisitos (ASADI, RAVAKHAH e RAMSIN 2008).

- **Produzir o PIM de análise**: O modelo PIM de análise é criado a partir do modelo de requisitos onde ocorrem as funcionalidades do sistema enquanto é mantida a rastreabilidade até os requisitos. Esse modelo PIM não é a versão final, mas um protótipo para que a versão final possa ser gerada;

- **Design da Arquitetura**: Nessa atividade, o Arquiteto de Sistemas cria a arquitetura do sistema. Além disso, se necessário, o planejamento pode ser revisto;

- **Produzir o PIM de design**: O modelo PIM de análise é refinado para produzir o modelo PIM de design que contém, de forma bem detalhada, a estrutura e o comportamento do sistema. Técnicas convencionais de design OO podem ser utilizadas nessa atividade;

- **Verificação/Validação**: Nessa atividade é verificado se todos os modelos estão corretos e de acordo com os requisitos. Essa atividade é executada quantas vezes forem necessárias até que todos os modelos estejam corretos, antes que o modelo PIM seja transformado no modelo PSM;

- **Generalização**: Se for preciso, após a atividade de Verificação/Validação, os modelos podem ser generalizados, para que possam ser mais facilmente reutilizados em outros sistemas.

A terceira fase da MDA-SDLC é a fase de Desenvolvimento do PSM e do Código-Fonte, que é composta pelas atividades abaixo, tendo como saída o pacote da aplicação, a documentação, o código-fonte, os modelos PIM e PSM, documentos de gerenciamento e o RAM (ASADI, RAVAKHAH e RAMSIN 2008).

- **Transformar o PIM para PSM**: O modelo PSM é produzido a partir do modelo PIM, utilizando ferramentas MDD de transformação. As orientações para essa atividade são fornecidas pela documentação das ferramentas empregadas;

- **Gerar o Código**: O código-fonte é gerado a partir do modelo PSM, utilizando ferramentas MDD de geração de código. As orientações para essa atividade são fornecidas pela documentação das ferramentas adotadas;

- **Testar**: Nessa atividade estão incluídos os testes padrões de sistemas. Há a possibilidade de utilizar testes automatizados, mas é recomendável ter testes manuais, pois nem todos os erros são detectados pelos testes automatizados.

A penúltima fase da metodologia MDA-SDLC é a fase de Implantação do sistema, que é composta pelas atividades abaixo, tendo como saída os testes de aplicação, os modelos do projeto documentados (modelos PIM e PSM), documentação do usuário e o RAM (ASADI, RAVAKHAH e RAMSIN 2008).

- **Teste final do sistema**: Nessa atividade, é executado o teste final do sistema, sendo que deve ser gerado uma versão beta do sistema e alguns usuários finais devem ser selecionados para o teste dessa versão. Com isso, espera-se que todos os erros sejam encontrados. Caso surja algum erro nessa atividade, deve-se voltar para as fases anteriores;

- **Finalizar o sistema e a documentação do usuário**: Durante o ciclo de desenvolvimento do sistema, a documentação deveria ser escrita. Nessa atividade, essa documentação deve ser finalizada, juntamente com a versão final do sistema. Além disso, a documentação do usuário deve ser elaborada;

- **Transição do sistema para o ambiente de produção**: Nessa atividade, o sistema é instalado no ambiente de produção.

A fase final da metodologia MDA-SDLC é a fase de Manutenção, que é composta pela atividade de Suporte e Manutenção que consiste em manter o sistema em produção após a entrega do mesmo, tendo como entrada o teste da aplicação, documentação do projeto, os modelos PIM e PSM, documentação para o usuário e a Matriz de Alocação de Requisitos. Após a análise da modificação, deve-se voltar para a terceira fase da MDA-SDLC. Contudo, não há a definição de uma atividade formal de manutenção e evolução dos sistemas nessa metodologia (ASADI, RAVAKHAH e RAMSIN 2008).

Karagiannis, Hrgovcic e Woitsch (2011) (P14) um processo baseado em ME com a aplicação do MDD para a criação de *e-applications*. Seu processo é constituído por três

fases: Conceituação, Implementação e Implantação. A primeira fase é a fase de Conceituação, que consiste na definição do método de modelagem e tendo como entrada do método a descrição da análise (requisito, atores, entre outros) em um metamodelo independente de plataforma. Após isso, há a definição das plataformas. Essa fase tem como saída um documento detalhado com as especificações do método para o metamodelo das plataformas selecionadas. Já a segunda fase é a de Implementação que tem como entrada o documento detalhado com as especificações do método e, após as transformações, terá como resultado a implementação do método modelado. Por fim, a terceira fase é implantação e utilização do método no desenvolvimento para as plataformas selecionadas.

3.2.2.2 Métodos baseados em Modelos

Céret, Calvary e Dupuy-Chessa (2011) (P15) provaram em seu estudo de caso que ao aumentar a flexibilidade de um método já existente há a diminuição da limitação do uso desse método. Sendo assim, eles utilizam o método UsiXML que foi proposto por Vanderdonckt (2005). Esse método propõe uma abordagem e uma série de ferramentas para a geração de *User Interfaces* (UIs) adaptativas.

O UsiXML utiliza um modelo de tarefas que por meio de várias transformações gera um *User Interface* (UI) abstrata, um UI concreta e, por fim, a versão final da UI. Várias ferramentas suportam esse método, tais como a SketchiXML (COYETTE e VANDERDONCKT 2005), ResersiXML (BOUILLON et al. 2005), VisiXML ou GrafiXML (MICHOTTE e VANDERDONCKT 2008), sendo que essas ferramentas variam na possibilidade de criar um protótipo de UI mais ou menos detalhada e precisa (CÉRET, CALVARY e DUPUY-CHESSA 2011).

Apesar da grande quantidade de ferramentas disponíveis para o UsiXML, a flexibilidade desse método é parcial, pois não há a reutilização do conhecimento nem de elementos já existentes. Sendo assim, Céret, Calvary e Dupuy-Chessa (2011) propõe o aumento da flexibilidade desse método com a inclusão da reutilização do conhecimento e competências dos designers e dos desenvolvedores.

O estudo de caso realizado por Céret, Calvary e Dupuy-Chessa (2011) foi em uma indústria de energia nuclear, que necessitava atualizar a interface do sistema existente e aumentar o dinamismo da manutenção do sistema. Sendo assim, os autores converteram todas as UIs do sistema existente para a sua adaptação do método UsiXML. Depois disso, todas as UIs foram re-geradas com sucesso, necessitando apenas de algumas correções, contudo foi possível a reutilização do conhecimento já existente dos desenvolvedores e designers e, com isso, consegui-se provar que houve uma diminuição na limitação do uso do método.

Guelfi *et al.* (2004) (P16) criaram um método para desenvolvimento de sistemas distribuídos tolerantes a falhas utilizando o *framework* DRIP, chamado de DRIP Catalyst.

Esse método foi elaborado com o intuito de facilitar o reúso e a instanciação do DRIP. O método também engloba uma ferramenta de suporte ao desenvolvimento utilizando MDD, utilizando anotações baseadas na UML e foi implementado como uma extensão do IBM/Rational XDE. Para validação do método, os autores desenvolveram dois sistemas orientados a serviço e tolerantes a falhas e, ao final do experimento, o método se mostrou fácil de utilizar e compreender. Contudo os autores não validaram a tolerância a falhas e irão trabalhar na consistência entre o modelo e o código gerado.

Por sua vez, Garro e Tundis (2012) (P17) propõem um método iterativo para a análise de confiabilidade de sistemas, com o enfoque em sistemas de alto risco. Este método é composto por quatro etapas: Análise dos Requisitos de Confiabilidade, Modelagem do Sistema, Simulação do Sistema e Resultados conforme a Figura 27.

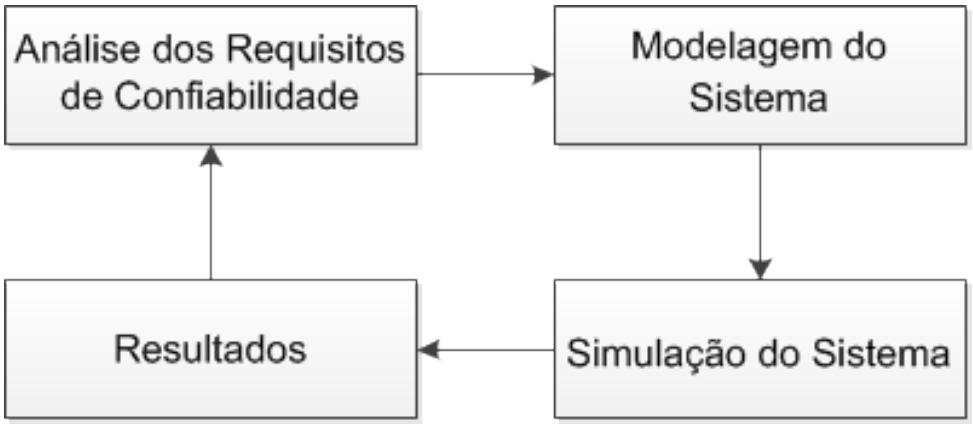


Figura 27: Proposta de um método baseado em modelos para a análise de confiabilidade de sistemas (GARRO e TUNDIS 2012).

Na primeira fase do projeto, os requisitos de confiabilidade do sistema são analisados e seus objetivos são especificados. Na segunda fase, a estrutura e o funcionamento do sistema são modelados utilizando a notação *SysML*, além disso, nessa fase a propagação e o gerenciamento das falhas são introduzidas. Na terceira fase, são realizadas diversas simulações em vários cenários. Por fim, os resultados dessas simulações são analisados de acordo com os requisitos de confiabilidade (GARRO e TUNDIS 2012).

A aplicação do método é exemplificada com a análise da confiabilidade de um sistema integrado para aviação, sendo que este se mostrou efetivo, flexível e escalável (GARRO e TUNDIS 2012).

Júnior (2012) (P18) criou um método chamado *Model-Driven Web Applications* (MDWA) para o desenvolvimento de aplicações Web utilizando MDD. Esse método é composto por sete etapas, conforme a Figura 28: Mapeamento dos Requisitos, Identificação dos Personagens, Identificação de Entidades, Identificação de Processos, Implementação da Infraestrutura, Detalhamento da Lógica de Negócio e Transformação para

Código-Fonte.

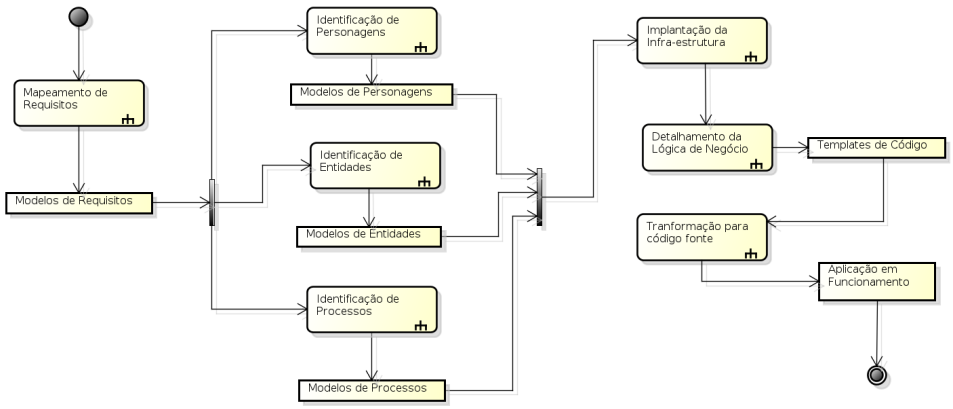


Figura 28: Método MDWA (JUNIOR 2012).

A primeira etapa é o Mapeamento dos Requisitos que tem como objetivo o levantamento dos requisitos da aplicação e tendo como resultado os modelos de requisitos. A segunda, é a Identificação dos Personagens que consiste na identificação dos possíveis usuários do sistemas e suas ações no mesmo e tem como saída os modelos de personagens. Enquanto que a terceira, é a Identificação de Entidades que representa a criação dos modelos de entidades do sistema, que deve conter todas as informações necessárias para sua implementação (JUNIOR 2012).

A quarta etapa é a Identificação de Processos que tem a finalidade da criação dos modelos de processos, os quais representam uma sequência de atividades que um ou mais personagens executam na aplicação, tendo como objetivo de prover um artefato ou serviço. A quinta etapa é a Implementação da Infraestrutura que tem como propósito a geração do código-fonte da infraestrutura do sistema a partir do modelo de requisitos. Para tal, é utilizado um *frameworks* do tipo *Middleware Intergration Framework* (MIF), que provem a arquitetura, testes automatizados, segurança, autenticação do usuário entre outros (JUNIOR 2012).

A sexta etapa é o Detalhamento da Lógica de Negócio que tem como propósito a criação de *templates* de código, seguindo as especificações da arquitetura da aplicação. Esses *templates* não geram apenas código-fonte, mas também, casos de testes, documentação, entre outros artefatos. Por fim, a sétima etapa é a Transformação para Código-Fonte que consiste na utilização de um *plug-in* gerador de código, como por exemplo, Acceleo ou JET e, após a geração do código, a aplicação deve ser testada e, caso tudo esteja de acordo, deve-se implantá-la (JUNIOR 2012). Contudo, não há a definição de uma etapa de manutenção e evolução de sistemas no método MDWA, tendo enfoque apenas na construção do software.

Júnior (2012) fez um caso de estudo para a avaliação de seu método. O caso de

estudo foi o desenvolvimento de um sistema Web para Gerenciamento de Projetos. O autor verificou que seu método auxiliou os desenvolvedores e se mostrou efetivo.

Soares (2012) (P19) criou um método para auxiliar a criação do *Model Plataform* (MP) ou Plataforma de Modelo (PM) no ambiente de *Real-Time Operating System* (RTOS), aplicando os conceitos do MDD, chamado de PM-MDA. O objetivo desse método é a separação dos conceitos da plataforma das transformações de modelos, possibilitando, dessa forma, uma maior reutilização das transformações em outros projetos.

O método PM-MDA é composto por cinco passos, conforme a Figura 29: Seleção do RTOS, Definição da Arquitetura de *Hardware* e de Processadores, Análise da *Application Programming Interface* (API) do RTOS selecionado, Construção do PM e Validação do PM RTOS. O primeiro passo é a Seleção do RTOS que consiste na definição do RTOS para qual o modelo será criado. Já o segundo passo é a Definição da Arquitetura de *Hardware* e de Processadores que tem como fim a definição de toda a arquitetura de *hardware* e processador que será englobada pelo PM (SOARES 2012).

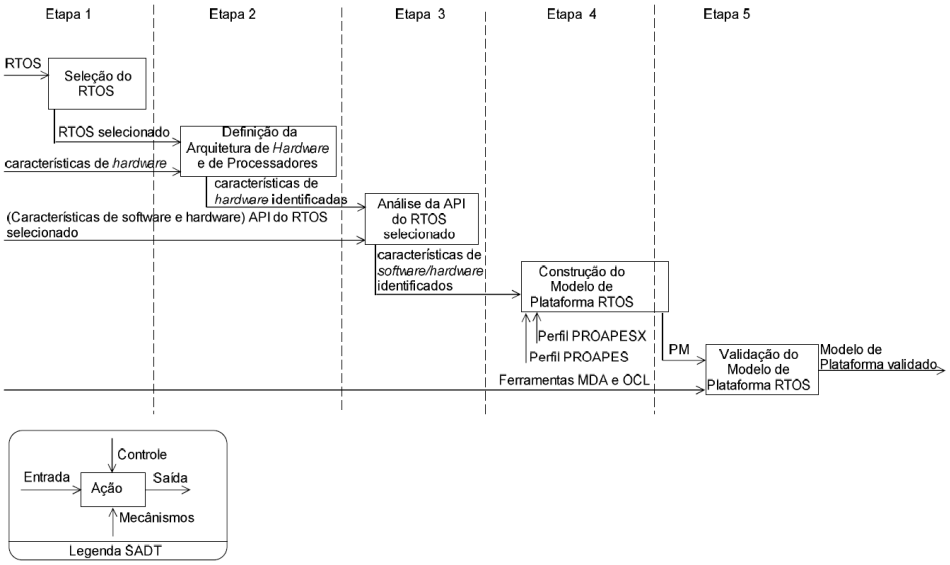


Figura 29: Método PM-MDA (SOARES 2012).

O terceiro passo é a Análise da *Application Programming Interface* (API) do RTOS selecionado, sendo que nesse passo é realizado uma análise mais profunda das API do *hardware* selecionado e, com isso, melhorar a qualidade do PM criado. O quarto passo é a Construção do PM que tem como objetivo criar um modelo que represente o *software* RTOS (definido no primeiro passo) e suas respectivas APIs (definidas no terceiro passo), juntamente com as características de *hardware* (definidas na segunda etapa). Por fim, o quinto passo é a Validação do PM RTOS, sendo que para essa validação é utilizado o *framework* de validação TOPCASED (SOARES 2012).

Utilizando o mesmo ambiente de Soares (2012), Agner (2012) (P20) criou o mé-

todo *Plataform Independent - Model Transformation* (PI-MT) com o intuito de auxiliar a criação de transformações reutilizáveis e aplicáveis a diferentes plataformas baseadas em *Real-Time Operating System* (RTOS). Esse método oferece a independência entre as transformação M2M dos modelos PIM para PSM e as características das plataformas, mediante a utilização de um *Model Plataform*.

O método PI-MT é composto por cinco passos, conforme a Figura 30: Criação do Modelo de Transformação, Definição do Modelo, Seleção do PM, Execução da Transformação e Verificação da Transformação. O primeiro passo é o de Criação do Modelo de Transformação que consiste na descrição do relacionamento entre os elementos do modelo-fonte e os elementos do modelo-alvo, estabelecido por meio de uma transformação. O Modelo de Transformação tem as seguintes características (AGNER 2012):

- **Domínio**: trata-se dos modelos que serão utilizados na transformação, sendo o domínio de origem composto pelos modelos PIM e PM e o domínio de destino o modelo PSM;
- **Organização**: engloba a estruturação geral das regras, que inclui o uso de ferramentas e/ou técnicas de modularização e reutilização;
- **metamodelos**: contém as características do domínio, contendo suas possíveis estruturas e os construtores dos modelos para o domínio em questão; e
- **Direcionalidade**: determina a direção da transformação, pois a mesma pode ser unidirecional ou multidirecional, sendo que o autor utiliza apenas as transformações unidirecionais.

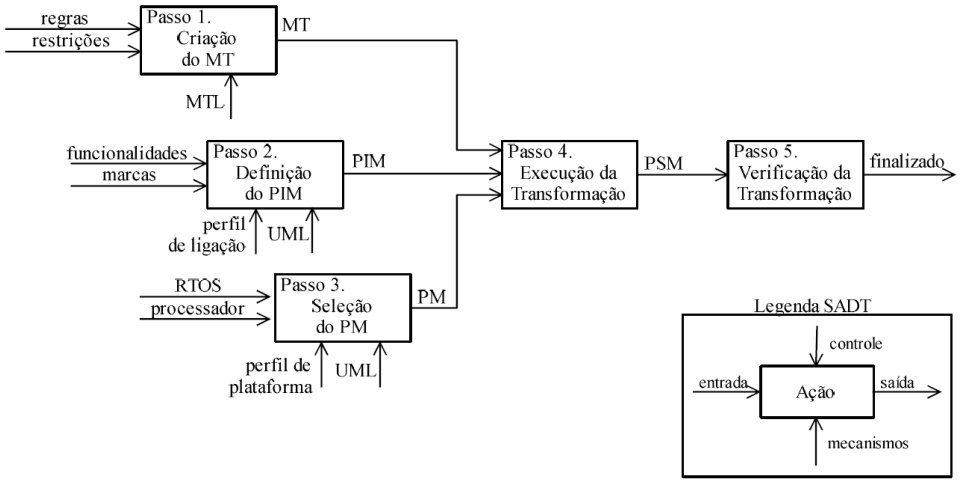


Figura 30: Método PI-MT (AGNER 2012).

O segundo passo é Definição do Modelo, que é a criação do modelo PIM. O terceiro passo é a Seleção do PM que consiste na definição dos conceitos das plataformas que serão

utilizadas pela aplicação. O quarto passo é a Execução da Transformação que tem como objetivo executar a transformação do modelo PIM, juntamente com o modelo PM, para o modelo PSM (transformação do tipo M2M). Por fim, o quinto passo é a Verificação da Transformação que tem como objetivo a verificação do modelo resultante da transformação (PSM) está de acordo com suas especificações (AGNER 2012).

Agner (2012) fez um caso de estudo para a avaliação de seu método, que foi o desenvolvimento de um sistema embarcado para ser um simulador de alarme. O autor verificou que o uso inicial de seu método tem um custo inicial de treinamento e na definição das regras de transformação, contudo, esse custo é atenuado com a facilitação da construção dos sistemas futuros, pois possibilita a reutilização dos modelos para diferentes plataformas.

3.2.2.3 Processos de apoio ao desenvolvimento utilizando MDD

Schramm *et al.* (2010) (P21) define um processo para manutenção e evolução da UI (*User Interface*) usando MDD para a automatização da geração das mesmas. Dessa forma, essa metodologia parte do pressuposto que os modelos de domínio da aplicação já estão criados e ela consiste em quatro passos, conforme representada na Figura 31.

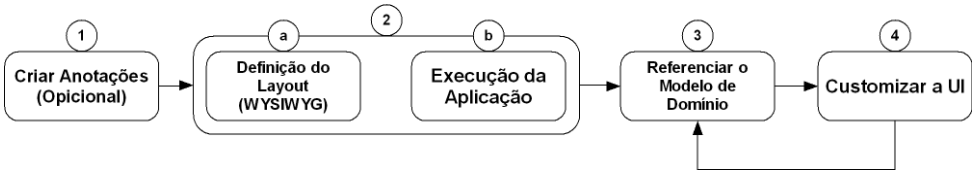


Figura 31: Proposta do processo para manutenção e evolução da UI (SCHRAMM et al. 2010).

O primeiro passo é chamado de Criar Anotações. Nesse passo, o desenvolvedor deve adicionar as anotações nos modelos de domínio para sua atualização. O segundo passo é a Definição do Layout e Referenciar o Modelo de Domínio. Primeiramente, o Design define o layout (Figura 31-2.a) e inclui as referências aos elementos do modelo de domínio (Figura 31-2.b), utilizando um editor WYSIWYG (*What You See Is What You Get*) (SCHRAMM et al. 2010).

Em seguida, a aplicação é executada (Figura 31-3), e graças à abordagem interpretativa proposta, não há a necessidade de outros passos de geração e compilação. No final, a UI pode ser customizada individualmente pelo usuário e adaptá-la as suas necessidades (SCHRAMM et al. 2010).

Para a avaliação desse processo, foi elaborado um estudo para a comparação entre o desenvolvimento tradicional e o desenvolvimento utilizando MDD e o processo proposto. Esse estudo foi a reconstrução de um editor baseado em formulários. O metamodelo desse editor era composto por 46 classes, com o total de 243 atributos. No desenvolvimento

tradicional foi gasto dez dias, já com a utilização do processo proposto, o gasto de tempo foi menor que duas horas, ou seja, nessa abordagem o custo de tempo foi reduzido para um décimo (SCHRAMM et al. 2010).

Cirilo (2011) (P22) criou um processo para desenvolvimento chamado *Model Driven Process to Construct Rich Interfaces for Context-Sensitive Ubiquitous Application* (Model Driven RichUbi) para Computação Ubíqua em plataformas Web, sendo que esse processo aplica os conceitos de MDD e *Domain-specific modeling* (DSM), tendo como enfoque na modelagem de interfaces e nas transformações M2C para diferentes tecnologias de implementação e, por isso, não se abordou todos aspectos do desenvolvimento, tais como, modelagem de dados, implementação da lógica de negócios, manutenção, entre outros.

O processo Model Driven RichUbi é composto por duas etapas, conforme demonstrado na Figura 32: Engenharia de Domínio e Engenharia de Aplicação. A Engenharia de Domínio engloba as atividades de construção dos artefatos para serem reutilizados nos projetos, onde esses artefatos são metamodelos do Domínio de Interfaces Ricas que são expressados utilizando DSL como apoio a modelagem, transformações M2C e adaptadores de conteúdo de interfaces. Os artefatos criados nessas atividades são armazenados no Repositório de Artefatos (CIRILO 2011).

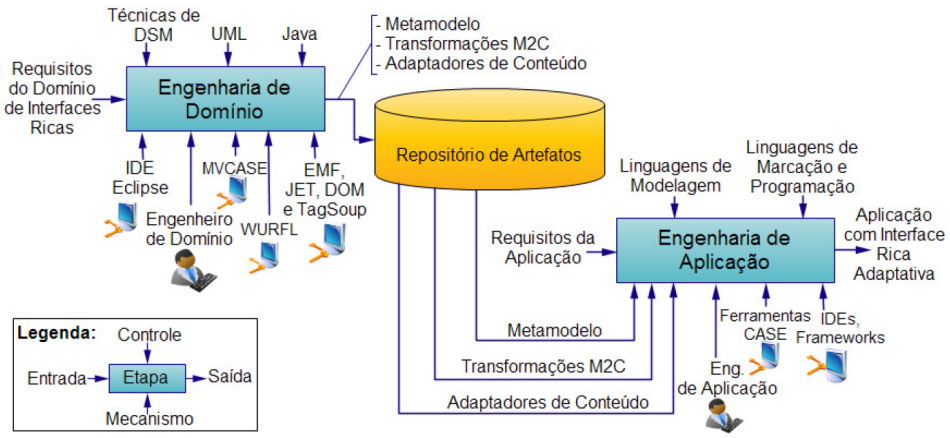


Figura 32: Processo Model Driven RichUbi (CIRILO 2011).

Enquanto que a etapa Engenharia de Aplicação é o desenvolvimento das aplicações reutilizando os artefatos produzidos na etapa descrita anteriormente. Ambas as etapas englobam as atividades tradicionais de projetos, sendo elas a Análise, Projeto, Implementação e Testes, sendo que essas atividades foram adaptadas às necessidades do processo (CIRILO 2011).

Cirilo (2011) avaliou seu processo com um estudo de caso que foi o desenvolvimento de uma aplicação no domínio de Atendimento de Urgência e Emergência e um

estudo comparativo, em que ambos os estudos se mostraram favoráveis à utilização do processo proposto, reduzindo o esforço de desenvolvimento, aumentando a produtividade e melhorando a adaptação das interfaces a vários dispositivos.

3.3 Considerações Finais

Apesar de não ter sido encontrado um método semelhante ao proposto por este trabalho, as publicações aqui listadas trouxeram contribuições para o mesmo, sendo que os trabalhos de Possatto (2013) e Perini (2015) serviram como base para a elaboração deste método.

Além disso, os trabalhos de Larrucea, Diez e Mansell (2004), Kulkarni, Barat e Ramteerthkar (2011a) e Asadi, Ravakhah e Ramsin (2008) englobam a fase de manutenção, contudo não há uma definição formal dessa atividade, ficando a caráter do engenheiro de software a implementação dessa atividade.

Outro ponto a ser destacado é a utilização de uma implementação de referência por Kulkarni, Barat e Ramteerthkar (2011a), cuja função é semelhante a deste trabalho, ou seja, ambas são uma versão atual do código-fonte que são utilizadas para a realização da modificação e sua testagem.

Sendo assim, a maioria dos trabalhos apresentados neste capítulo são de alto-nível e, portanto, não direcionam o desenvolvedor nas tarefas de manutenção e evolução de sistemas que utilizam a abordagem MDD, ficando a caráter do mesmo a definição dos passos para a realização dessas tarefas. Dessa forma, faz-se necessário a criação de um método que guie o desenvolvedor na manutenção e evolução desse tipo de sistema. O Capítulo 4 descreve o método MME-MDD que conduz o desenvolvedor nas atividades de manutenção e evolução dessa abordagem.

4 MME-MDD: Um Método para Manutenção e Evolução de sistemas baseados no MDD

Um método engloba orientações para a realização de determinada atividade e tem o intuito de melhorar a qualidade durante o processo de desenvolvimento de sistemas. A migração automática de código é um recurso para otimizar o tempo gasto no desenvolvimento e na qualidade dos sistemas computacionais. Contudo, as ferramentas de automação ainda possuem algumas restrições e necessitam de um conhecimento prévio do desenvolvedor para que seu uso alcance um bom custo-benefício. Portanto, um método para o uso de tais ferramentas é primordial para que o desenvolvedor faça o uso efetivo das mesmas.

Com o objetivo de facilitar as atividades de manutenção e evolução de sistemas e, consequentemente, a diminuição no esforço nessas atividades, neste trabalho se propôs a desenvolver um Método para Manutenção e Evolução de sistemas baseados no MDD (MME-MDD). Tal método norteia o desenvolvedor no uso das ferramentas elaboradas por Possatto (2013) e Perini (2015), auxiliando na escolha das tarefas que devem ser automatizadas, tendo como base a complexidade de cada uma, tornando, assim, efetiva a automação das mesmas. O MME-MDD também engloba outros dois níveis de abstração que não foram englobados pelos trabalhos de Possatto (2013) e Perini (2015), que são o nível de Modelo e de Metamodelo.

Além da elaboração do método, foi criada uma ferramenta baseada no Quadro Kanban com o intuito de facilitar a manutenção/evolução dos sistemas utilizando o método proposto. Como Quadro Kanban é uma ferramenta visual para o controle de tarefas, ela facilita a organização das tarefas e permite que se marque o progresso em cada tarefa, diminuindo a complexidade da realização da mesma. Outra vantagem da utilização do mesmo é a redução do tempo de realização da tarefa que essa ferramenta traz (FITZGERALD, MUSIAL e STOL 2014).

Além disso, com o intuito de facilitar a utilização e a compreensão do método e das ferramentas, foi elaborado um Tutorial explicando os conceitos do MDD, as ferramentas de Possatto (2013) e Perini (2015) e cada passo do método proposto.

Este capítulo está organizado da seguinte forma: Na Seção 4.1 é descrito o ambiente de desenvolvimento utilizado na concepção do método. Na Seção 4.2 é apresentado o estudo inicial das tarefas para o levantamento dos tipos de tarefas presentes na manu-

tenção e na evolução de sistemas utilizando a abordagem MDD. O método proposto está descrito na Seção 4.3. Na Seção 4.4 é apresentado a ferramenta de apoio ao método que foi baseada no Quadro Kanban e, na Seção 4.5, o Tutorial que foi elaborado para auxiliar no entendimento e na apresentação do método. Por fim, na Seção 4.6 há a finalização do capítulo com algumas considerações finais a respeito do método proposto.

4.1 Ambiente utilizado no estudo das tarefas e na concepção do método

O método proposto trabalha com quatro níveis de abstração, conforme representado na Figura 33. O primeiro nível de abstração é o metamodelo, sendo que sua DSL é criada utilizando o plug-in Xtext[1] para a IDE Eclpise[2]. A geração do *plug-in* da DSL do metamodelo é realizada pela execução do MWE2 *Workflow*.

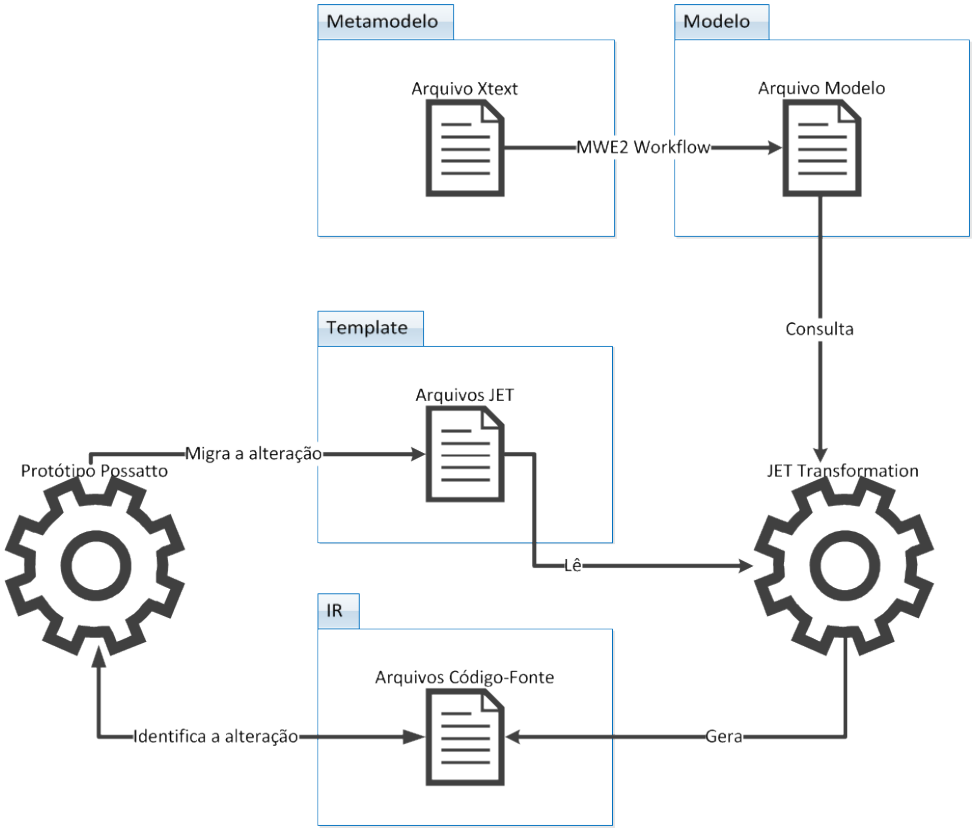


Figura 33: Esquematização do Ambiente utilizado no método.

Com o intuito de facilitar o *deploy* da DSL do metamodelo, foi utilizada uma segunda instância da IDE Eclipse para a edição do Modelo, dos *Templates* e do código-fonte da IR, dessa forma o desenvolvedor não precisa localizar a nova versão do *plug-in*

[1] plug-in Xtext - Versão 2.7.2.v201409160908
[2] IDE Eclpise - Versão Luna (4.4.1)

da DSL do metamodelo e incluí-la na pasta de *plug-ins* do Eclipse, basta excluir a pasta ".metadata" e, após isso, executar o *Run As Eclipse Application*.

A DSL do metamodelo criada tanto para os estudos teórico-práticos quanto para o experimento foi configurada para utilizar a extensão de arquivo **dmodel** para o modelo. Enquanto que o *Split JET Editor*[3], juntamente com o JET[4] e o JET *Mapping*[5].

É importante ressaltar que alguns passos do método proposto são dependentes das tecnologias supracitadas, principalmente do *Split JET Editor*, e existindo a necessidade de substituir alguma dessas tecnologias, será necessário a adaptação dos passos, tais como os sub-passos de Refazer o *deploy* do Metamodelo (Seção 4.3.3.1) e Editar o código-fonte da IR (Seção 4.3.3.2).

4.2 Estudo dos Tipos de Tarefas presentes na manutenção e na evolução com a abordagem MDD

O objetivo deste trabalho foi, primeiramente, investigar a melhor forma de realizar uma tarefa de manutenção ou evolução e, após esse estudo, propor um método para a sua realização. Para isso, foram realizados estudos teórico-práticos com base nos protótipos já desenvolvidos por Possatto (2013) e Perini (2015) para identificar que tipos de tarefas são mais ou menos suscetíveis à migração de código e propor um conjunto de atividades que irão guiar o desenvolvedor durante a migração.

Esses estudos teórico-práticos englobaram a realização novamente do experimento realizado por Possatto (2013) e Perini (2015). Além disso, foi desenvolvido uma nova aplicação utilizando os conceitos do MDD e, após isso, foram realizadas tarefas de manutenção e evolução.

O principal desafio da pesquisa é a complexidade das tarefas de manutenção e evolução de *templates* de geração de código. O trabalho de migração de código é complexo, pois envolve a mistura entre código gerado e código para geração, a sincronização dos artefatos e a duplicação de código. Consequentemente, este processo exige muita cautela para que não haja confusões entre código gerado e código para a geração.

Ainda com relação a esse assunto, a sincronização exige cuidado para manter a consistência entre a funcionalidade antes e depois da migração. Sincronizar significa replicar mudanças realizadas na IR para os *templates* e vice-versa. Caso a sincronização falhe, a quantidade de retrabalho necessária para corrigir eventuais problemas pode superar os benefícios da automação. Portanto, é importante uma definição precisa de como realizar esta tarefa.

[3] plug-in Split JET Editor - Versão 1.0.0.201505271417
[4] plug-in JET - Versão 1.1.1.v201101311015
[5] plug-in JET *Mapping* - Versão 1.0.0.201505271418

Além do exposto, este processo envolve quatro níveis de abstração: o nível do metamodelo, o nível do modelo, o nível de *template* e o nível de código gerado que acaba dificultando a manutenção, testes, depuração e implementação das modificações. Outra complexidade, existente na realização das tarefas de manutenção e evolução de sistemas, é a questão de quando deve ser realizada uma modificação no código-fonte da IR, quando deve ser alterado um *template*, quando deve o modelo deve ser atualizado e quando o metamodelo deve ser modificado.

Para a criação das tarefas de manutenção e evolução nos estudos teórico-práticos, foi tomado como base o estudo realizado por Corrêa, Oliveira e Werner (2011) que classificou as tarefas de manutenção e/ou evolução de acordo com o impacto que elas causam no sistema. Sendo assim as classificações criadas foram: *Non-Breaking Changes* (NBC), *Breaking but Resolvable Changes* (BRC) e *Breaking and Unresolvable Changes* (BUC).

As tarefas do tipo NBC não quebram a consistência entre os artefatos e suas regras de variação. Além disso, as alterações não precisam ser propagadas. A Tabela 5 mostra a aplicação das tarefas do tipo NBC em três níveis de abstração, sendo eles o metamodelo, modelo e *templates*.

Metamodelo	As mudanças no metamodelo não impactam nos modelos e transformações relacionados. Não há necessidade de evoluir ou atualizar as especificações de transformação (ex: inclusão da regra opcional de requerido nos atributos das classes).
Modelo	As mudanças no modelo são locais e não tem impacto nos outros modelos ou links de rastreabilidade. Portanto, não há necessidade de propagar as mudanças para outros modelos e, com isso, não há perda da rastreabilidade (ex: mudança do nome do atributo dataNasc para dataNascimento).
Templates	As mudanças nas especificações das transformações não invalidam os modelos gerados anteriormente. Com isso, não há necessidade de atualizar os modelos gerados e os links de rastreabilidade (ex: inclusão da classe CSS *currency* no campo custo).

Tabela 5: Aplicação e exemplo de tarefas do tipo NBC ao nível de metamodelo, modelo e *template*.

As tarefas do tipo BRC tem a quebra na consistência entre artefatos relacionados ou nas regras de variação, de modo que tanto as alterações que necessitam ser propagadas como a correção da quebra das regras de variabilidade, podem ser automaticamente identificadas e executadas. A Tabela 6 mostra a aplicação das tarefas do tipo NBC em três níveis de abstração, sendo eles o metamodelo, modelo e *templates*.

As tarefas do tipo BUC, assim como as modificações do tipo BRC, tem a quebra na consistência entre artefatos relacionados ou nas regras de variação, contudo não há a possibilidade de propagar essas alterações de forma automática. Sendo assim há a necessidade de intervenção por parte do desenvolvedor para a realização da propagação das

Metamodelo	A consistência entre os metamodelos e os modelos é quebrada. Com isso, o modelo correspondente deve ser migrado e as transformações atualizadas (ex: retirada do tipo *timestamp* por não ser utilizada).
Modelo	A consistência entre os modelos é quebrada. Com isso, os modelos devem ser sincronizados (ex: a classe Cliente não tem mais herança da classe Pessoa).
Templates	A consistência entre as transformações e os modelos gerados é quebrada. Primeiramente, os modelos gerados devem ser atualizados para as conformidades da especificação (ex: troca do nome fixo de uma variável para o nome de um atributo do modelo).

Tabela 6: Aplicação e exemplo de tarefas do tipo BRC ao nível de metamodelo, modelo e *template*.

modificações. A Tabela 7 mostra a aplicação das tarefas do tipo NBC em três níveis de abstração, sendo eles o metamodelo, modelo e *templates*.

Metamodelo	A consistência entre os metamodelos e os modelos é quebrada. Com isso, o modelo correspondente deve ser migrado e as transformações atualizadas (ex: criação da regra que obriga que todas as classes tenham um campo do tipo ID).
Modelo	A consistência entre os modelos é quebrada. Com isso, os modelos devem ser sincronizados (ex: mudança do nome da classe Pessoa para Cliente).
Templates	A consistência entre as transformações e os modelos gerados é quebrada. Primeiramente, os modelos gerados devem ser atualizados para as conformidades da especificação (ex: alteração do nome do arquivo templateHtml.jet para templatePagina.jet).

Tabela 7: Aplicação e exemplo de tarefas do tipo BUC ao nível de metamodelo, modelo e *template*.

Conforme o exposto, uma alteração realizada ao nível do metamodelo, como a inclusão da regra obrigatória de toda classe ter um atributo do tipo ID, como a inclusão de um atributo obrigatório que não está previsto no *template*, pode afetar os modelos e os *templates* e, da mesma forma, uma alteração ao nível do modelo, pode afetar os *templates*. Dessa forma, é fundamental que o método considere essas variações em cada tarefa a ser realizada, bem como, conclui-se que as alterações que são passíveis de automação na migração de código a nível de *template* são do tipo NBC e BRC, pois no primeiro não há quebra de sincronia entre o código-fonte da IR e o *template*, já com o segundo há quebra de sincronia, mas ela é resolvida de forma automática, tanto com a migração da alteração no código-fonte da IR para o *template*, como a execução de uma transformação para a re-geração do código-fonte da IR.

Dessa forma, conforme a Figura 34, foram identificados seis caminhos possíveis nas tarefas de manutenção e evolução de sistemas baseados em MDD. O primeiro caminho possível é a atualização do modelo, que ocorre quando há alteração num trecho de código-fonte associado às informações definidas no modelo e não nos *templates*, como por exemplo,

a alteração de um atributo de uma Classe. Após a alteração do modelo, basta re-gerar o código-fonte que a alteração já estará implementada.

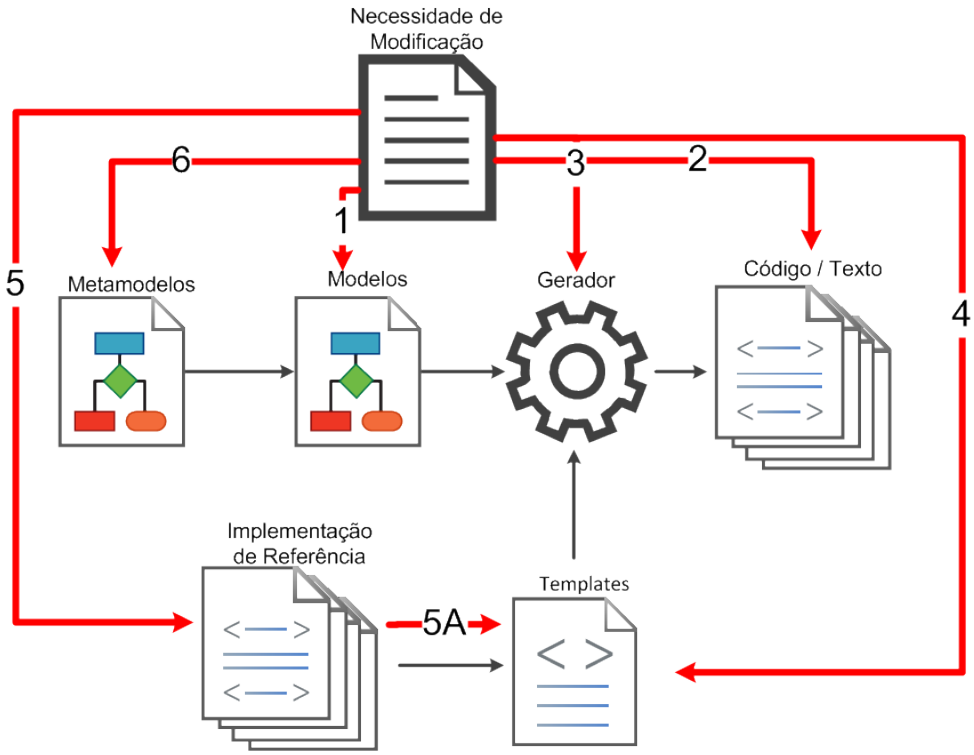


Figura 34: Esquematização dos caminhos possíveis nas tarefas de manutenção e evolução.

O segundo caminho possível é a modificação direta no código-fonte gerado, contudo quando houver uma nova geração de código, essas modificações serão perdidas. Além disso, essa modificação ficará restrita a um único sistema. Sendo assim, esse caminho deve ser evitado. Já o terceiro caminho é a modificação no gerador, isso é raro, pois as ferramentas de geração, como o JET e o MOFScript, de código baseadas em *templates* são bastante estáveis.

O quarto caminho é a modificação direta nos *templates*, sendo que a modificação direta neles é complexa e exige um grande nível de conhecimento da linguagem de programação utilizada e da linguagem utilizada pelo gerador para os *templates*, além de ser mais difícil de executar os testes. Outra questão é a perda de sincronia entre os *templates* e a IR e, portanto, esse caminho deve ser evitado.

O quinto caminho é a modificação do código-fonte da IR, sendo essa a melhor opção quando há a necessidade de modificar o código-fonte e não o modelo. Contudo, esse caminho exige um passo extra que é a migração das modificações para os *templates* correspondentes e isso tem um alto custo, mas com a utilização das ferramentas de Possatto (2013) e Perini (2015) esse custo é reduzido, sendo que apenas em algumas situações, como a inclusão de uma nova consulta ao modelo, é que se faz necessário a modificação

manual do *template*.

Por fim, o caminho seis é quando há a necessidade da modificação de um ou mais conceitos do metamodelo, sendo esse caminho com um alto custo de tempo, pois é necessário modificar os demais níveis de abstração (modelo, *template* e IR).

Como há seis caminhos possíveis em uma modificação, se fez necessário a criação de um método que auxilie o desenvolvedor na escolha de qual caminho seguir e quais serão os passos necessários para a conclusão do mesmo.

4.3 MME-MDD: Um Método para Manutenção e Evolução de sistemas baseados no MDD

Baseado no estudo apresentado na Seção 4.2, foi elaborado o método MME-MDD (Método para Manutenção e Evolução de sistemas baseados no MDD), que está representado na Figura 35. Este método engloba três tipos de tarefas: MDD, Criativa e Manutenção. As tarefas do tipo MDD são aquelas que irão modificar apenas o modelo, sem qualquer alteração a nível de metamodelo, *template* ou IR. O tipo Criativa engloba as modificações em um arquivo fixo ou templatizar um arquivo fixo, que consiste na criação de um *template* correspondente ao arquivo fixo. Por sua vez, as tarefas do tipo Manutenção podem alterar o metamodelo (podendo abranger a alteração do modelo) e da IR com a migração dessas alterações para o *template* utilizando o *Split JET Editor*.

O MME-MDD tem por início pelos passos de Analisar a modificação e Selecionar a primeira tarefa. O primeiro passo é Analisar a modificação, que consiste na identificação dos requisitos da modificação e a criação de suas tarefas na ferramenta Kanban, que está descrita na Seção 4.4. Após isso, a primeira tarefa deve ser selecionada e movida para a coluna *In Progress* na ferramenta Kanban. Em seguida, o desenvolvedor deve seguir os passos de acordo com o tipo da tarefa, que foi estabelecido na análise dos requisitos.

Os passos de cada tipo de tarefa estão descritos nas seções seguintes, sendo que na Seção 4.3.1 está a descrição dos passos necessários para as tarefas do tipo MDD. Já a Seção 4.3.2 está a descrição dos passos das tarefas do tipo Criativa. Por sua vez, os passos do tipo Manutenção e Evolução estão na Seção 4.3.3.

Após os passos específicos de cada tipo de tarefa, deve-se Testar as Modificações realizadas na tarefa atual e se ela está de acordo com os requisitos. Caso ela não esteja ou o sistema apresente algum erro, deve-se adicionar uma nova tarefa no quadro Kanban para a correção da mesma.

Por fim, caso exista alguma tarefa a ser realizada no quadro Kanban, a mesma deve ser selecionada e movida para a coluna *In Progress*.

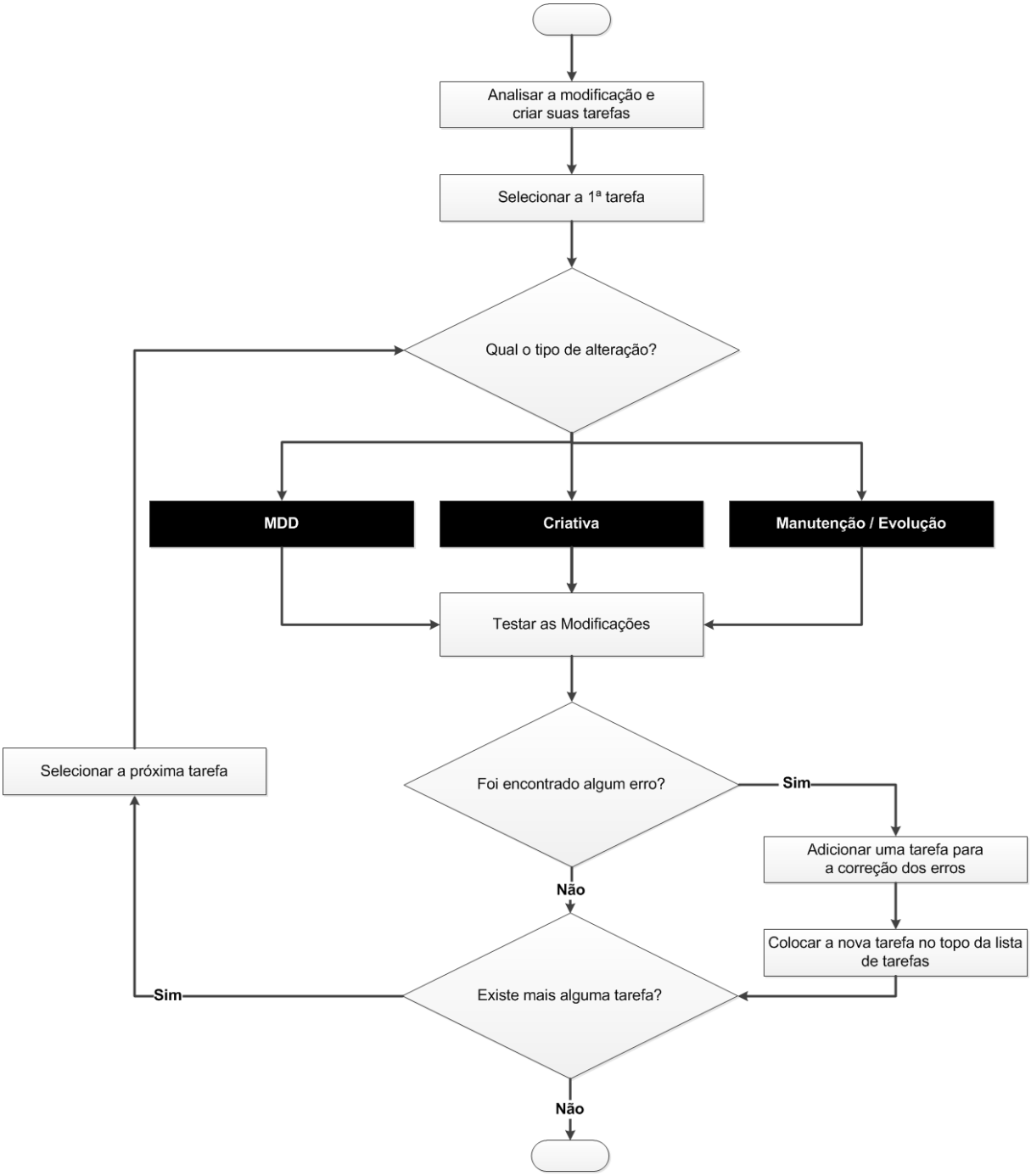


Figura 35: Método MME-MDD.

4.3.1 Passos das tarefas do tipo MDD

As tarefas do tipo MDD são as tarefas que irão apenas modificar o modelo, não englobando qualquer mudança a nível de metamodelo, *template* e/ou IR, sendo assim ela aplica totalmente os conceitos do MDD, pois com a alteração apenas de um modelo, é possível realizar toda uma modificação no sistema.

A Figura 36 apresenta os passos para a realização das tarefas do tipo MDD. O

primeiro passo é Tentar modificar o Modelo de acordo com as necessidades da tarefa e seguindo as definições do metamodelo. Após isso, deve-se Tentar executar novamente a transformação, sendo que essa transformação é realizada pelo *plug-in* JET e irá gerar novamente todo o código-fonte da IR e, caso não exista erros, irá aplicar todas as alterações na Implementação de Referência.

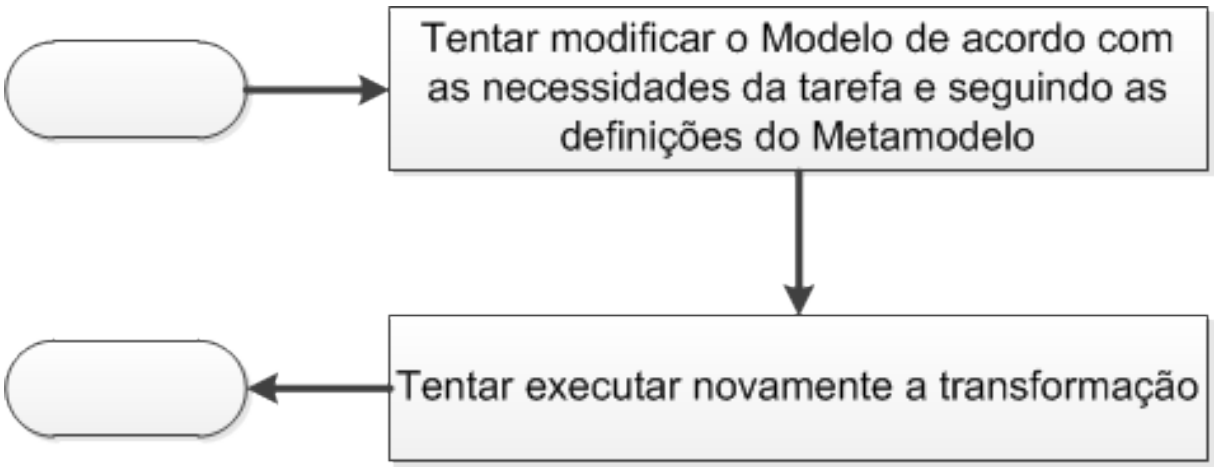


Figura 36: Passos do tipo MDD no método MME-MDD.

Nesse tipo de tarefa, engloba as modificações como a inclusão de um atributo em uma classe (exemplo: inclusão do campo de desconto na classe de Contas a Receber), a criação de uma nova classe (exemplo: criação da classe Funcionário com os campos de CPF, Nome, Data de Nascimento e Cargo) e a modificação de um atributo (exemplo: mudança do nome do atributo dataNasc para dataNascimento ou mudança do tipo do atributo salario de inteiro para *currency*).

4.3.2 Passos das tarefas do tipo Criativa

As tarefas do tipo Criativa incluem as tarefas de templatizar arquivos fixos e/ou modificação de arquivos fixos. Os arquivos são considerados fixos quando não são gerados por um ou mais *templates*, sendo adicionados manualmente no sistema, tais como os arquivos de CSS e JS. Com isso não é possível utilizar o *plug-in Split JET Editor* e há a necessidade de total intervenção do desenvolvedor nesse tipo de modificação. Os passos para a realização deste tipo de tarefa estão representados na Figura 37.

Primeiramente, deve-se Localizar e abrir os arquivos da IR que precisam ser modificados. Após a abertura do primeiro arquivo, há duas possibilidades: a primeira, é templatizar o arquivo atual, de modo que, inicialmente, deve-se Copiar o arquivo fixo para o novo *Template* e, em seguida, Colocar um *include*, seguindo a sintaxe do JET, em um *Template* já existente. Como o arquivo fixo já está presente na IR, não há a necessidade de executar uma transformação JET. A segunda possibilidade é a edição de um arquivo

fixo, sendo que nesse passo, deve-se editar o arquivo fixo e, caso haja a necessidade de aplicar essa alteração em outros projetos. Para isso, deve-se migrar de forma manual, pois não há mapeamento para esse tipo de arquivo. Ao fim das duas possibilidades, deve-se editar o próximo arquivo, até terminar todos os arquivos que precisam ser modificados.

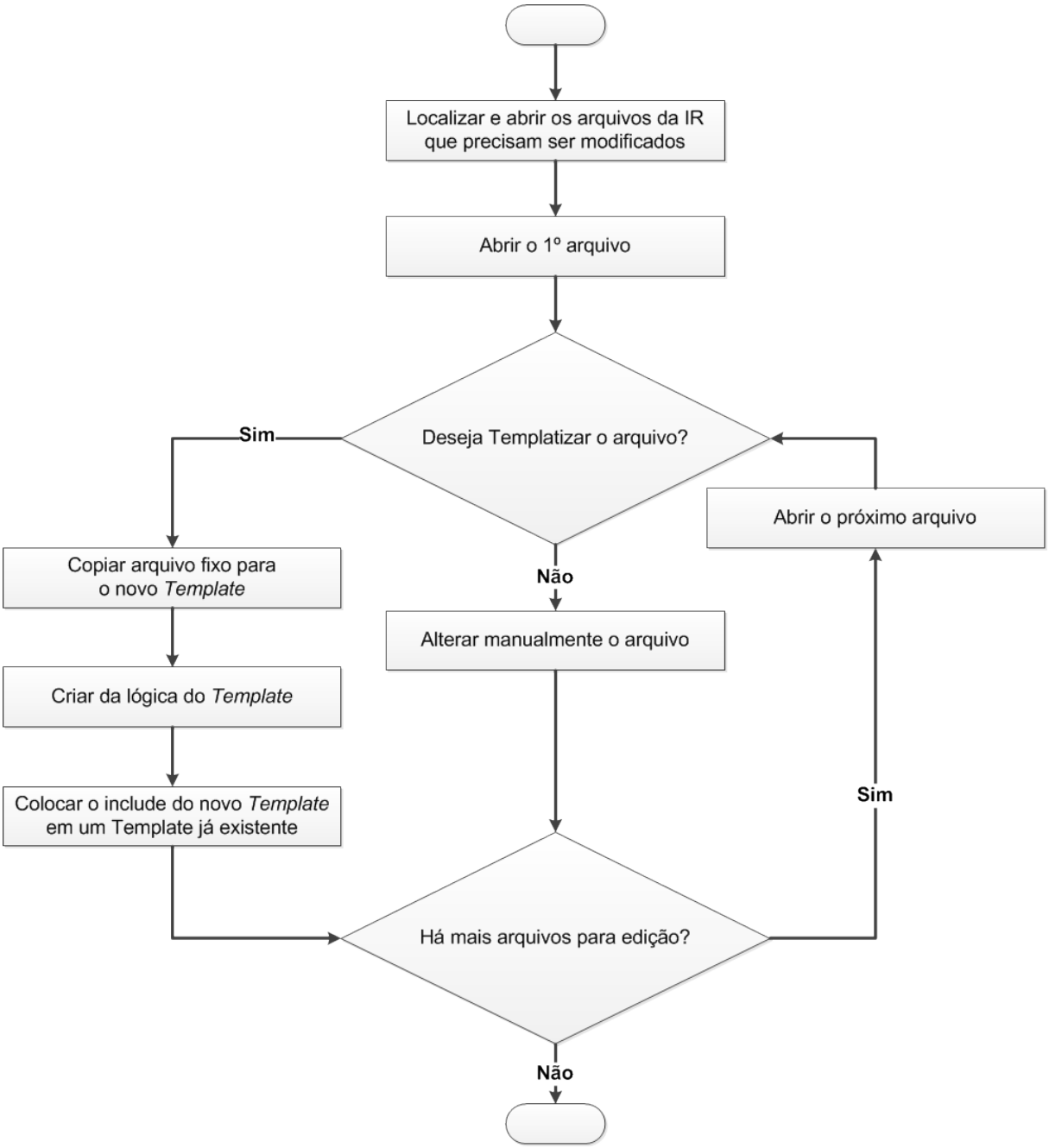


Figura 37: Passos do tipo Criativa no método MME-MDD.

Nesse tipo de tarefa, engloba as modificações como a mudança de um arquivo JS ou um arquivo CSS e copiar o conteúdo do um arquivo JS ou um arquivo CSS e colocá-lo em um *template*.

4.3.3 Passos das tarefas do tipo Manutenção e Evolução

As tarefas do tipo Manutenção e Evolução contemplam as modificações no metamodelo e/ou no código-fonte da IR e *template*. Contudo, conforme exposto na Seção 4.2, deve-se evitar a edição direta dos *templates*, pois a modificação dos mesmos irá causar a perda da sincronia entre a IR e eles. Os passos para a realização deste tipo de tarefa estão representados na Figura 38.

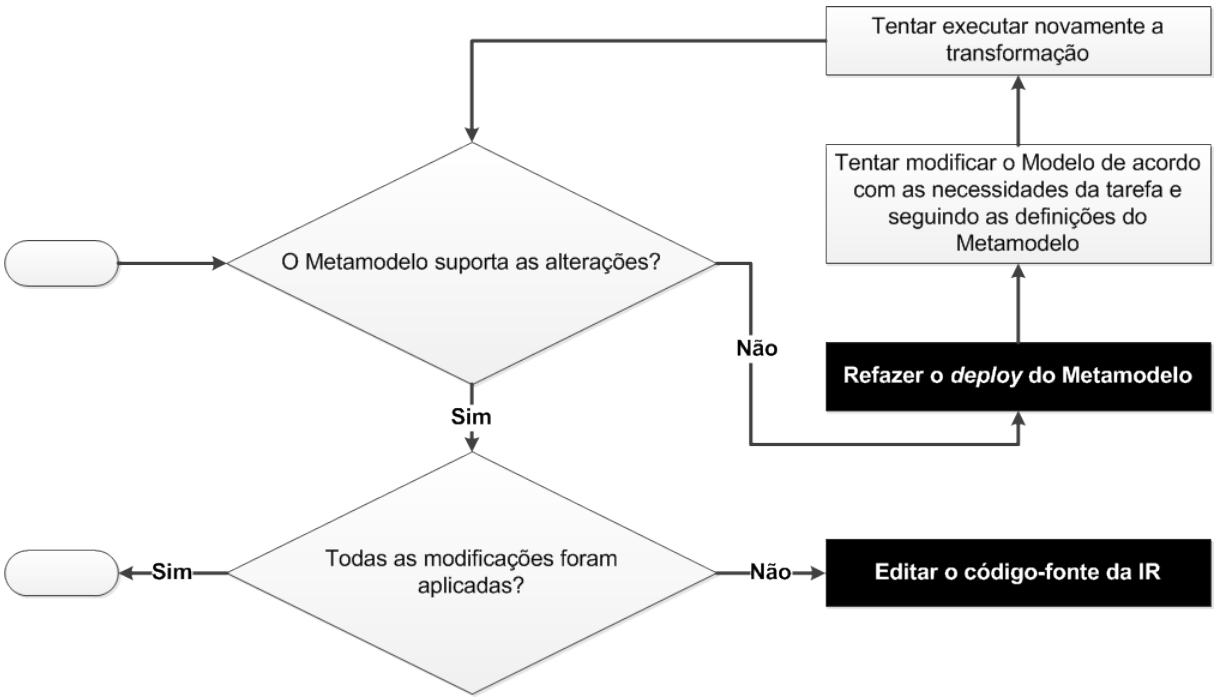


Figura 38: Passos do tipo Manutenção no método MME-MDD.

No tipo Manutenção e Evolução, de acordo com a análise dos requisitos, pode existir a necessidade de modificar o metamodelo e, para isso, deve-se seguir os passos que estão descritos na Seção 4.3.3.1. Após refazer o *deploy* do metamodelo, deve-se Tentar modificar o Modelo de acordo com as necessidades da tarefa, seguindo as novas definições do Metamodelo e, após essa alteração, Tentar executar novamente a transformação.

Caso a transformação ou o modelo apresentaram erros devido à alteração realizada do metamodelo, deve-se seguir novamente os passos de edição do metamodelo. Caso seja outro tipo de erro, deve-se adicionar uma nova tarefa para correção do mesmo. Caso não existam erros, pode-se prosseguir para o próximo passo, que poderá ser a edição do código-fonte da IR ou o término dos passos deste tipo de tarefa. Os passos necessários para a edição do código-fonte da IR e dos *templates* estão descritos na Seção 4.3.3.2.

4.3.3.1 Sub-passos do passo Refazer o *deploy* do Metamodelo

Os passos para a realização do *deploy* do Metamodelo estão apresentados na Figura 39. Primeiramente, deve-se Fechar a instância atual do eclipse, que é a instância de edição do Modelo, *templates* e IR. Logo após, deve-se abrir o arquivo com extensão ".xtext" no projeto da DSL do metamodelo. Com isso, deve-se Modificar e salvar o Xtext de acordo com as necessidade da tarefa e seguindo sua sintaxe.

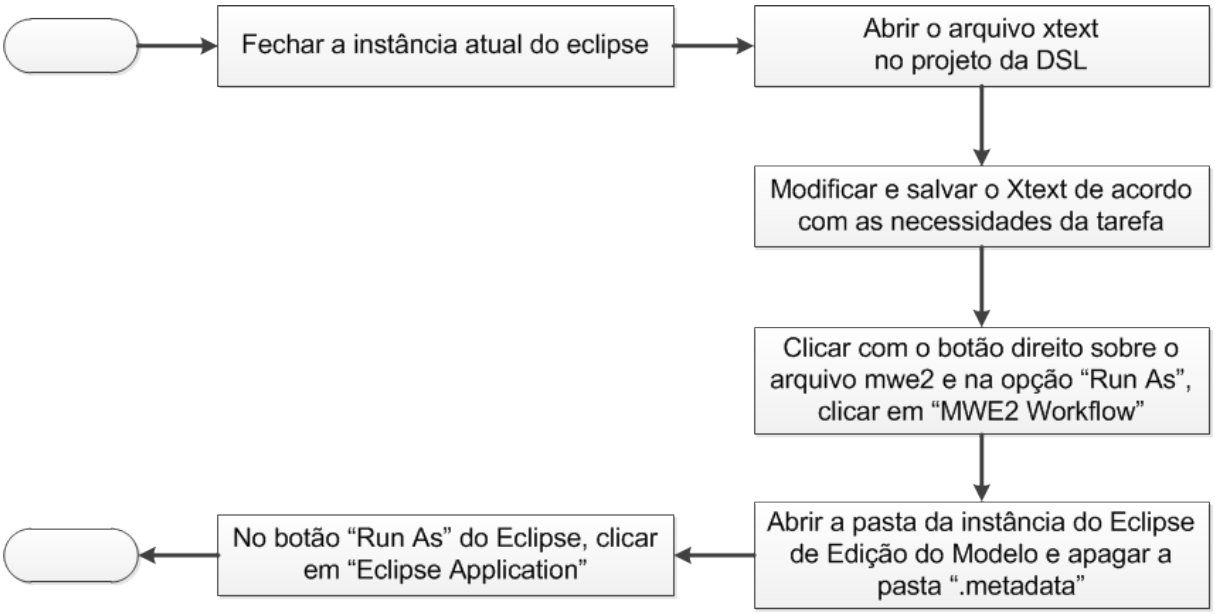


Figura 39: Sub-passos do passo "Refazer o *deploy* do Metamodelo" no método MME-MDD.

Após a edição do Xtext, é necessário gerar novamente o *plug-in* da DSL do metamodelo e, para isso, deve-se Clicar com o botão direito sobre o arquivo com extensão ".mwe2" e na opção "*Run As*", clicar em "*MWE2 Workflow*". Dessa forma, o Eclipse irá gerar uma nova versão do *plug-in* da DSL. Para que não exista nenhum conflito com versões antigas do *plug-in* da DSL, é necessário apagar a pasta ".metadata" da instância de edição do Modelo, *templates* e IR. Por fim, para abrir novamente essa instância, deve-se No botão "*Run As*" do Eclipse, clicar em "*Eclipse Application*" e o Eclipse irá abrir a nova instância.

4.3.3.2 Sub-passos do passo Editar o código-fonte da IR

Os passos para a realização da edição do código-fonte da IR estão apresentados na Figura 40. Conforme descrito na Seção 3.1.2, quando habilitado, o ambiente *Split JET Editor* tem cinco status possíveis:

- Edição Habilitada;
- Arquivo de *template* fechado;

- Editando Região de Modelo;
- Editando um arquivo de *template*; e
- Sincronia perdida.

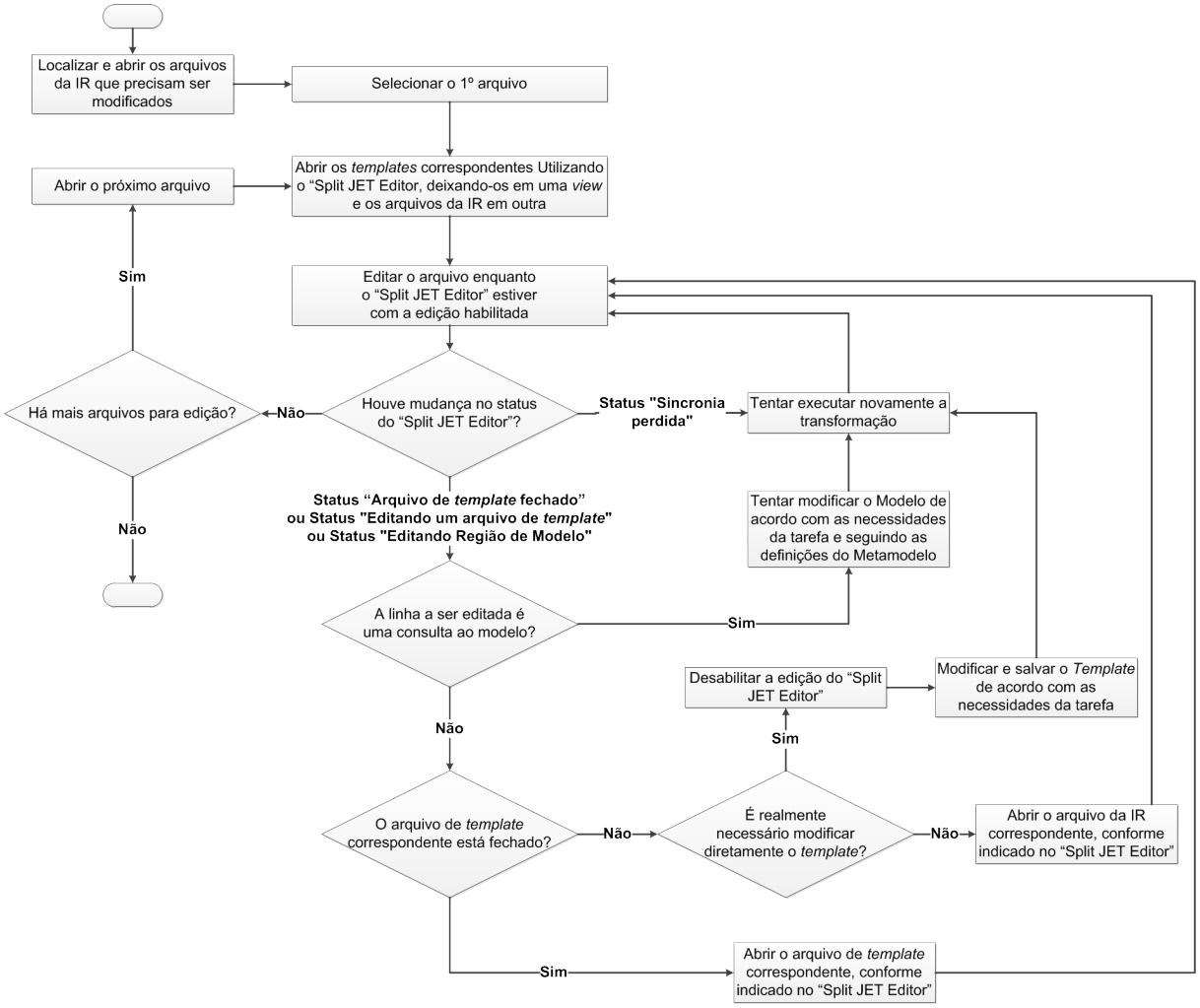


Figura 40: Sub-passos do passo "Editar o código-fonte da IR" no método MME-MDD.

O status "Edição Habilitada" é o status padrão do *Split JET Editor*, sendo representado pela cor de fundo do texto verde tanto na código-fonte da IR como no respectivo trecho de código do *template*. Quando o *template*, que gerou o trecho de código-fonte da IR que está sendo editado, está fechado, o ambiente impede a edição e muda a cor de fundo do texto para vermelho. Isso ocorre devido à necessidade do *template* estar aberto para que as alterações possam ser migradas automaticamente.

O status "Editando Região de Modelo", ocorre quando o trecho de código da IR, que está sendo editado, foi gerado por uma consulta ao modelo, o qual deve ser editado e não a IR. Dessa forma, o *Split JET Editor* bloqueia a edição nesse trecho e muda a cor de fundo do texto para vermelho em ambos os lados.

Conforme o exposto neste trabalho, deve-se evitar ao máximo editar diretamente o *template*, por isso o *Split JET Editor* não permite a edição de *templates*, bloqueando totalmente a edição, dessa forma, somente é possível editá-los diretamente com a desativação do ambiente. Ese tipo de edição ocorre quando há a mudança em uma consulta ao modelo ou a necessidade da inclusão de uma TAG JET.

A sincronização entre a IR e os *templates* é perdida quando um outro arquivo da IR já modificou aquele trecho de código do *template* e, ainda, não foi re-executada a transformação. Então, o *Split JET Editor* muda a cor de fundo do texto para amarelo em ambos os lados e solicita que a transformação seja executada.

O primeiro passo para a realização da edição do código-fonte da IR é Localizar e abrir os arquivos da IR que precisam ser modificados e, após isso, selecionar o primeiro arquivo. Utilizando o *Split JET Editor*, abrir os *templates* correspondentes, deixando-os em uma *view* e os arquivos da IR em outra, conforme o exemplo da Figura 41. Nesse exemplo, o desenvolvedor está editando um arquivo SQL, que está a direita, e o *Split JET Editor* está demonstrando qual linha no *template* corresponde a linha que está sendo editada na IR, que está a esquerda.

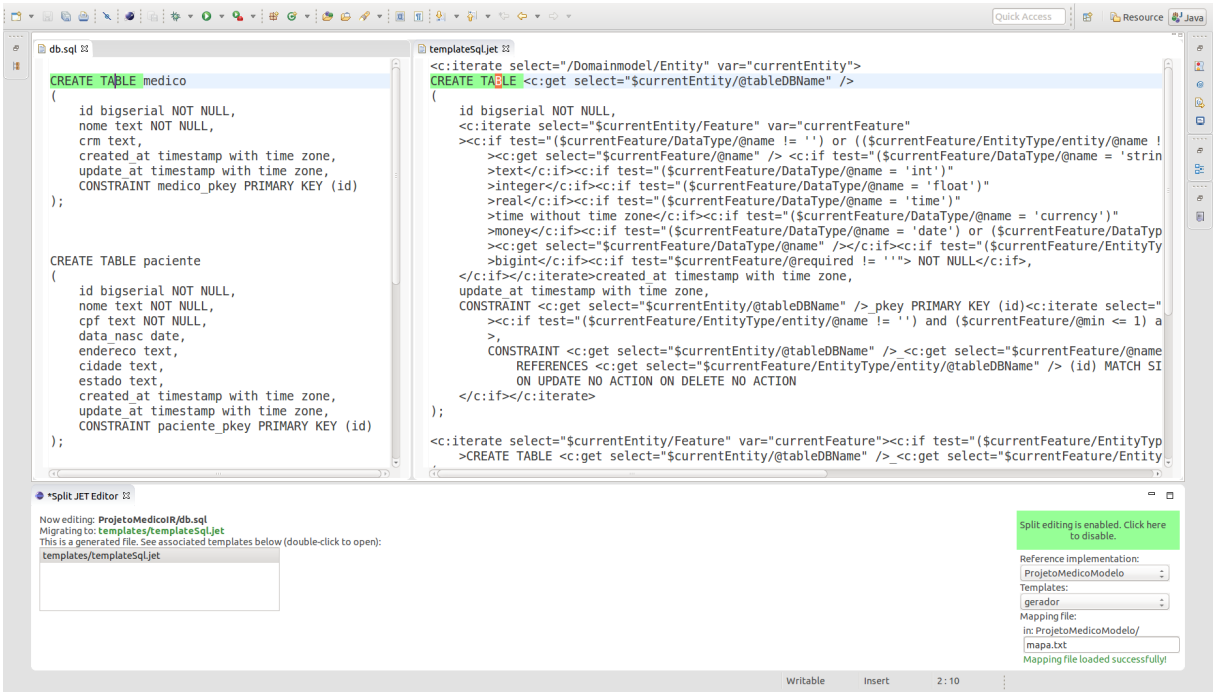


Figura 41: Exemplo de Divisão da Tela para edição simultânea da IR com os *templates*.

Logo após, deve-se fazer a modificação dos arquivos enquanto o *Split JET Editor* estiver com o status “Edição Habilitada”. Caso não exista mudança no status, deve-se ir editar o próximo arquivo, se existir, ou finalizar este sub-passo. Caso o status mude para “Sincronia perdida”, basta executar novamente a transformação JET, sendo que

esse status ocorre quando o trecho de código do *template* correspondente já foi modifico anteriormente e não houve uma nova geração de código.

Caso o status mude para "Arquivo de *template* fechado", basta abrir o arquivo de *template* correspondente, conforme indicado no *Split JET Editor* e exemplificado na Figura 42, isso ocorre quando o desenvolvedor tenta modificar um arquivo que foi aberto recentemente e seu *template* correspondente ainda não foi aberto.

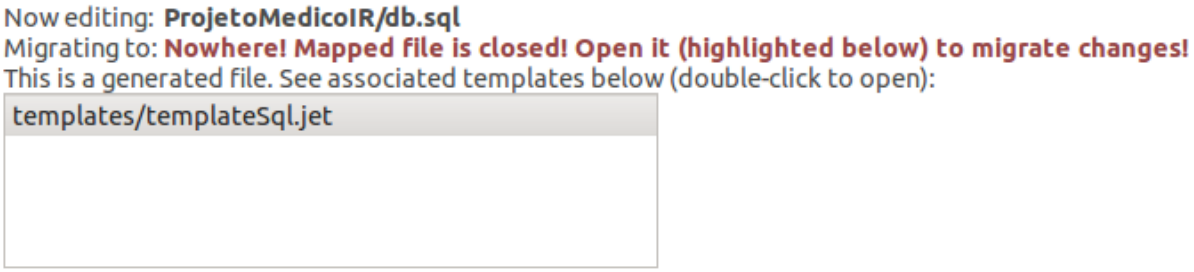


Figura 42: Exemplo de indicação do arquivo de *template* fechado *Split JET Editor*.

Caso o status mude para "Editando um arquivo de *template*", deve-se abrir o arquivo da IR correspondente, conforme indicado no *Split JET Editor* e exemplificado na Figura 42, esse status ocorre quando o desenvolvedor clica em algum trecho do código do *templae*. Contudo, se realmente for necessário a edição direta no *template*, deve-se desabilitar a edição do *Split JET Editor* e modificar o *template* e, em seguida, executar a transformação para que a sincronia, através do mapeamento, entre a IR e os *templates* seja re-estabelecida. A edição direta em um *template* deve ocorrer somente quando há a necessidade da inclusão ou modificação de uma TAG JET, caso contrário, deve-se modificar apenas o código-fonte da Implementação de Referência.

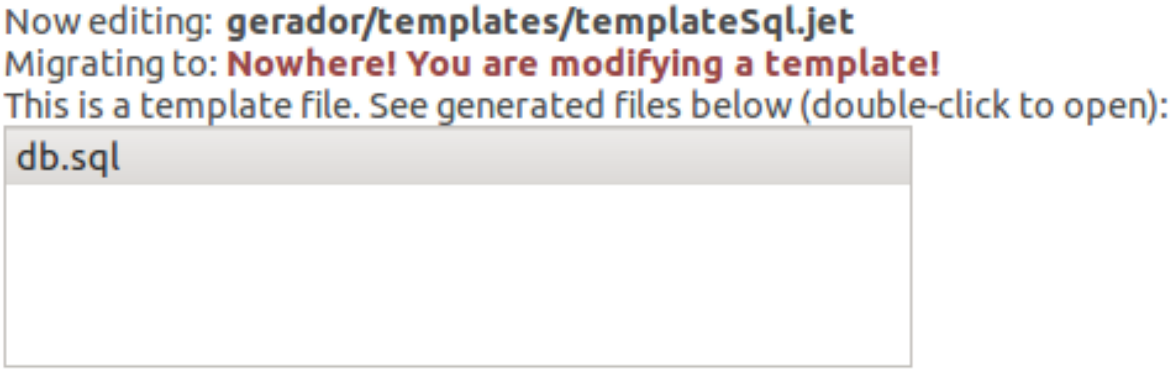


Figura 43: Exemplo de indicação do arquivo da IR a ser editado ao invés do *template Split JET Editor*.

Caso o status mude para "Editando Região de Modelo", deve-se Tentar modificar o Modelo de acordo com as necessidades da tarefa e seguindo as definições do Metamodelo, pois não se deve alterar uma região do código-fonte da IR que é resultado de uma consulta

ao modelo, uma vez que a sincronia entre a IR e o modelo será perdida e na próxima execução da transformação, essa alteração será desfeita e será mantido o resultado da consulta ao modelo e, portanto, o *Split JET Editor* não permite a modificação desse trecho de código. Após a modificação do modelo, deve-se Tentar executar novamente a transformação, sendo que esse status acontece quando o desenvolvedor tenta alterar o nome de uma classe que é resultado de uma consulta ao modelo.

4.4 Ferramenta baseada no Quadro Kanban

Esta ferramenta foi idealizada com base no Quadro *Kanban* que é comumente utilizado no método *Kanban*. O referido quadro tem o intuito de facilitar, de forma visual, o controle das tarefas, pois nele é possível dividir as tarefas a serem feitas, a serem feitas no dia de hoje, em andamento e concluídas. Além de permitir a marcação do tempo de execução esperado e gasto e o progresso. Dessa forma, ela facilita a detecção de gargalos e a identificação da próxima tarefa que deve ser desenvolvida (FITZGERALD, MUSIAL e STOL 2014).

Assim sendo, a ferramenta proposta foi elaborada como um sistema *Web*[6] e sua página principal será um Quadro, adaptação do Quadro *Kanban*, conforme a Figura 44. Ele é dividido em quatro colunas: *To-Do*, *Do Today*, *In progress* e *Done*. A coluna *To-Do* é composta pelas tarefas que serão desenvolvidas. Já a coluna *Do Today* contém as tarefas que serão desenvolvidas no dia. A coluna *In progress* é constituída pelas tarefas que estão sendo desenvolvidas. Por fim, a coluna *Done* é formada pelas tarefas finalizadas.

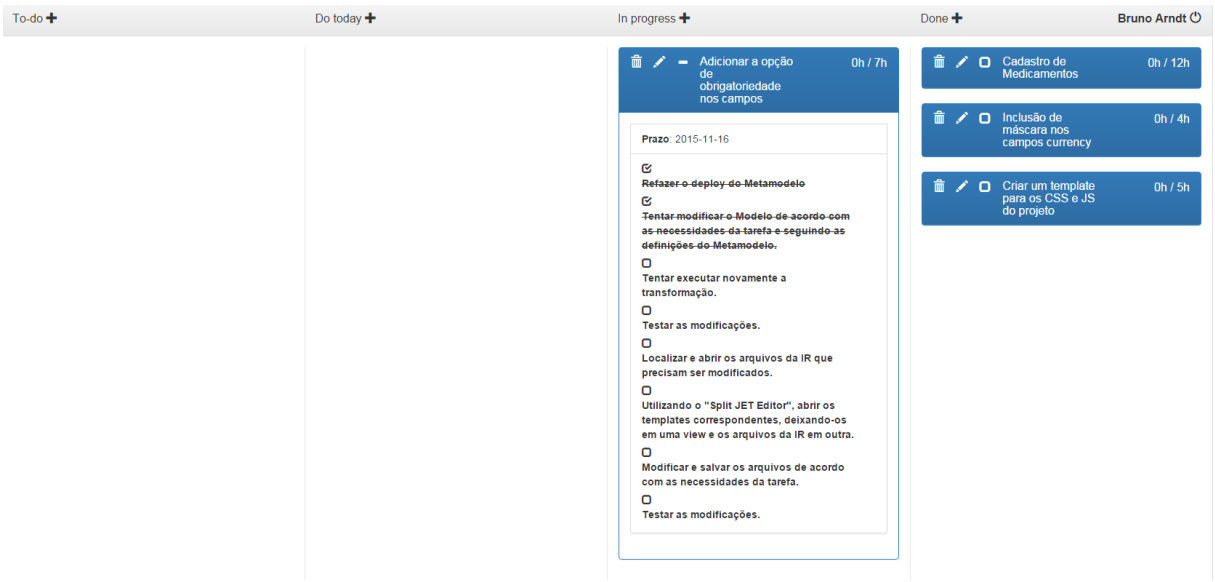


Figura 44: Interface da Ferramenta Kanban.

6 Link para a ferramenta Kanban - http://goo.gl/RaQy2U

Cada coluna do Quadro tem a possibilidade de adicionar uma tarefa. Ao clicar no botão adicionar, é aberto um diálogo composto por um *form* que contém os seguintes campos, conforme a Figura 45: Nome da Tarefa, Descrição, Tempo Estimado, Tempo Gasto e Prazo de Entrega. Além desses campos, o *form* de Adicionar Tarefa contém quatro questões sobre a nova tarefa, que determinará quais passos devem ser seguidos para a conclusão da mesma, baseando-se no método MME-MDD.

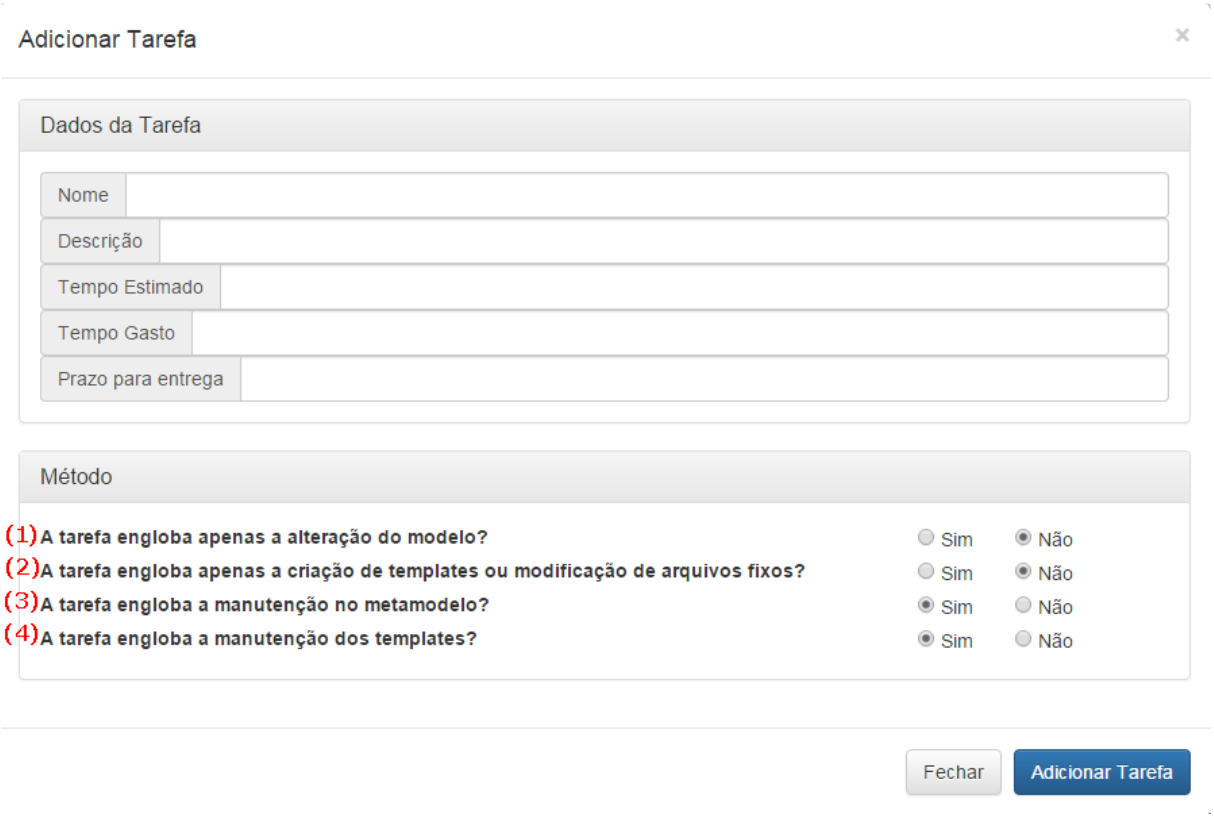


Figura 45: Adicionar uma Tarefa na Ferramenta Kanban.

Caso a **tarefa engloba apenas a alteração no modelo** (Figura 45-1), ela será do tipo MDD e seguirá os passos que estão descritos na Seção 4.3.1. Caso a **tarefa engloba apenas a criação de *templates* ou modificação de arquivos fixos** (Figura 45-2), ela será do tipo Criativa e seguirá os passos que estão descritos na Seção 4.3.2.

Caso contrário, a tarefa será do tipo Manutenção e Evolução, podendo englobar os sub-passos da edição do metamodelo (Seção 4.3.3.1 e Figura 45-3) e podendo incluir os sub-passos da edição do código-fonte da IR (Seção 4.3.3.2 e Figura 45-4), sendo que a tarefa deve abranger um dos dois sub-passos. Um exemplo de tarefa criada nesta ferramenta está apresentado na Figura 46.

Além de adicionar uma tarefa, a ferramenta Kanban permite que se mova as tarefas entre as colunas e que se re-ordene as tarefas em uma mesma coluna. Além disso, pode-se editar, minimizar/maximizar e excluir uma tarefa. Há também a possibilidade de se

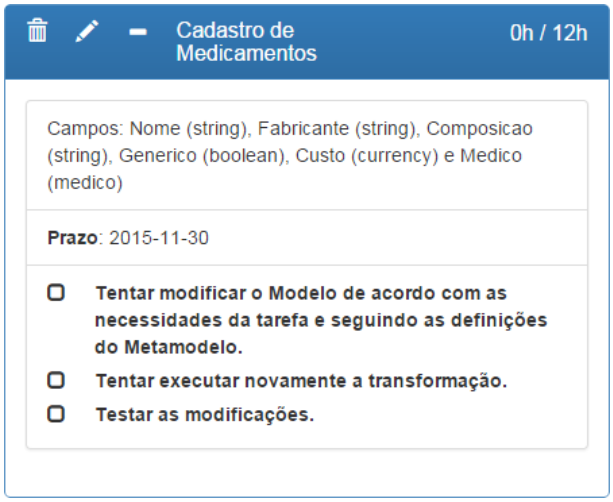


Figura 46: Exemplo de uma Tarefa na Ferramenta Kanban.

marcar ou desmarcar o progresso na tarefa, clicando no passo concluído. Quando a tarefa for finalizada, sugere-se que a mova para a coluna *Done*, pois essa medida facilitará o acompanhamento do progresso do projeto.

4.5 Tutorial

Com o intuito de facilitar a compreensão do método e das ferramentas de Possatto (2013) e Perini (2015) foi elaborado um *cyber tutorial*[7], que contém a descrição, um passo-a-passo para e vídeos demonstrativos para a utilização dos mesmos. O *cyber tutorial* é um portal *Web* que tem as seguintes seções, conforme apresentado na Figura 47: Introdução ao MDD, Ferramenta Possatto, Ferramenta Perini, Método e *Downloads*.

A seção de Introdução ao MDD contém a descrição dos conceitos englobados por este trabalho, sendo assim a apresentação dos conceitos do MDD, Transformação de Modelo e Geração de código baseado em *templates*. Além disso, todos os trabalhos utilizados na descrição dos referidos conceitos estão com um *link* para os mesmos, facilitando o acesso aos mesmos.

Na seção Ferramenta Possato, há a apresentação do protótipo desenvolvido por Possatto (2013) e, também, é apresentado o arquivo de mapeamento utilizado para a sincronização entre a IR e os *templates*, bem como, está descrito o experimento realizado pelo mesmo.

Há a apresentação do *Split JET Editor* na seção Ferramenta Perini. Além disso, há um passo-a-passo de como utilizar e como configurar a mesma. Também há uma explicação detalhada de cada status do *Split JET Editor*.

[7] Link para o *cyber tutorial* - http://goo.gl/7PQKCU

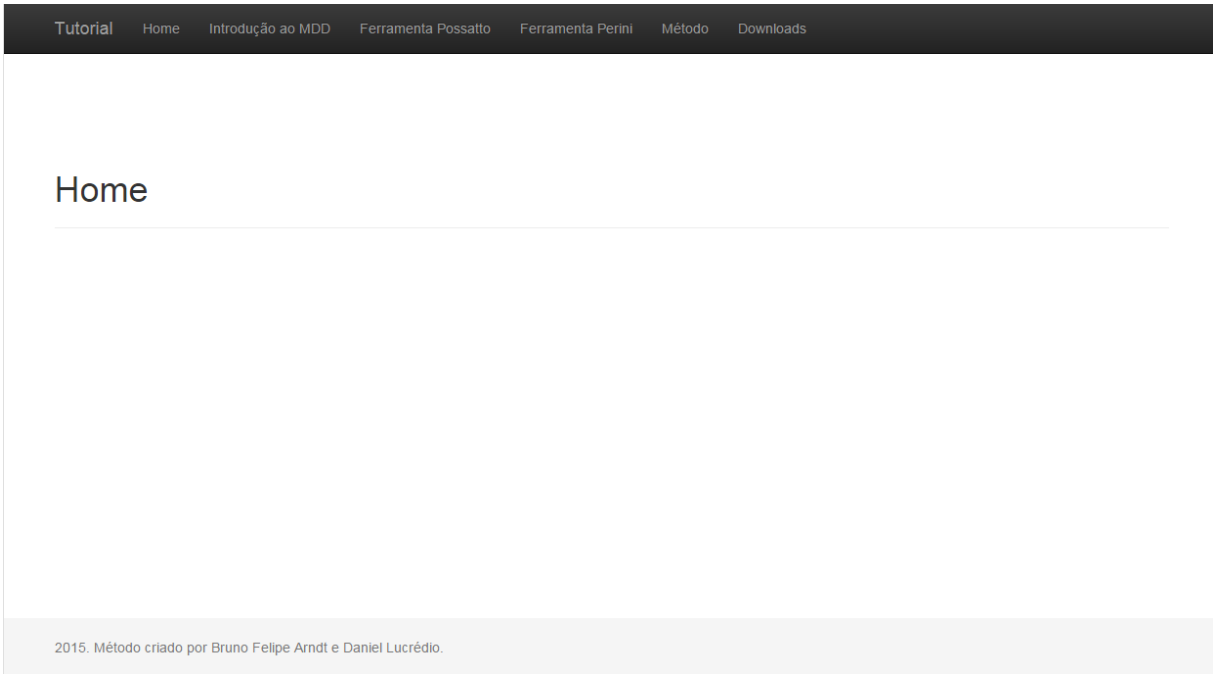


Figura 47: Interface do Tutorial.

Apresenta-se o método proposto na seção Método, incluindo a descrição dos tipos de tarefas. Também é descrito todos os quatro níveis de abstração que estão presentes no ambiente de desenvolvimento que o método proposto está inserido. Além de tudo, há a descrição da ferramenta Kanban e um passo-a-passo de como utilizá-la. Todo o conteúdo descrito nessa seção, está presente em um vídeo introdutório ao método, e um vídeo explicativo de cada tipo de tarefa.

A última seção é a de *Downloads* que contém o link para o acesso do GitHub do método. Esse GitHub contém a ferramenta de Possatto (2013) e Perini (2015), a ferramenta Kanban e o código-fonte do Tutorial.

4.6 Considerações Finais

Este Capitulo apresentou o método MME-MDD, que propõe uma série de passos para a manutenção e a evolução de sistemas baseados em MDD. O método norteia o desenvolvedor na utilização das ferramentas de Possatto (2013) e de Perini (2015), maximizando a sua utilização. Ele é composto por uma ferramenta, baseada no quadro *Kanban*, que tem como objetivo auxiliar o desenvolvedor no processo de desenvolvimento, e um *cyber tutorial* com a finalidade de nortear o desenvolvedor na utilização do método proposto.

Além do enfoque na redução do tempo gasto na manutenção e na evolução de sistemas baseados em MDD, o método traz uma série de facilidades para o desenvolvedor.

A primeira facilidade é em relação a facilitar o aprendizado nesse tipo de ambiente, pois o método é um passo-a-passo que apresenta as etapas possíveis para a realização de cada tarefa, de acordo com o seu tipo e requisitos. Além disso, há o *cyber tutorial* que contém a descrição de cada tipo de tarefa e suas atividades necessárias e um vídeo que demonstra na prática a utilização do método e das ferramentas de Possatto (2013) e de Perini (2015).

Outra facilidade que há na utilização do MME-MDD é na utilização das ferramentas de Possatto (2013) e de Perini (2015), pois o mesmo descreve as etapas que precisam ser realizadas para a correta utilização das mesmas e quando há a alteração de status, o método mostra os passos necessários para que o erro seja concertado e o status volte ao padrão.

Entretanto, o MME-MDD possui uma grande dependência das ferramentas de Possatto (2013), de Perini (2015) e do Eclipse, pois os sub-passos de Refazer o *deploy* do Metamodelo e Editar o código-fonte da IR dependem totalmente nas ferramentas anteriormente citadas, sendo que seria necessário fazer uma adaptação desses sub-passos para a utilização de outras ferramentes e outros ambientes.

5 Estudo de Caso - Evolução de um sistema para controle de pacientes

Wohlin *et al.* (2000) definem que um estudo de caso deve ser realizado com o objetivo de investigar uma entidade, um objeto ou um fenômeno em um específico espaço de tempo e, dessa forma, evitar o problema de escalabilidade. Sendo assim, ele é muito utilizado para a avaliação de métodos e ferramentas em engenharia de software. Outro tipo de estudo que também é amplamente usado é a experimentação.

A diferença entre um estudo de caso e a experimentação é que o primeiro é bem mais simples de se planejar, tem mais facilidades em sua execução e representar uma situação típica no desenvolvimento de sistemas, contudo seus resultados são difíceis de generalizar e interpretar. Dessa maneira, seus resultados apresentam os efeitos dos métodos e/ou ferramentas testados em situações típicas, mas não podem ser generalizados a todas as situações (WOHLIN et al. 2000).

Conforme o exposto, um estudo de caso, utilizando o método MME-MDD na manutenção e evolução de um sistema, foi realizado com o intuito de comparar o mesmo com uma abordagem manual em situações típicas de manutenção e evolução de sistemas de software.

O sistema utilizado neste estudo está descrito na Seção 5.1. Já na Seção 5.2 é apresentado o planejamento deste estudo, na Seção 5.3 é exposto os resultados obtidos neste estudo de caso e as ameaças à validade deste estudo estão na Seção 5.4. Por fim, as considerações finais estão na seção 5.5.

5.1 Descrição do Sistema

O sistema utilizado neste estudo de caso foi desenvolvido pelo pesquisador especificamente para este estudo e para a experimentação deste trabalho (Capítulo 6), sendo que o mesmo representa um sistema de pequeno porte para o controle de pacientes de uma clínica da área da saúde. Para a criação desse sistema, foi utilizado o mesmo ambiente de desenvolvimento descrito na Seção 4.1, a linguagem de programação PHP e os *frameworks* Bootstrap[1] e jQuery[2].

Esse sistema de controle de pacientes engloba um cadastro de pacientes e médicos. O cadastro de pacientes é composto por nome, número do Cadastro de Pessoa Física

[1] Bootstrap - http://getbootstrap.com/
[2] jQuery - https://jquery.com/

(CPF), data de nascimento, endereço, cidade e estado. Já o cadastro de médicos conta apenas com o nome e o número do Conselho Regional de Medicina (CRM) do médico.

5.2 Planejamento do Estudo

Este estudo de caso tem como objetivo verificar se em determinadas tarefas há diminuição no tempo de realização da mesma e facilidade na utilização do método proposto para a produção do código-fonte. Para tal, foram elaboradas quatro tarefas, que estão descritas na Tabela 8, sendo uma para o tipo MDD, uma para o tipo Criativa, uma para o tipo Manutenção sem o *deploy* do Metamodelo e, por fim, uma para o tipo Manutenção com o *deploy* do Metamodelo e Edição do código-fonte da IR.

Para a análise da complexidade da tarefa, foi utilizada a métrica de Complexidade ciclomática de McCabe (Anexo B.2), cujo resultado é baseado na quantidade de ramificações existentes no código-fonte, ou seja, a cada ocorrência de estruturas de controle é incrementado em um a medida dessa métrica.

ID	Descrição	Tipo	Diferença de tempo?	Complexidade Ciclomática de McCabe
T1	Adicionar um Cadastro de Medicamentos com os campos Nome (tipo *string*), Fabricante (tipo *string*), Composição (tipo *string*), Genérico (tipo *boolean*), Custo (tipo *currency*) e Médico (tipo Médico).	MDD	Sim	0
T2	Criar um *template* para cada CSS e JS do sistema.	Criativa	Não	0
T3	Colocar um total nos campos do tipo *currency*.	Manutenção sem o *deploy* do Metamodelo	Não	4
T4	Adicionar a opção de obrigatoriedade nos campos.	Manutenção com Metamodelo e Edição da IR	Sim	6

Tabela 8: Descrição das Tarefas do Estudo de Caso.

Nas tarefas T1 e T2 não se espera um ganho na utilização do método devido a sua baixa complexidade, zero de complexidade na métrica de McCabe (Anexo B.2), e, dessa forma, a utilização do MME-MDD não irá interferir na execução. Já as tarefas T3 e T4 tem uma complexidade média, quatro e seis de complexidade na métrica de McCabe (Anexo B.2), e, por esse motivo, esperou-se que a utilização do método traria ganho.

O participante deste estudo foi o próprio pesquisador com experiência no desenvolvimento de sistemas Web e com um ano de experiência no desenvolvimento de sistemas utilizando MDD.

A execução deste estudo de caso foi dividida em duas fases. A primeira fase foi a de planejamento, cujo objetivo foi à elaboração das tarefas. Já a segunda fase foi a de execução do estudo, que consistia na resolução das tarefas elaboradas na fase anterior, sendo que primeiramente foi executado o experimento utilizando o método proposto e, após isso, as mesmas tarefas foram executadas sem o método e sem qualquer ferramenta de auxílio. O método foi executado primeiro, pois não se desejava que o ganho de facilidade da segunda execução interferisse na avaliação do método e, dessa forma, preferiu-se que esse ganho ocorresse na abordagem manual.

5.3 Resultados

Os resultados deste estudo de caso estão na tabela 9, sendo que o tempo está no formato "hora:minuto:segundo". Cada tarefa que utilizou o MME-MDD seguiu rigorosamente os passos estabelecidos pelo mesmo. Já nas tarefas que foram executadas sem nenhum auxílio, seguiram os passos padrões na manutenção e evolução de sistemas utilizando a abordagem MDD, que consiste em, primeiramente, modificar o código-fonte da IR. Após isso, a modificação era migrada de forma manual para o *template* e, por fim, as TAGs do JET necessárias eram incluídas.

ID da Tarefa	Tempo Método	Tempo Manual	Diferença	Redução de tempo utilizando o Método
T1	00:02:05	00:02:27	-00:00:22	15%
T2	00:24:06	00:25:33	-00:01:27	6%
T3	00:19:56	00:28:34	-00:08:38	30%
T4	00:13:19	00:24:29	-00:11:10	46%

Tabela 9: Resultados do Estudo de Caso.

Conforme o exposto na Tabela 9, a execução com o método apresentou um menor custo de tempo de desenvolvimento do que a execução manual. Nas tarefas T1 e T2, onde não era esperado um ganho de tempo, devido a sua baixa complexidade, houve uma redução do tempo de apenas de 15% e 6% com a utilização do método. Já as tarefas T3 e T4, onde era esperado um ganho de tempo maior com a utilização do método, que tem uma complexidade média, houve uma redução do tempo de 30% e 46% e, dessa forma, a utilização do método trouxe um ganho maior do que a estimativa de 25% observada por Muszynski (2005).

Outro ponto a ser observado é que o tempo das tarefas manuais foram beneficiados, uma vez que o método foi executado primeiramente e, dessa forma, todos os erros que

ocorreram somente nele e, por esse motivo, o ganho da utilização do método seria maior.

Dessa forma, o método se mostrou efetivo nas situações testadas de manutenção e evolução de sistemas utilizando a abordagem MDD. Além disso, o método facilitou a execução das tarefas, pois norteou e listou cada passo que se fez necessário na execução das mesmas e, quando houve uma situação de um possível erro, o método apresentou os passos corretos a serem seguidos para a solução do mesmo.

5.4 Ameaças à Validade

Como o estudo foi executado pelo pesquisador que já possui certa experiência em desenvolvimento Web e utilizando MDD, os resultados obtidos podem não representar todas as situações na utilização do método, pois um desenvolvedor sem nenhuma experiência com essas abordagens pode ter uma maior dificuldade.

Outro ponto a ser considerado é de o pesquisador ter criado o sistema e, dessa forma, possuir um grande conhecimento sobre o mesmo e, por isso, os resultados obtidos podem, também, não representar as situações que o desenvolvedor não conheça o sistema.

Outro fator a ser considerado é a ordem de execução das abordagens, pois como são as mesmas tarefas para ambas as abordagens, a execução da segunda pode ser influenciada pela primeira, devido ao conhecimento dos erros já encontrados e de como realizar a tarefa e, por esse motivo, a execução da abordagem manual pode ter sido favorecida.

Além do exposto, o estudo foi realizado com tarefas de baixa e média complexidade e com um pequeno tempo de execução e, assim, os resultados podem variar quando aplicados as tarefas mais complexas.

5.5 Considerações Finais

Neste capítulo foi descrito um estudo de caso que teve como objetivo a comparação entre o método proposto nesta dissertação e uma abordagem manual. Dessa forma, foram elaboradas quatro tarefas para a manutenção e a evolução de um sistema já existente que utiliza a abordagem MDD.

Após a execução do estudo, os resultados demonstraram que o método reduz o tempo de desenvolvimento nas tarefas que englobam as tarefas do tipo Manutenção, devido a sua complexidade e a utilização do *Split JET Editor* que migra automaticamente as alterações da IR para os *templates*. Além disso, o método facilitou a execução de todas as tarefas, pois ele lista e descreve cada passo necessário para a execução das mesmas.

Porém, faz-se necessário a execução de outros estudos, tanto estudos de caso como experimentos, para que haja um maior conhecimento sobre o funcionamento do método em

todas as situações que são englobadas pela manutenção e evolução de sistemas utilizando MDD. Sendo assim, foi realizado um experimento que está descrito no Capítulo 6.

6 Experimentação da Proposta

Neste Capitulo é apresentado um estudo quantitativo, baseado na metodologia experimental (WOHLIN et al. 2000). Este estudo teve como intuito avaliar o impacto da utilização do método MME-MDD nas tarefas de manutenção e evolução de sistemas que utilizam a abordagem MDD. Para tal, foi utilizado a abordagem manual para comparação com o método proposto, sendo que foi criado dois grupos, o primeiro utilizando o método e as ferramentas de Possatto (2013) e Perini (2015), já o segundo não teve apoio de nenhuma ferramenta ou método.

Durante o experimento, os participantes tiveram que realizar 5 tarefas que englobavam os 3 tipos de tarefas e com diferentes níveis de complexidade, sendo que o tempo de desenvolvimento foi cronometrado. Após isso, eles tiveram que responder o Protocolo de simulação das tarefas (Apêndice C) e o Questionário de Opinião sobre o Método e Tutorial Proposto (Apêndice D). Ao final de cada tarefa, foi solicitado que os participantes enviassem seu código-fonte, para que o mesmo fosse analisado com a métrica *Lines of Code* (LOC) (Anexo B.1).

Este Capítulo está organizado da seguinte forma: a Seção 6.1 introduz alguns conceitos sobre os princípios da Experimentação, a Seção 6.2 que descreve a metodologia para a experimentação do método proposto. Já a Seção 6.3 apresenta os resultados deste experimento, as ameaças à validade deste estudo estão na Seção 6.4 e, por fim, o capitulo apresenta algumas considerações finais na Seção 6.5.

6.1 Princípios da Experimentação

Encontra-se descrito na literatura quatro métodos científicos relevantes para a condução de experimentos na área de Engenharia de Software: científico, de engenharia, experimental e analítico (WOHLIN et al. 2000; TRAVASSOS, GUROV e AMARAL 2002).

Na Engenharia de Software, a experimentação possui uma grande importância, em virtude do seu caráter de verificar teorias, de explorar os fatores críticos e dar indícios de melhoria nas novas teorias. A experimentação oferece um modo sistemático, disciplinado, computável e controlado para a avaliação de novos métodos, técnicas, linguagens e ferramentas. Dessa forma, é possível identificar a aplicabilidade prática do objeto de estudo (WOHLIN et al. 2000; TRAVASSOS, GUROV e AMARAL 2002; PAPOTTI 2013).

Sendo assim, esta experimentação teve como objetivo avaliar o método proposto em diferentes situações, podendo então verificar a eficácia do mesmo.

6.2 Metodologia do estudo experimental

A experimentação do método proposto seguiu as fases experimentais, como proposto por Wohlin *et al.* (2000) e foi planejada para ser executada no segundo semestre de 2015. O experimento constituiu de uma análise comparativa da qualidade do código, utilizando a métrica *Lines of Code* (LOC), sendo que a mesma foi mensurada pelo pesquisador e somente foram contadas as linhas modificadas pelos participantes, e do tempo gasto na manutenção e na evolução de tarefas entre a abordagem manual e a utilização do método proposto.

Na realização do experimento, foi utilizado o mesmo sistema do Estudo de Caso (Capítulo 5), que representa um sistema de pequeno porte para o controle de pacientes de uma clínica da área da saúde.

Dos dez participantes que tinham se comprometido a realizar o experimento, apenas quatro o realizaram, sendo que os participantes foram submetidos a cinco tarefas (Apêndice B). A primeira tarefa foi do tipo MDD (Seção B.1) e tem zero de complexidade seguindo a métrica de McCabe (Anexo B.2). Já a segunda tarefa foi do tipo Criativa (Seção B.2), com complexidade zero. A terceira tarefa foi do tipo Manutenção somente com a edição do código-fonte da IR (Seção B.3), com uma complexidade de seis. A quarta tarefa foi do tipo Manutenção somente com alteração do metamodelo (Seção B.4), com complexidade de zero. Por fim, a quinta tarefa foi do tipo Manutenção com edição da IR e do metamodelo (Seção B.5), com complexidade de oito. Para a realização das tarefas, foram disponibilizado vídeos e textos explicando o ambiente e o método para os participantes que utilizaram o método, já para os participantes que utilizaram a abordagem manual foram disponibilizados textos explicando somente o ambiente.

Para a execução do experimento, foi permitido que os participantes o realizassem em seus próprios computadores e no horário que desejassem, sendo que caso o participante não utilizasse em seu computador um sistema operacional Linux, foi disponibilizado uma máquina virtual totalmente configurada e operacional para os mesmos.

O experimento proposto foi definido com o seguinte objetivo:

- Analisar a utilização do método proposto em tarefas de manutenção/evolução;
- Com propósito de avaliar o método proposto no apoio ao desenvolvedor na realização das tarefas de manutenção e de evolução de sistemas utilizando a abordagem MDD;
- Com respeito à eficiência em termo de tempo despendido e qualidade de código-fonte gerado;
- Do ponto de vista de desenvolvedores de software; e

- No contexto de estudantes de graduação e pós-graduação em Ciência e Engenharia da Computação.

Foram planejadas as seguintes etapas durante o experimento:

6.2.1 Seleção do Contexto

Selecionar a população e amostra que fará parte deste estudo, fazendo também o seu recrutamento.

6.2.2 Formulação das Hipóteses

Visando determinar o efeito da utilização do referido método no resultado obtido, foram elaboradas seis hipóteses para o experimento, sendo divididas em dois grupos: hipóteses relativas ao tempo gasto (três hipóteses) e hipóteses relativas à qualidade do código-fonte gerado (três hipóteses). Para a formulação das hipóteses, foram consideradas as seguintes métricas:

T = tempo gasto para a execução da tarefa

mT = média do tempo gasto por todas as equipes para a execução da tarefa

A = Coeficiente de qualidade do código-fonte gerado

mCQ = média do coeficiente de qualidade do código-fonte gerado desenvolvido por todas as equipes

As hipóteses formuladas para o experimento, em relação ao tempo gasto, são:

Hipótese Nula (H0): Em geral, não há diferença entre equipes utilizando o método proposto em conjunto com as ferramentas de geração automática de códigos e o desenvolvimento manual, com respeito à eficiência (e) da equipe.

$$\mathbf{H0} : e_{\text{Método}} = e_{\text{Clássico}} \Big) mT_{\text{Método}} = mT_{\text{Clássico}} \tag{6.1}$$

Hipótese Alternativa (H1): Equipes utilizando o método proposto, em geral, são mais eficientes do que equipes que utilizam o desenvolvimento manual.

$$\mathbf{H1} : e_{\text{Método}} > e_{\text{Clássico}} \Big) mT_{\text{Método}} > mT_{\text{Clássico}} \tag{6.2}$$

Hipótese alternativa (H2): Equipes utilizando o método proposto, em geral, tem eficiência inferior a equipes que utilizam o desenvolvimento manual.

$$\mathbf{H2} : e_{\text{Método}} < e_{\text{Clássico}} \Big) mT_{\text{Método}} < mT_{\text{Clássico}} \tag{6.3}$$

Da mesma forma, as hipóteses formuladas para o experimento, em relação à qualidade do código-fonte gerado, que foi analisado utilizando a métrica LOC, são:

Hipótese Nula (H0): Em geral, não há diferença entre equipes utilizando o método proposto em conjunto com as ferramentas de geração automática de códigos e o desenvolvimento manual, com respeito à coeficiência de qualidade (a) do código-fonte gerado.

$$\mathbf{H0} : a_{\text{Método}} = a_{\text{Clássico}} \Big) mCQ_{\text{Método}} = mCQ_{\text{Clássico}} \tag{6.4}$$

Hipótese Alternativa (H1): Códigos-fonte gerados utilizando o método proposto, em geral, tem um coeficiente de qualidade maior do que equipes que utilizam o desenvolvimento manual.

$$\mathbf{H1} : a_{\text{Método}} > a_{\text{Clássico}} \Big) mCQ_{\text{Método}} > mCQ_{\text{Clássico}} \tag{6.5}$$

Hipótese alternativa (H4): Códigos-fonte gerados utilizando o método proposto, em geral, tem um coeficiente de qualidade inferior a equipes que utilizam o desenvolvimento manual.

$$\mathbf{H2} : a_{\text{Método}} < a_{\text{Clássico}} \Big) mCQ_{\text{Método}} < mCQ_{\text{Clássico}} \tag{6.6}$$

6.2.3 Seleção das Variáveis

Antes do início do projeto experimental, as variáveis dependentes e independentes foram escolhidas. Nessa fase, as principais variáveis foram escolhidas para a realização do experimento, a fim de analisar a eficiência do grupo usando diferentes abordagens de desenvolvimento. Dessa forma, a escolha da variável dependente é derivada diretamente das hipóteses formuladas. As variáveis foram escolhidas da seguinte forma:

- **Escolha das variáveis independentes**: As variáveis independentes que foram selecionadas na realização do experimento incluem a aplicação desenvolvida; o método proposto; o ambiente; e as tecnologias utilizadas. Dado que o intuito do experimento foi a investigação do impacto do uso de um método para a manutenção e evolução de sistemas na eficiência da equipe e na qualidade do código-fonte gerado, a variável

"método para migração automática de código" foi escolhida como fator que receberá tratamentos distintos. As demais variáveis independentes permaneceram constantes durante a execução do experimento:

- **Aplicação**: Ferramenta de Possatto (2013) e Perini (2015);
- **Ambiente de Desenvolvimento**: IDE Eclipse Luna (versão 3.7), Sistema Operacional Ubuntu, Banco de Dados PostgreSQL e Apache;
- **Tecnologias de Desenvolvimento**: PHP, JET, Xtext e *Eclipse Web Tools Platform*; e
- **Escolha das variáveis dependentes**: A eficiência da equipe e a qualidade do código-fonte gerado foram as variáveis dependentes selecionadas para a execução do experimento.

6.2.4 Seleção dos participantes

A seleção dos participantes do experimento foi realizada segundo a **técnica de amostragem não-probabilística por conveniência** (WOHLIN et al. 2000), de modo que os participantes selecionados foram os mais próximos e mais convenientes para realizar o experimento.

O experimento proposto seguiu o princípio geral de projeto de agrupamento dos participantes **em blocos** (WOHLIN et al. 2000) homogêneos, de modo que o fator nível de experiência dos participantes não teve impacto direto nos resultados dos tratamentos do fator processo de desenvolvimento, visando aumentar a precisão do experimento. Os participantes foram divididos em grupos homogêneos de acordo com o seu nível de experiência. Para determinar o nível de experiência de cada participante, foi utilizado um Formulário de Caracterização dos Participantes (Apêndice A), de modo que os participantes responderam questões de múltipla escolha, a respeito do próprio conhecimento sobre assuntos abordados no experimento (e.g. conhecimentos sobre geração automática de códigos, MDD e *templates*) e, dessa forma, foi possível quantificar o nível de experiência de cada participante relacionado às tecnologias utilizadas.

6.2.5 Instrumentação

Todo material necessário para apoiar os participantes ao longo do experimento foi previamente planejado, incluindo a preparação de objetos, diretrizes e instrumentos de coleta de dados que foram utilizados durante a execução do experimento.

Além da experimentação do método propriamente dita, os participantes do experimento responderam um questionário de opinião sobre o método e o tutorial (Apêndice

D), e a partir destas respostas o pesquisador irá verificar o nível de satisfação dos usuários. Esse questionário foi adaptado do questionário produzido por Papotti (2013) em seu estudo sobre a elaboração de um processo dirigido a modelos.

6.3 Resultados

Esse experimento contou com a participação de quatro pessoas, cujas experiências estão descritas na Tabela 10, já os conhecimentos dos participantes, em relação aos assuntos relacionados a este trabalho, estão descritas na Tabela 11. Para a tabela das experiências dos participantes, há as seguintes opções de respostas:

0 = Nenhum;

1 = Estudei em aula ou em livro;

2 = Pratiquei em projetos em sala de aula;

3 = Já utilizei em projetos pessoais; e

4 = Utilizo em grande parte dos projetos que realizo.

		Grau de Experiência				
ID	Titulação	MDD	*Templates* para geração de código	Geração automática de código	IDE Eclipse	Aplicação Web
P1	Mestrando	2	1	2	4	2
P2	Mestrando	1	2	1	3	3
P3	Mestrando	3	3	3	3	4
P4	Mestrando	3	3	3	3	4

Tabela 10: Nível de Experiência dos participantes.

Após o preenchimento do questionário, os participantes foram classificados em dois grupos, sendo um grupo com a utilização do método proposto (Grupo Método ou GMe) e outro com a utilização de uma abordagem manual de desenvolvimento (Grupo Manual ou GMa). Além disso, o pesquisador tentou deixar os grupos o mais homogêneo possível e, dessa forma, os participantes P1 e P3 ficaram no GMa e os participantes P2 e P4 ficaram no GMe.

No site do Tutorial, foi incluído uma opção no menu para o acesso ao experimento[1]. Contudo, para que os participantes do GMa não tivessem acesso as informações do método e, dessa forma, a validade deste experimento fosse comprometida, foi criado

[1] Link do experimento (Grupo Método) - http://goo.gl/4kCLpD

Assunto	P1	P2	P3	P4
Desenvolvimento Dirigido a Modelos	Básico	Básico	Avançado	Avançado
Desenvolvimento de Software para Web	Nenhum	Básico	Especialista	Especialista
Níveis de Abstração do MDD	Básico	Básico	Especialista	Especialista
Desenvolvimento de *plug-ins* para Eclipse	Básico	Nenhum	Nenhum	Básico
Domain-Specific Language (DSL)	Nenhum	Nenhum	Médio	Médio
Framework Xtext	Nenhum	Nenhum	Médio	Médio
Modelagem de Software	Básico	Básico	Médio	Médio
Templates	Nenhum	Básico	Médio	Avançado
Java Emitter Templates (JET)	Nenhum	Nenhum	Médio	Médio

Tabela 11: Nível de Experiência dos participantes.

um link especial[2] para o acesso ao site do Tutorial, sendo que as informações a respeito das ferramentas de Possatto (2013) e Perini (2015) e do método proposto foram ocultas.

Com o envio do link para os participantes, o pesquisador não teve mais nenhuma interferência no experimento, ou seja, o pesquisador não auxiliou de nenhuma forma os participantes, sendo assim o experimento pode comparar as abordagens em uma situação que os participantes não conheciam o sistema.

Foi concedido aos participantes dois meses para a execução do experimento e, após esse tempo, foram coletados os seguintes dados que estão na Tabela 12. Com base no código-fonte enviado pelos participantes, foi validado se eles executaram corretamente cada tarefa e, após isso, foi calculado o LOC resultante de cada uma, comparando o código-fonte original da aplicação e o código-fonte enviado pelos participantes. Porém, na T5 o participante P1 não o executou totalmente a tarefa e, por isso, seu LOC não foi considerado.

A Tabela 13 apresenta a comparação do tempo de execução entre os grupos. Apesar da grande diferença de tempo entre os participantes de mesmo grupo, o método se mostrou eficaz na maioria das tarefas (T1, T2 e T5), com tempo semelhante em uma tarefa (T3) e bem menos eficaz em uma tarefa (T4). Segundo o questionário respondido pelo participante P2, o mesmo teve dificuldade com a linguagem Xtext e, por isso, demorou tanto na execução da mesma.

Já em relação à qualidade do código-fonte produzido, conforme a Tabela 14, não houve uma grande diferença no código-fonte criado entre os integrantes de mesmo grupo, sendo que o método proposto se mostrou com produção de um código-fonte de melhor qualidade em quatro tarefas (T2, T3, T4 e T5) e, semelhante, em uma tarefa (T1). Como foi utilizado a métrica LOC para mensurar a qualidade do código-fonte produzido, os

[2] Link do experimento (Grupo Manual) - http://goo.gl/IUattt

Tarefa	Participante	Grupo	Tempo	LOC
T1	P1	GMa	00:19:35	20
	P2	GMe	00:15:00	21
	P3	GMa	00:22:25	22
	P4	GMe	00:01:43	21
T2	P1	GMa	01:08:18	66
	P2	GMe	00:20:00	24
	P3	GMa	00:08:07	46
	P4	GMe	00:02:22	22
T3	P1	GMa	00:10:05	131
	P2	GMe	00:20:00	82
	P3	GMa	00:12:12	112
	P4	GMe	00:03:16	90
T4	P1	GMa	00:35:18	9
	P2	GMe	01:40:00	1
	P3	GMa	00:14:35	1
	P4	GMe	00:12:26	1
T5	P1	GMa	13:06:33	55*
	P2	GMe	06:59:35	177
	P3	GMa	03:39:37	232
	P4	GMe	00:55:51	167

Tabela 12: Resultados do experimento.

Tarefa	Média Tempo GMa	Média Tempo GMe	Diferença	Redução de Tempo
T1	00:21:00	00:08:21	-00:12:38	60,20%
T2	00:38:13	00:11:11	-00:27:02	70,73%
T3	00:11:08	00:11:38	00:00:29	-4,41%
T4	00:24:56	00:56:13	00:31:16	-125,39%
T5	08:23:05	03:57:43	-04:25:22	52,75%

Tabela 13: Comparação entre o resultado dos grupos.

participantes que utilizaram o MME-MDD produziram um código-fonte com uma maior facilidade de manutenção, uma vez que, um código-fonte com uma menor quantidade de linhas de código é mais fácil de manter do que um com uma maior quantidade.

Tarefa	Média LOC GMa	Média LOC GMe	Diferença	Redução de LOC
T1	21,00	21,00	0,00	0,00%
T2	56,00	23,00	-33,00	58,93%
T3	121,50	86,50	-33,50	29,22%
T4	5,00	1,00	4,00	80,00%
T5	232,00	172,00	-60,00	25,86%

Tabela 14: Comparação entre o resultado dos grupos.

Se compararmos entre os pares P1 com P2 (Tabela 15) e P3 com P4 (Tabela 16) é possível ver os benefícios da utilização do método proposto. Na comparação entre o par

P1 com P2, que representam desenvolvedores sem experiência com essa abordagem, há a redução do tempo de realização da tarefa na maioria delas (T1, T2 e T5) e melhora na qualidade do código-fonte em quase todas as tarefas (T2, T3 e T4), sendo que a T5 do participante P1 foi desconsiderada devido à realização incorreta da mesma.

Tarefa	Tipo	P1	P2	Diferença	Redução
T1	Tempo	00:19:35	00:15:00	-00:04:35	23,40%
	LOC	20	21	1	-5,00%
T2	Tempo	01:08:18	00:20:00	-00:48:18	70,72%
	LOC	66	24	-42	63,64%
T3	Tempo	00:10:05	00:20:00	00:09:55	-98,35%
	LOC	131	82	-49	37,40%
T4	Tempo	00:35:18	01:40:00	01:04:42	-183,29%
	LOC	9	1	-8	88,89%
T5	Tempo	13:06:33	06:59:35	-06:06:58	46,66%
	LOC	-	177	-	-

Tabela 15: Comparação entre os pares P1 e P2.

Já no par P3 e P4, que representa os desenvolvedores com maior experiência nessa abordagem, o método se mostrou mais eficiente em todas as tarefas e produziu um código-fonte melhor em quase todas elas (T2, T3 e T5).

Tarefa	Tipo	P3	P4	Diferença	Redução
T1	Tempo	00:22:25	00:01:43	-00:20:42	92,34%
	LOC	22	21	-1	4,55%
T2	Tempo	00:08:07	00:02:22	-00:05:45	70,84%
	LOC	46	22	-24	52,17%
T3	Tempo	00:12:12	00:03:16	-00:08:56	-73,22%
	LOC	112	90	-22	19,64%
T4	Tempo	00:14:35	00:12:26	-00:02:09	14,74%
	LOC	1	1	0	0,00%
T5	Tempo	03:39:37	00:55:51	-02:43:49	74,57%
	LOC	232	167	-65	28,02%

Tabela 16: Comparação entre os pares P3 e P4.

Ao término das tarefas, os participantes que utilizaram o método foram questionados sobre sua avaliação do método, das ferramentas e do tutorial (Apêndice D). Já os participantes que utilizaram a abordagem manual foram questionados sobre sua avaliação do experimento (Apêndice E).

Primeiramente, os participantes P2 e P4 foram questionados sobre sua avaliação do método proposto, das ferramentas e do tutorial, conforme a Figura 48, sendo que os participantes acreditaram totalmente que os três os auxiliou nas tarefas de manutenção e evolução de sistemas, dado que em 70% das tarefas não encontraram dificuldades e, quando encontraram, foi de forma parcial, uma vez que o método e o tutorial se complementaram.

Além disso, os três receberam uma nota média de 9,7 e o grupo do método considera em utilizá-los em outros tipos de sistemas.

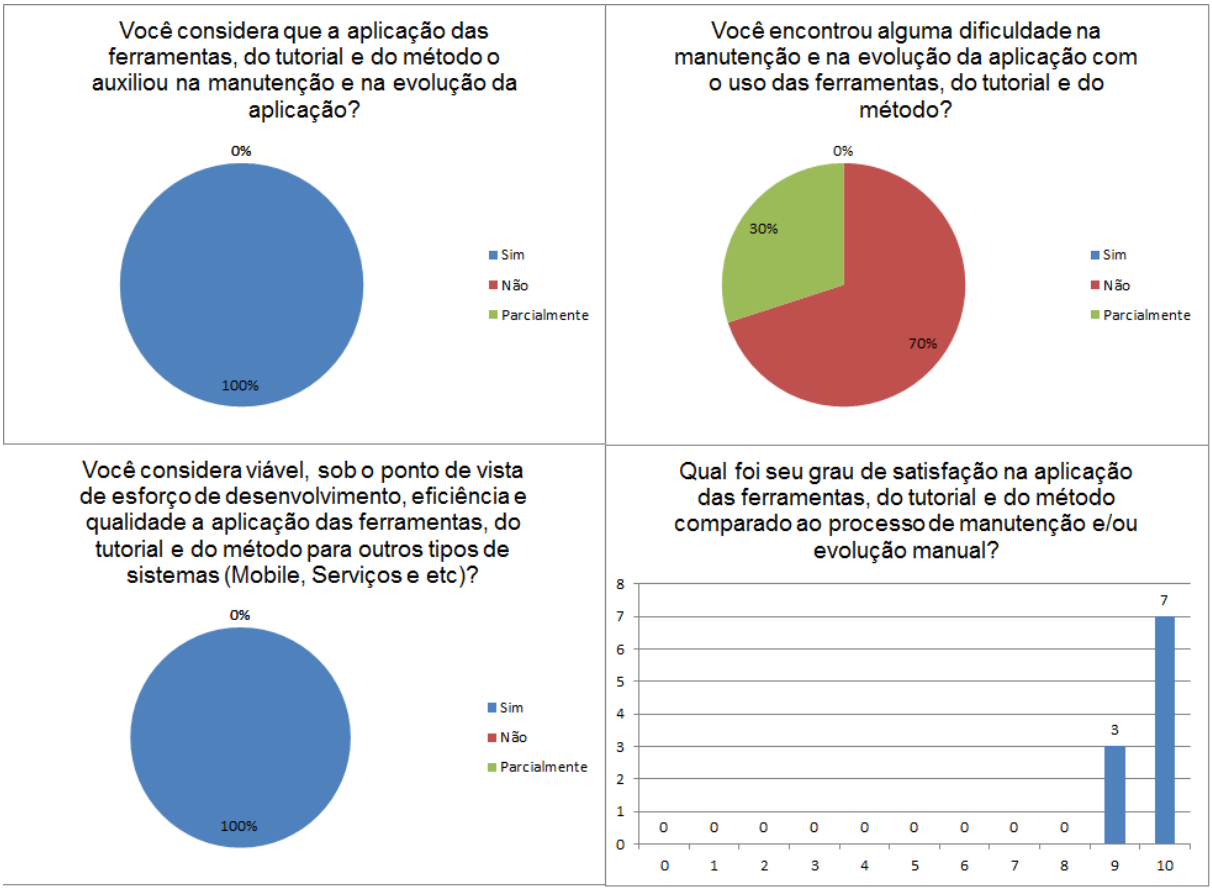


Figura 48: Avaliação do Método, das Ferramentas e do Tutorial.

Depois disso, os participantes foram questionados exclusivamente sobre o método, conforme a Figura 49. Apenas com a utilização do método, em 80% das tarefas foi encontrada uma dificuldade parcial na utilização do método, mostrando a importância do tutorial. Apesar dessa dificuldade, P2 e P4 acreditaram que o método os auxiliou nas tarefas de manutenção e evolução de sistemas, recebendo uma nota 9,9.

Por fim, o grupo do método avaliou o tutorial (Figura 50) e em todas as situações o mesmo facilitou a compreensão do método proposto, uma vez que os vídeos auxiliaram em quase todas as tarefas, contudo ainda há a necessidade de melhorá-los, principalmente na questão da qualidade de áudio e vídeo. Além disso, o site do tutorial se mostrou muito bem avaliado, tanto na facilidade de uso, da disposição das informações e compreensão do texto, quanto na qualidade das figuras utilizadas.

Já os participantes do grupo manual, avaliaram o experimento (Figura 51), onde eles encontraram dificuldade na execução das tarefas em 75% das situações, sendo que a dificuldade parcial foi de apenas em 25% das situações. Com isso, eles acreditam que em todas as situações, um método que guie o desenvolvedor nas tarefas de manutenção e

Figura 49: Avaliação do Método proposto.

evolução de sistemas facilitaria as mesmas (20% das vezes pensam que essa facilidade será parcial). Porém, os participantes P1 e P3 encontraram certa dificuldade no entendimento das tarefas, principalmente na tarefa T5 que foi considerada muito extensa.

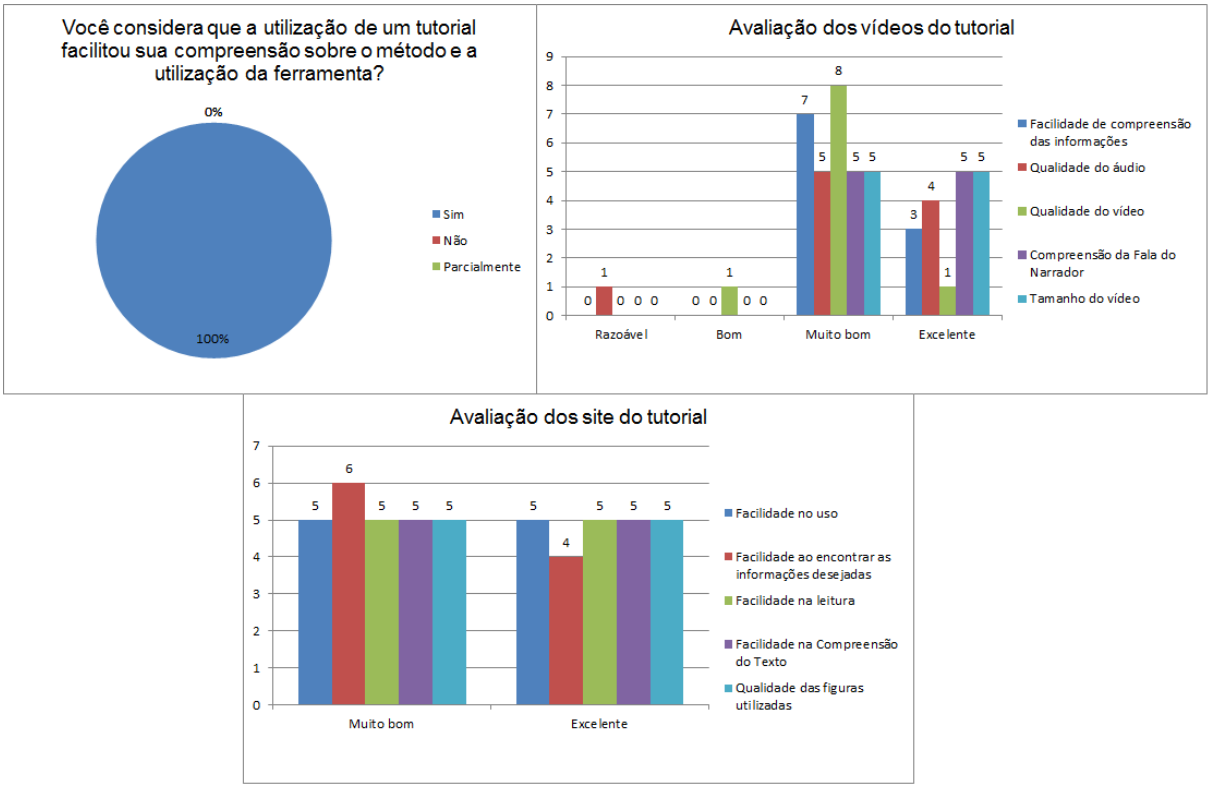


Figura 50: Avaliação do Tutorial.

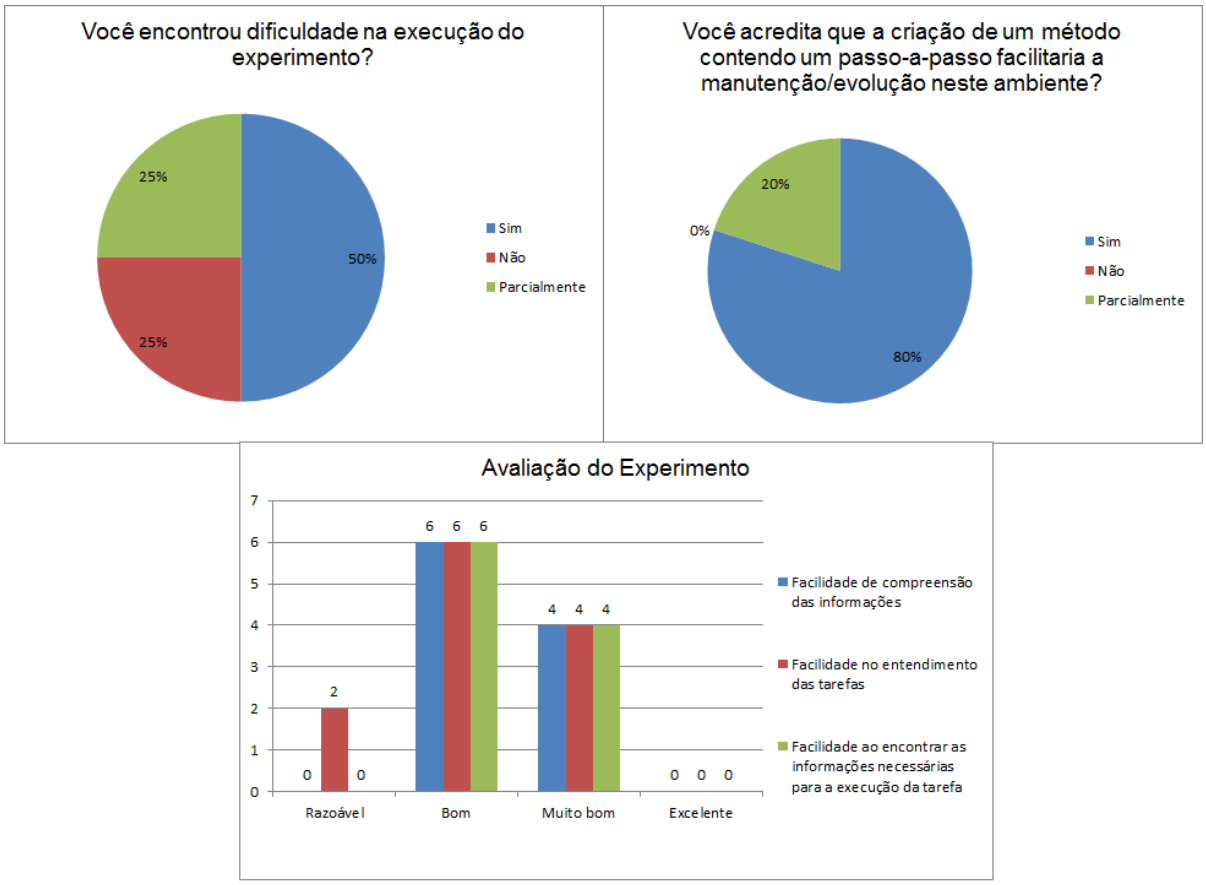


Figura 51: Avaliação do Experimento.

6.4 Ameaça à validade

Estudos experimentais muitas vezes possuem alguns vieses, que por sua vez podem ser considerados uma ameaça ao experimento. No referido caso podemos citar alguns desses fatores que não puderam ser controlados e consequentemente, podem ter influenciado nos resultados do estudo.

O primeiro desses fatores é o tamanho amostral, como o estudo foi executado por uma amostra pequena (n=4), os resultados obtidos podem não representar todas as situações na utilização do método, além disso a interpretação, especialmente estatística, dos resultados fica limitada, pois uma única resposta diferente pode influenciar nessa interpretação.

Outro ponto a ser considerado é a possível omissão de informações no questionário de Nível de Experiência, uma vez que o participante pode não querer colocar o seu real nível de conhecimento e experiência de determinado assunto ou pode haver disparidade na classificação dos mesmos, já que essa medida é subjetiva.

Além disso há a questão de controle do ambiente, uma vez que o estudo foi executado em um ambiente não controlado, pois os participantes tiveram a liberdade de executar o mesmo no horário e local que os participantes da pesquisa desejassem. Além disso, o experimento pode ter sido influenciado pela configuração da máquina em que foi executado, uma vez que os participantes utilizaram suas máquinas. Por fim, há a possibilidade de que um ou mais participantes possam não ter compreendido ou interpretado o enunciado das tarefas de forma correta, como foi observado por exemplo o P1 na T5.

Também há um aspecto a ser considerado na medição do fator tempo, visto que foram os participantes que cronometraram o tempo e, por esse motivo, há a possibilidade que um ou mais participantes possam ter começado a cronometrar o tempo no momento errado, ou seja, não começaram no início de cada tarefa, conforme descrito nos passos de cada tarefa.

Tais observações são uteis, especialmente para a replicação de estudos futuros, mas todas são compreensíveis e não afetam a confiabilidade do presente estudo. Contudo, os resultados devem ser interpretados e discutidos levando-se em consideração as situações acima descritas.

6.5 Considerações Finais

Este Capítulo apresentou a experimentação do método proposto que, apesar da amostra ser pequena, se mostrou válido e efetivo na sua utilização, tanto no fator tempo quanto no fator qualidade do código-fonte. O método trouxe um ganho considerável na qualidade do código-fonte e, em algumas situações, trouxe ganho no fator tempo, sendo

necessário testá-lo com uma amostra maior, para que se possa ter uma melhor medição no fator tempo

Conforme o Questionário de Opinião sobre o Método e Tutorial Proposto, os quais se complementam, contudo ainda há a necessidade de aperfeiçoá-los, principalmente em relação às modificações que englobam o metamodelo, já que desenvolvedores com pouca experiência neste tipo de modificação podem encontrar muita dificuldade.

Além disso, o questionário demonstrou que os vídeos auxiliaram bastante no entendimento do método, contudo eles ainda precisam de um aprimoramento na qualidade de áudio e vídeo, além de ser necessária a diminuição da velocidade do vídeo em alguns trechos, principalmente na explicação sobre o metamodelo.

Todos os dados coletados por este trabalho estão disponibilizados integralmente no *link* `http://goo.gl/7PQKCU` para consulta.

7 Considerações Finais

Apesar da grande evolução dos processos de engenharia de software para o desenvolvimento de sistemas baseado em modelos, ainda há falta de metodologias e métodos que guiem os desenvolvedores nas atividades de manutenção e evolução de sistemas. O método que foi proposto nesta dissertação vem de encontro com essa necessidade. Além de guiar o desenvolvedor nas referidas atividades, o método trouxe uma redução no tempo gasto e uma melhora na qualidade e na facilidade da execução de tarefas de manutenção e evolução, conforme verificado através de um estudo de caso (Capítulo 5) e uma experimentação (Capítulo 6).

Outro ponto abordado pelo método proposto foi a utilização das ferramentas propostas por Possatto (2013) e Perini (2015), sendo que o MME-MDD guia o desenvolvedor na utilização dessas ferramentas e quando deve utilizá-las, pois nem todas as situações necessitam delas, já que uma modificação pode englobar apenas um modelo, por exemplo. Sendo que os níveis de modelo e metamodelo ainda não tinham sido abordados pelos trabalhos anteriores a este.

Apesar do método cumprir seu objetivo, ele ainda precisa de algumas melhorias, já que os participantes da experimentação relataram algumas dificuldades e, dessa forma, o método ainda precisa de algumas correções e melhorias para que se possa aumentar sua eficácia. Além disso, os vídeos que estão presentes no *cyber-tutorial* precisam ser revisados e sua qualidade de áudio melhorada.

Outra necessidade identificada ao término desta pesquisa foi que devido a pequena amostra que realizou o experimento, se faz necessário a execução de uma nova experimentação com um número maior de participantes e com mais tarefas de cada tipo, para que se possa melhorar verificar o benefício da utilização do MME-MDD. Além disso, há a necessidade da utilização de mais grupos, para que se possa comparar o real ganho do método em relação das ferramentas propostas por Possatto (2013) e Perini (2015).

Por fim, com este projeto, acredita-se que foi realizada uma contribuição com uma pequena parte do processo de MDD, mais especificamente na manutenção e na evolução de sistemas, tornando este mais simples, claro e rápido, melhorando com isso o custo/benefício dessa abordagem de desenvolvimento.

7.1 Trabalhos Futuros

Apesar dos diversos pontos englobados durante o desenvolvimento deste trabalho, há algumas possibilidades de melhorias em relação a este estudo. Os trabalhos futuros

que podem contribuir para a evolução desta pesquisa, destacam-se:

- O aprimoramento deste método, principalmente para facilitar o entendimento do mesmo;
- A criação de uma Metodologia que englobe este método;
- O aprimoramento da ferramenta *Split JET Editor*, já que a mesma funciona apenas na perspectiva do Java no Eclipse e, dessa forma, limita as opções de utilização da mesma; e
- A criação de uma solução automatizada para a criação dos *templates*, já que há uma grande complexidade nesse passo.

Bibliografia

ACCELEO (2010). *MDA Generator*. URL: http://www.acceleo.org/ (ver p. 39).

AGNER, L. T. W. (2012). "PI-MT: Método para a Criação de Transformações de Modelos no Contexto da MDA". Tese de doutorado. Universidade Tecnológica Federal do Paraná (ver pp. 60, 84–86).

ALEGRIA, J. A. H. et al. (2011). "An MDE Approach to Software Process Tailoring". Em: *Proceedings of the 2011 International Conference on Software and Systems Process*, pp. 43–52 (ver pp. 60, 68, 69).

ANGYAL, L., L. LENGYEL e H. CHARAF (2008). "A Synchronizing Technique for Syntactic Model-Code Round-Trip Engineering". Em: *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pp. 463–472 (ver pp. 27, 34, 35, 37–39).

ANTKIEWICZ, M. e K. CZARNECKI (2006). "Framework-Specific Modeling Languages with Round-Trip Engineering". Em: *9th International Conference Model Driven Engineering Languages and Systems (MoDELS 2006)*, pp. 692–706 (ver p. 38).

ANTKIEWICZ, M., K. CZARNECKI e M. STEPHAN (2009). "Engineering of Framework-Specific Modeling Languages". Em: *IEEE Transactions on Software Engineering* 35.6, pp. 795–824 (ver p. 38).

ARCSTYLER (2010). *Model-centric software development (MDA/MDD)*. URL: http://www.interactive-objects.com/en/model-driven-engineering.html (ver p. 39).

ARNOLD, R. S. e S. A. BOHNER (1993). "Impact analysis-Towards a framework for comparison". Em: *Conference on Software Maintenance (CSM-93)*, pp. 292–301 (ver pp. 29, 48).

ASADI, M., N. ESFAHANI e R. RAMSIN (2010). "Process Patterns for MDA-Based Software Development". Em: *Eighth ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010)*, pp. 190–197 (ver pp. 44, 60, 68).

ASADI, M., M. RAVAKHAH e R. RAMSIN (2008). "An MDA-based system development lifecycle". Em: *Proceedings of 2nd Asia International Conference on Modelling and Simulation, AMS 2008*, pp. 836–842 (ver pp. 60, 77–80, 88).

BARRETO, A. S. (2011). "Uma Abordagem para Definição de Processos baseada em Reutilização Visando à Alta Maturidade em Processos". Tese de doutorado. Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia - Universidade Federal do Rio de Janeiro (ver pp. 45, 54).

BEYDEDA, S., M. BOOK e V. GRUHN (2005). *Model-Driven Software Development*. Springer (ver p. 36).

BIOLCHINI, J. et al. (2005). *Systematic Review in Software Engineering*. Rel. téc. Hewlett-Packard Company (ver p. 45).

BOHLEN, M. (2003). *AndroMDA*. URL: http://www.andromda.org/ (ver p. 39).

BOHNER, S. A. (2002). "Software change impacts-an evolving perspective". Em: *International Conference on Software Maintenance*, pp. 263–272 (ver pp. 29, 48).

BOOCOCK, P. (2003). *Jamda: The Java Model Driven Architecture*. URL: http://sourceforge.net/projects/jamda/ (ver p. 38).

BOUILLON, L. et al. (2005). "Reverse engineering of Web pages based on derivations and transformations". Em: *Proceedings of the Third Latin American Web Congress (LA-WEB 2005)* (ver p. 81).

BOURQUE, P. e R. E. FAIRLEY (2014). *SWEBOK Guide V3.0: Guide to the Software Engineering Body of Knowledge*. 3ª ed. IEEE Computer Society (ver p. 42).

BRAGA, R. T. V. (2002). "Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico". Tese de doutorado. Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (ver p. 37).

CHAN, A. (2008). "Peônia: Um ambiente web para apoiar processos de desenvolvimento com utilização de padrões de software e requisitos de teste no projeto de aplicações". Diss. de mestrado. Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (ver pp. 42, 43).

CHITFOROUSH, F., M. YAZDANDOOST e R. RAMSIN (2007). "Methodology Support for the Model Driven Architecture". Em: *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pp. 454–461 (ver pp. 37, 60, 66–68).

CIRILO, C. E. (2011). "Model Driven RICHUBI - Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 60, 87).

CLEAVELAND, J. C. (1988). "Building application generators". Em: *IEEE Software*, 25–33 (ver pp. 27–29, 38, 39, 41, 42).

Compuware (2006). *Model-Driven Development for Java*. URL: http://www.omg.org/mda/mda_files/MDA_OptimalJ.pdf (ver p. 37).

CORREA, C. K. F., T. C. OLIVEIRA e C. M. L. WERNER (2011). "An Analysis of Change Operations to Achieve Consistency in Model-Driven Software Product Lines". Em: *Proceedings of the 15th International Software Product Line Conference* 2 (ver p. 92).

COSTA, A., E. SALES e C.A.L. REIS (2007). "Apoio a Reutilização de Processos de Software através de Templates e Versões". Em: *VI Simpósio Brasileiro de Qualidade de Software*, pp. 47–61 (ver p. 44).

COSTA, T., F. GOMES e M. I. CAGNIN (2007). "Estudo para Adaptação de um Processo Ágil de Desenvolvimento baseado em Framework para apoiar o Desenvolvimento de Software baseado em Modelos". Em: *WDRA: Workshop em Desenvolvimento Rápido de Aplicações* (ver p. 36).

COYETTE, A. e J. VANDERDONCKT (2005). "A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces". Em: *Human-Computer Interaction - INTERACT 2005*, pp. 550–564 (ver p. 81).

CZARNECKI, K. e U. W. EISENECKER (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional (ver pp. 27–29, 37, 39, 41, 42).

CZARNECKI, K. e S. HELSEN (2003). "Classification of model transformation approaches". Em: *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, pp. 1–17 (ver pp. 27, 34, 35, 37, 39).

CÉRET, E., G. CALVARY e S. DUPUY-CHESSA (2011). "Flexibility in MDE for scaling up from simple applications to real case studies: illustration on a Nuclear Power Plant". Em: *Proceedings of the 25th Conference on l'Interaction Homme-Machine (IHM '13)*, pp. 33–42 (ver pp. 60, 81).

DEURSEN, A., P. KLINT e J. VISSER (2000). "Domain-Specific Languages: An Annotated Bibliography". Em: *ACM SIGPLAN Notices* 35.6, pp. 26–36 (ver p. 37).

FEILKAS, M. (2006). "How to represent models, languages and transformations?" Em: *6th OOPSLA Workshop on Domain-Specific Modeling*, 169–176 (ver pp. 27, 34, 38, 39).

FIEBER, F. e R. PETRASCH (2005). "Erweiterung des V-Modell XT-Eine Projektdurchführungsstrategie für die modellgetriebene Software-Entwicklung mit der MDA". Em: *GI Softwaretechnik-Trends* 25.3 (ver p. 77).

FITZGERALD, B., M. MUSIAL e K.-J. STOL (2014). "Evidence-Based Decision Making in Lean Software Project Management". Em: *ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 93–102 (ver pp. 30, 89, 104).

FRAKES, W. B. e S. ISODA (1994). "Success factors of systematic software reuse". Em: *IEEE Software* 11.1, 14–19 (ver p. 34).

FRANCE, R. e B. RUMPE (2007). "Model-driven development of complex software: A research roadmap". Em: *29th International Conference on Software Engineering 2007 - Future of Software Engineering*, 37–54 (ver pp. 27, 34, 35, 37).

FUGGETTA, A. (2000). "Software Process: A Roadmap". Em: *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, pp. 25–34 (ver pp. 42–44).

GAMMA, E. et al. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison Wesley (ver pp. 38, 39).

GARRO, A. e A. TUNDIS (2012). "A Model-based Method for System Reliability Analysis". Em: *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation* (ver pp. 60, 82).

GAVRAS, A. et al. (2004a). "Towards an MDA-based development methodology". Em: *Lecture Notes in Computer Science* 3047, 230–240 (ver pp. 36, 60–63).

— (2004b). "Towards an MDA-based development methodology for distributed applications". Em: *Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004)*, pp. 71–81 (ver p. 63).

GERVAIS, M.-P. (2002). "Towards an MDA-oriented methodology". Em: *Proceedings of IEEE Computer Society's International Computer Software and Applications Conference*, pp. 265–270 (ver pp. 59–61).

GRIGERA, J. et al. (2012). "From requirements to web applications in an agile model-driven approach". Em: *IEEE Software*, pp. 200–214 (ver p. 77).

GRISS, M. (1995). *Software Reuse: Objects and Frameworks are not Enough*. Rel. téc. Hewlett-Packard Company (ver p. 34).

GUELFI, N. et al. (2004). "DRIP Catalyst: an MDE/MDA method for fault-tolerant distributed software families development". Em: *OOPSLA and GPCE Workshop on Best Practices for Model Driven Development*, pp. 81–90 (ver pp. 60, 81).

HAILPERN, B. e P. TARR (2006). "Model-driven development: The good, the bad, and the ugly". Em: *IBM systems journal* 45.3, 451–461 (ver pp. 27, 34, 35, 37).

HASSINE, J. et al. (2005). "Change impact analysis for requirement evolution using use case maps". Em: *Eighth International Workshop on Principles of Software Evolution*, pp. 81–90 (ver pp. 29, 48).

HEBIG, R. e R. BENDRAOU (2014). "On the Need to Study the Impact of Model Driven Engineering on Software Processes". Em: *Proceedings of the 2014 International Conference on Software and System Process*, pp. 164–168 (ver pp. 60, 76, 77).

HENNINGER, S. (1998). "An environment for reusing software processes". Em: *Proceedings of Fifth International Conference on Software Reuse*, pp. 103–112 (ver p. 44).

HILDENBRAND, T. e A. KORTHAUS (2004). "A model-driven approach to business software engineering". Em: *Eighth World Multi-Conference on Systemics, Cybernetics and Informatics*, pp. 18–21 (ver pp. 60, 70–73).

HILL, J. H. e A. GOKHALE (2012). "Using Template Metaprogramming to Enhance Reuse in Visitor-Based Model Interpreters". Em: *19th International Conference and Workshops on Engineering of Computer Based Systems (ECBS 2012)*, pp. 5–14 (ver pp. 38, 39).

HOLLENBACH, C. e W. FRAKES (1996). "Software Process Reuse in an Industrial Setting". Em: *4th International Conference on Software Reuse*, pp. 22–30 (ver p. 44).

IEEE (2010). *Systems and software engineering — Vocabulary*. IEEE International Standard (ver pp. 42, 43).

JUNIOR, M. B. T. (2012). "MDWA: Uma Abordagem Guiada por Modelos para Desenvolvimento de Software Web". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 60, 82, 83).

JÚNIOR, M. B. THEODORO (2013). "MDWA: Uma abordagem guiada por modelos para desenvolvimento de software Web". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 27, 34–36).

KARAGIANNIS, D., V. HRGOVCIC e R. WOITSCH (2011). "Model driven design for e-applications: the meta model approach". Em: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS '11)*, pp. 451–454 (ver pp. 60, 80).

KIM, S. D. et al. (2005). "DREAM: A practical product line engineering using model driven architecture". Em: *Third International Conference on Information Technology and Applications*, pp. 70–75 (ver pp. 60, 75, 76).

KITCHENHAM, B. (2004). *Procedures for Performing Systematic Reviews*. Rel. téc. Keele University (ver p. 45).

KOCH, N. (2006). "Transformation techniques in the model-driven development process of UWE". Em: *ICWE '06 Workshop proceedings of the sixth international conference on Web engineering* (ver pp. 37, 38).

KOSCIANSKI, A. e M. S. SOARES (2007). *Qualidade de Software: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software*. 2ª ed. Novatec (ver pp. 175, 176).

KRUEGER, C. W. (1992). "Software Reuse". Em: *ACM Computing Surveys* 24.2, pp. 131–183 (ver p. 34).

KULKARNI, V., S. BARAT e U. RAMTEERTHKAR (2011a). "Early experience with agile methodology in a model-driven approach". Em: *Proceedings of the 14th international conference on Model driven engineering languages and systems (MODELS'11)*, pp. 578–590 (ver pp. 60, 73–75, 77, 88).

— (2011b). *MasterCraft – Component-based Development Environment*. Tata Research Development e Design Centre. URL: http://www.tata-mastercraft.com/ (ver p. 73).

LARRUCEA, X., A. B. G. DIEZ e J. X. MANSELL (2004). "Practical model driven development process". Em: *Second European Workshop on Model Driven Architecture*, pp. 99–108 (ver pp. 36, 60, 63–65, 88).

LIMA, V. C. M. (2013). "Programação em Par: Investigando sua Eficácia Perante Tarefas de Modelagem e Construção de Software". Diss. de mestrado. Universidade Tecnológica Federal do Paraná (ver pp. 175, 176).

LIU, D. et al. (2004). “Natural language requirements analysis and class model generation using ucda”. Em: *Innovations in Applied Artificial Intelligence* (ver p. 77).

LONIEWSKI, G., A. ARMESTO e E. INSFRAN (2011). “An agile method for model-driven requirements engineering”. Em: *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, pp. 570–575 (ver p. 77).

LUCRÉDIO, D. (2009). “Uma Abordagem Orientada a Modelos para Reutilização de Software”. Tese de doutorado. Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (ver pp. 27, 33–35, 37, 38).

LUCRÉDIO, D. e R. P. M. FORTES (2010). “Mapping code generation templates do a reference implementation - towards automatic code migration”. Em: *I Brazilian Workshop on Model-Driven Development*, 85–92 (ver pp. 28, 46, 171–173).

LUCRÉDIO, D. et al. (2006). “The Draco approach revisited: Model-driven software reuse”. Em: *VI WDBC - Workshop de Desenvolvimento Baseado em Componentes*, 72–79 (ver p. 34).

LÉDECZI, Á. et al. (2001). “Composing domain-specific design environments”. Em: *Computer* 34.11, pp. 44–51 (ver p. 38).

MATINNEJAD, R. (2011). “Agile model driven development: An intelligent compromise”. Em: *Proceedings of the 2011 Ninth International Conference on Software Engineering Research, Management and Applications (SERA ’11)*, pp. 197–202 (ver p. 77).

MEIRELLES, P. R. M. (2008). *Levantamento de Métricas de Avaliação de Projetos de Software Livre*. Rel. téc. Centro de Competência em Software Livre do Departamento de Ciência da Computação do IME/USP (ver pp. 175, 176).

MICHOTTE, B. e J. VANDERDONCKT (2008). “GrafiXML, a Multi-target User Interface Builder Based on UsiXML”. Em: *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008)*, pp. 15–22 (ver p. 81).

MIDDLEWARE, The Middleware Company (2003). *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis* (ver p. 35).

MILLER, J. e J. MUKERJI (2003). *MDA Guide*. 1.0.1. OMG (ver p. 34).

MUSZYNSKI, M. (2005). “Implementing a domain-specific modeling environment for a family of thick-client gui components”. Em: *The 5th OOPSLA Workshop on Domain-Specific Modeling*, 5–14 (ver pp. 27–29, 38, 41, 42, 47, 111).

NAKICENOVIC, M. BOSTROM (2012). “An agile driven architecture modernization to a model-driven development solution-an industrial experience report”. Em: *International Journal On Advances in Software* 5, pp. 308–322 (ver p. 77).

NEIGHBORS, J. M. (1980). “Software Construction Using Components”. Tese de doutorado. University of California at Irvine (ver pp. 27, 34).

OMG (2003). *MDA Guide Version 1.0.1*. Object Management Group (ver pp. 36, 37, 44, 77).

— (2015). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.2*. Object Management Group (ver p. 37).

PAPOTTI, P. E. (2013). "Um Processo Dirigido a Modelos para Geração de Código". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 27, 34, 35, 38, 39, 115, 120).

PAPOTTI, P. E., A. F. DO PRADO e W. L. DE SOUZA (2012). "An Approach to Support Legacy Systems Reengineering to MDD Using Metaprogramming". Em: *XXXVIII Conferencia Latinoamericana En Informatica (CLEI'2012)*, pp. 1–10 (ver pp. 27, 34, 35, 38).

PARREIRAS, F. S. e M. P. BAX (2005). "Geração de Sistemas de Gestão de Conteúdo com Softwares Livres". Em: *XXXI Latin American Informatics conference (CLEI'2005)* (ver pp. 36, 37).

PELLEGRINI, F. G. et al. (2010). "Transformações de Modelos para um Processo MDA". Em: *X Escola Regional de Computação Bahia-Alagoas-Sergipe (ERBASE 2010)* (ver p. 36).

PERINI, L. C. (2015). "Um Editor Simultâneo para Facilitar a Evolução de Templates de Geração de Códigos". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 29, 30, 39, 40, 45, 48, 52, 53, 88, 89, 91, 94, 106–108, 115, 119, 121, 131, 171).

PHAM, H. et al. (2007). "Applying Model-Driven Development Techniques to the Development of Search and Rescue Systems". Em: *IEEE International Conference on System of Systems Engineering*, pp. 1–6 (ver pp. 27, 34, 35, 37).

POSSATTO, M. A. (2013). "Uma Abordagem para Migração Automática de Código no Contexto do Desenvolvimento Orientado a Modelos". Diss. de mestrado. Departamento de Computação - Universidade Federal de São Carlos (ver pp. 27–30, 34, 38, 39, 41, 42, 45–48, 52, 53, 88, 89, 91, 94, 106–108, 115, 119, 121, 131, 171).

PRESSMAN, R. S. (2011). *Engenharia de Software: Uma abordagem profissional*. 7ª ed. The McGraw-Hill (ver pp. 33, 42, 43).

RAUSCH, A. et al. (2005). "The V-Modell XT Applied - Model-Driven and Document-Centric Development". Em: *Proceedings 3rd World Congress for Software Quality* 3 (ver p. 77).

RAUSCHMAYER, A., A. KNAPP e M. WIRSING (2004). "Consistency Checking in an Infrastructure for Large-Scale Generative Programming". Em: *IEEE Int. Conf. Automated Software Engineering* (ver p. 33).

RAUTENBERG, S. et al. (2008). "Uma Metodologia para o Desenvolvimento de Ontologias". Em: *Revista Ciências Exatas e Naturais* 10.2, 237–262 (ver p. 43).

REIS, R. Q., C. A. L. REIS e D. J. NUNES (2001). "Automated support for software process reuse: Requirements and early experiences with the APSEE model". Em: *Proceedings of Seventh International Workshop on Groupware*, pp. 50–55 (ver p. 44).

ROUILLÉ, E. et al. (2013). "Improving Reusability in Software Process Lines". Em: *Software Engineering and Advanced Applications (SEAA)*, pp. 90–93 (ver p. 44).

RU-ZHI, X. et al. (2005). "Reuse-Oriented Process Component Representation and Retrieval". Em: *International Conference on Computer and Information Technology*, pp. 911–915 (ver p. 44).

RUIZ, F. J. B., Ó. S. RAMÓN e J. G. MOLINA (2013). "Definition of Processes for MDE-based Migrations". Em: *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering* (ver pp. 60, 69, 70).

SCHMIDT, D. C. (2006). "Guest Editor's Introduction: Model-Driven Engineering". Em: *Computer* 39.2, pp. 25–31 (ver p. 34).

SCHRAMM, A. et al. (2010). "Rapid UI Development for Enterprise Applications: Combining Manual and Model-driven Techniques". Em: *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, pp. 271–285 (ver pp. 60, 86, 87).

SOARES, I. W. (2012). "PM-MDA: Um Método para o Desenvolvimento de Modelos de Plataforma no Contexto da MDA". Tese de doutorado. Universidade Tecnológica Federal do Paraná (ver pp. 60, 84).

TEIXEIRA, E. N. (2014). "ODYSSEYPROCESSREUSE: Uma Proposta de Sistemática de Engenharia de Linha de Processo de Software Baseada em um Projeto Arquitetural de Componentes de Processos de Software". Qualificação de Doutorado. Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia - Universidade Federal do Rio de Janeiro (ver p. 43).

TRAVASSOS, G., D. GUROV e E. AMARAL (2002). *Introdução à engenharia de software experimental*. Rel. téc. Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia - Universidade Federal do Rio de Janeiro (ver p. 115).

VANDERDONCKT, J. (2005). "A MDA-Compliant Environment for Developing User Interfaces of Information Systems". Em: *Advanced Information Systems Engineering*, pp. 16–31 (ver p. 81).

VARRO, D. e A. BALOGH (2007). "The model transformation language of the VIATRA2 framework". Em: *Science of Computer Programming*, 214–234 (ver p. 37).

VOGEL, L. (2009). *Java Emitter Template (JET) - Tutorial*. URL: http://www.vogella.de/articles/EclipseJET/article.html (ver p. 39).

VÖELTER, M. (2008). *MD* Best Practices Version 1.1*. URL: http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf (ver p. 34).

VÖELTER, M. e I. GROHER (2007). "Product line implementation using aspect-oriented and model-driven software development". Em: *Proceedings of the 11th International Software Product Line Conference*, pp. 233–242 (ver p. 34).

WHITE, J., D. C. SCHMIDT e A. GOKHALE (2005). "Simplifying Autonomic Enterprise Java Bean Applications Via Model-Driven Development: A Case Study". Em: *8th In-*

ternational Conference Model Driven Engineering Languages and Systems (MoDELS 2005), pp. 601–615 (ver p. 38).

WOHLIN, C. et al. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers (ver pp. 109, 115, 116, 119).

YAGHOOBI, M., M. TORKAMANI e M. YAGHOOBI (2011). "Review and Comparison some model driven methodologies and mapping MDA to RUP". Em: *Proceedings of the 5th Symposium on Advances in Science and Technology (SASTech)* (ver p. 77).

YOON, Y. e B. A. MYERS (2011). "Capturing and Analyzing Low-level Events from the Code Editor". Em: *3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU '11)*, pp. 25–30 (ver pp. 28, 45).

ZAMBONI, A. B. et al. (2010). "StArt Uma Ferramenta Computacional de Apoio à Revisão Sistemática". Em: *Proceedings of the Brazilian Conference on Software: Theory and Practice - Tools session* (ver p. 58).

ZHANG, Y. e S. PATEL (2011). "Agile Model-Driven Development in Practice". Em: *IEEE Software* 28 (ver p. 77).

Apêndices

APÊNDICE A – Protocolo de nível de experiência do usuário

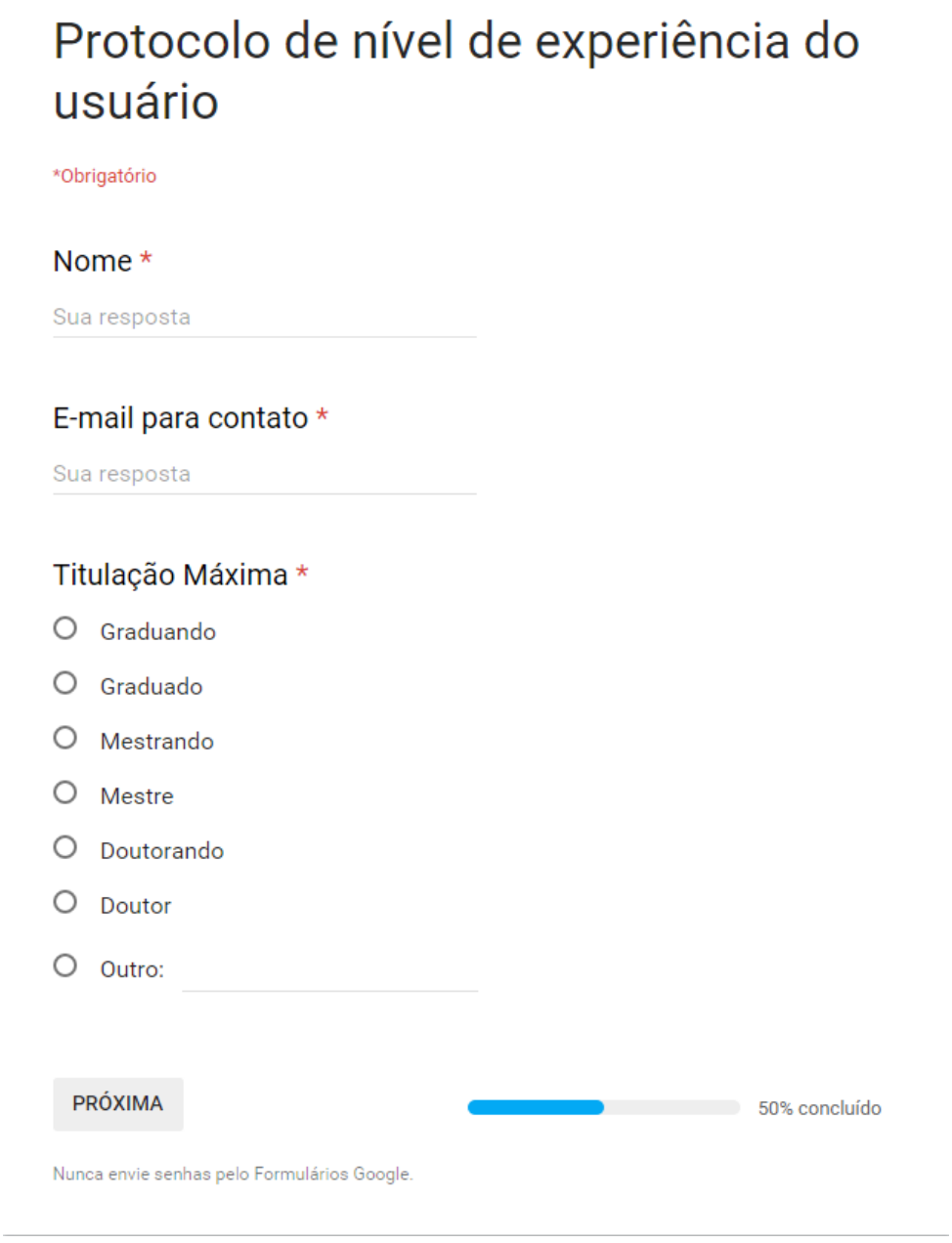
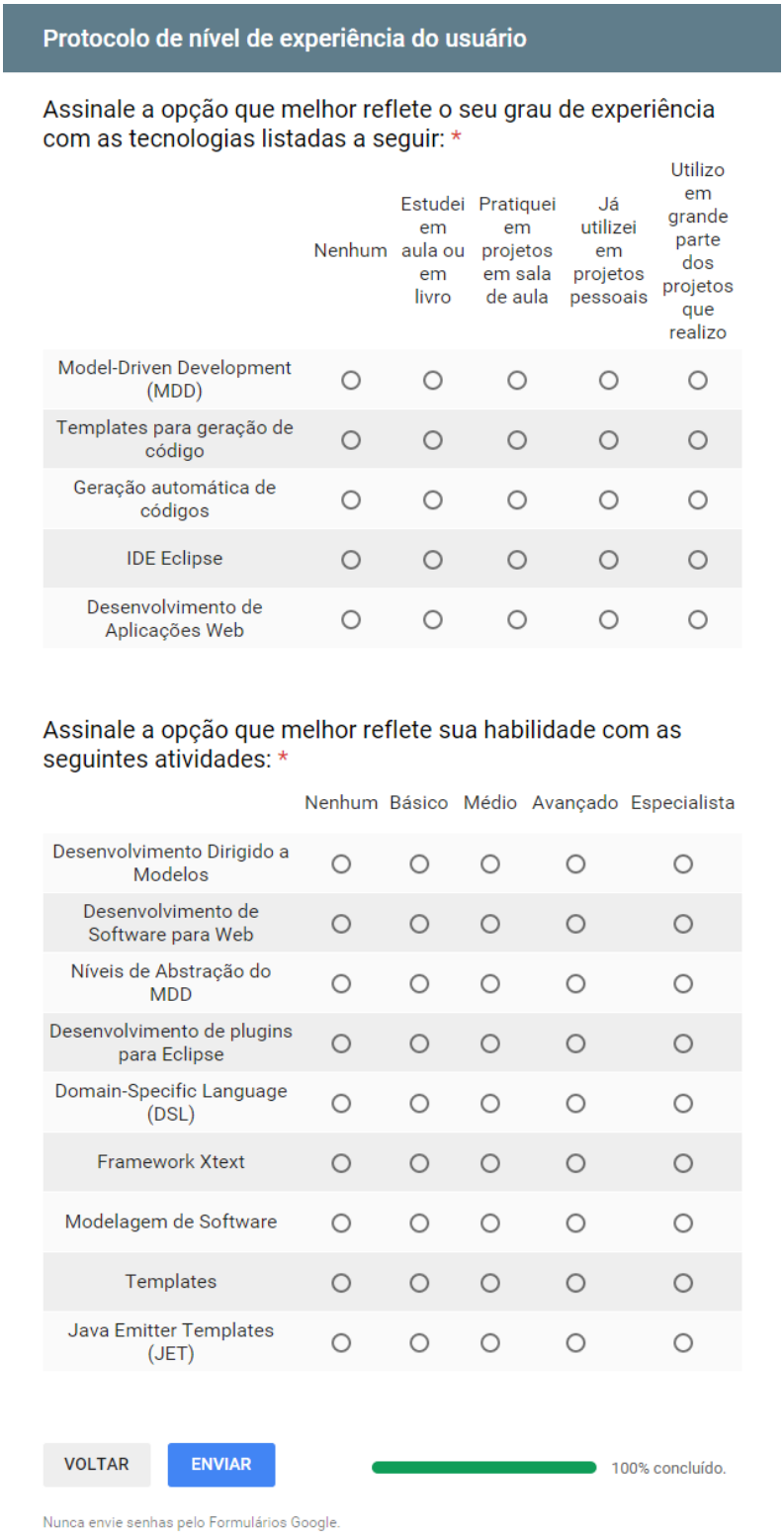


Figura 52: Primeira parte do Protocolo de nível de experiência do usuário.



Protocolo de nível de experiência do usuário

Assinale a opção que melhor reflete o seu grau de experiência com as tecnologias listadas a seguir: *

	Nenhum	Estudei em aula ou em livro	Pratiquei em projetos em sala de aula	Já utilizei em projetos pessoais	Utilizo em grande parte dos projetos que realizo
Model-Driven Development (MDD)	○	○	○	○	○
Templates para geração de código	○	○	○	○	○
Geração automática de códigos	○	○	○	○	○
IDE Eclipse	○	○	○	○	○
Desenvolvimento de Aplicações Web	○	○	○	○	○

Assinale a opção que melhor reflete sua habilidade com as seguintes atividades: *

	Nenhum	Básico	Médio	Avançado	Especialista
Desenvolvimento Dirigido a Modelos	○	○	○	○	○
Desenvolvimento de Software para Web	○	○	○	○	○
Níveis de Abstração do MDD	○	○	○	○	○
Desenvolvimento de plugins para Eclipse	○	○	○	○	○
Domain-Specific Language (DSL)	○	○	○	○	○
Framework Xtext	○	○	○	○	○
Modelagem de Software	○	○	○	○	○
Templates	○	○	○	○	○
Java Emitter Templates (JET)	○	○	○	○	○

VOLTAR ENVIAR 100% concluído.

Nunca envie senhas pelo Formulários Google.

Figura 53: Segunda parte do Protocolo de nível de experiência do usuário.

APÊNDICE B – Tarefas do Experimento

B.1 Tarefa 1

Criar um Cadastro de Receitas que deve conter o seguintes campos:

- Médico;
- Paciente;
- Medicamento; e
- Dosagem (Tipo: Texto).

Esta alteração tem um prazo estimado de 1 hora.

Figura 54: Passos para a realização da tarefa 1 para o grupo método.

Figura 55: Passos para a realização da tarefa 1 para o grupo manual.

B.2 Tarefa 2

Criar um arquivo de configuração fixo para o Banco de Dados, deixando os seguintes dados como padrão:

- Host: localhost;
- Port: 5432;
- User: postgres;
- Password: postgres; e
- Dbname: Nome do projeto.

Dica: Faça esse arquivo como uma classe PHP. Esta alteração tem um prazo estimado de 30 minutos.

Figura 56: Passos para a realização da tarefa 2 para o grupo método.

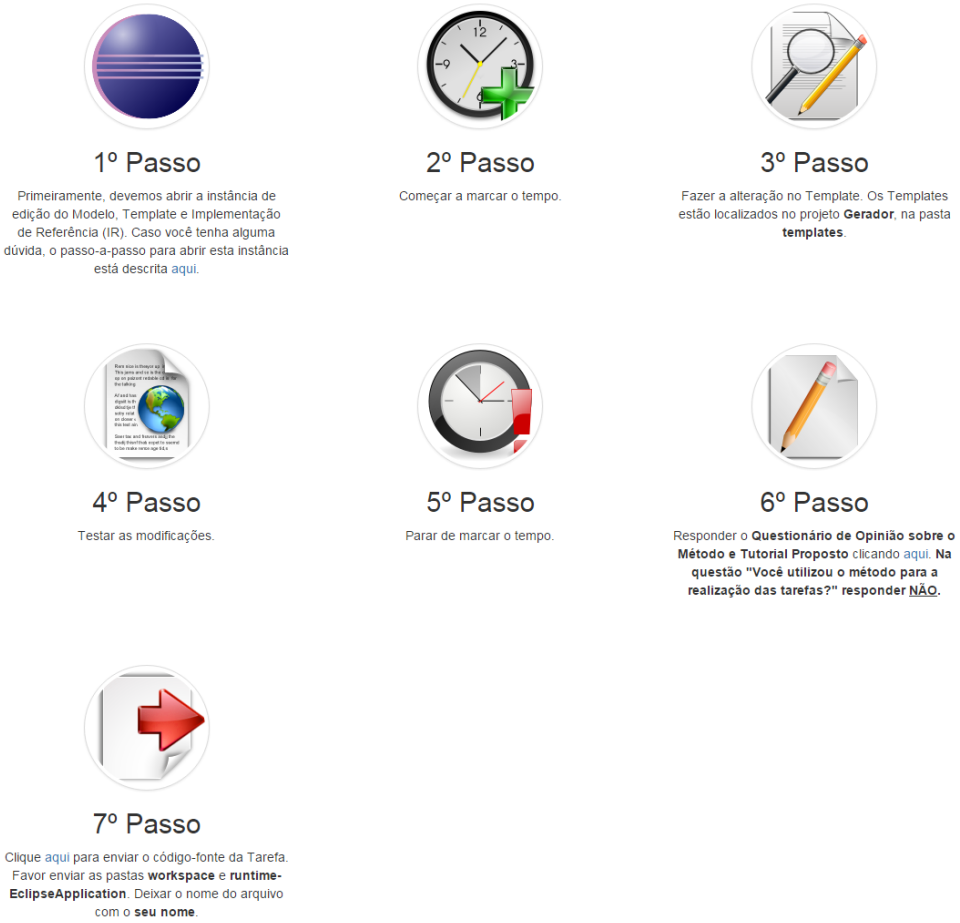


Figura 57: Passos para a realização da tarefa 2 para o grupo manual.

B.3 Tarefa 3

Colocar os Totais em campos do tipo int e currency em cada tabela. Após isso, fazer com que as classes DB utilizem as configurações do arquivo anterior. Esta alteração tem um prazo estimado de 3 horas.

Figura 58: Passos para a realização da tarefa 3 para o grupo método.

Figura 59: Passos para a realização da tarefa 3 para o grupo manual.

B.4 Tarefa 4

Criar o tipo Password no Metamodelo e sua DSL. Esta alteração tem um prazo estimado de 2 horas.

Figura 60: Passos para a realização da tarefa 4 para o grupo método.

Figura 61: Passos para a realização da tarefa 4 para o grupo manual.

B.5 Tarefa 5

Alterar o sistema para que funcione como um Software as a Service (SaaS). Para isso você precisará:

- Criar um Cadastro de Consultório;
- Criar um Cadastro de Usuário com os seguintes campos:
  - Nome;
  - E-mail;
  - Senha (utilizar MD5 como criptografia); e
  - Consultório.
- Adicionar no Metamodelo e na sua DSL um novo tipo de Form (FormTypes) com o nome de FormLogin, tendo como parâmetros:
  - Entity
  - userField
  - passwordField
- Fazer a página de login e guardar a ID do usuário na $_SESSION;
- Alterar o Metamodelo e sua DSL para que ele suporte atributos (Features) que não serão preenchidos pelo usuário (hidden).
- Adicionar um atributo do tipo Consultório em todos os Cadastros, sendo que este atributo não será preenchido pelo usuário; e
- Após isso, fazer esse tratamento nos Formulários, sendo que o código do Consultório deve ser lido do Cadastro do Usuário atual.

Esta alteração tem um prazo estimado de 10 horas.

Figura 62: Primeira parte dos passos para a realização da tarefa 5 para o grupo método.

Figura 63: Segunda parte dos passos para a realização da tarefa 5 para o grupo método.

Figura 64: Primeira parte dos passos para a realização da tarefa 5 para o grupo manual.

Figura 65: Segunda parte dos passos para a realização da tarefa 5 para o grupo manual.

APÊNDICE C – Protocolo de simulação das tarefas

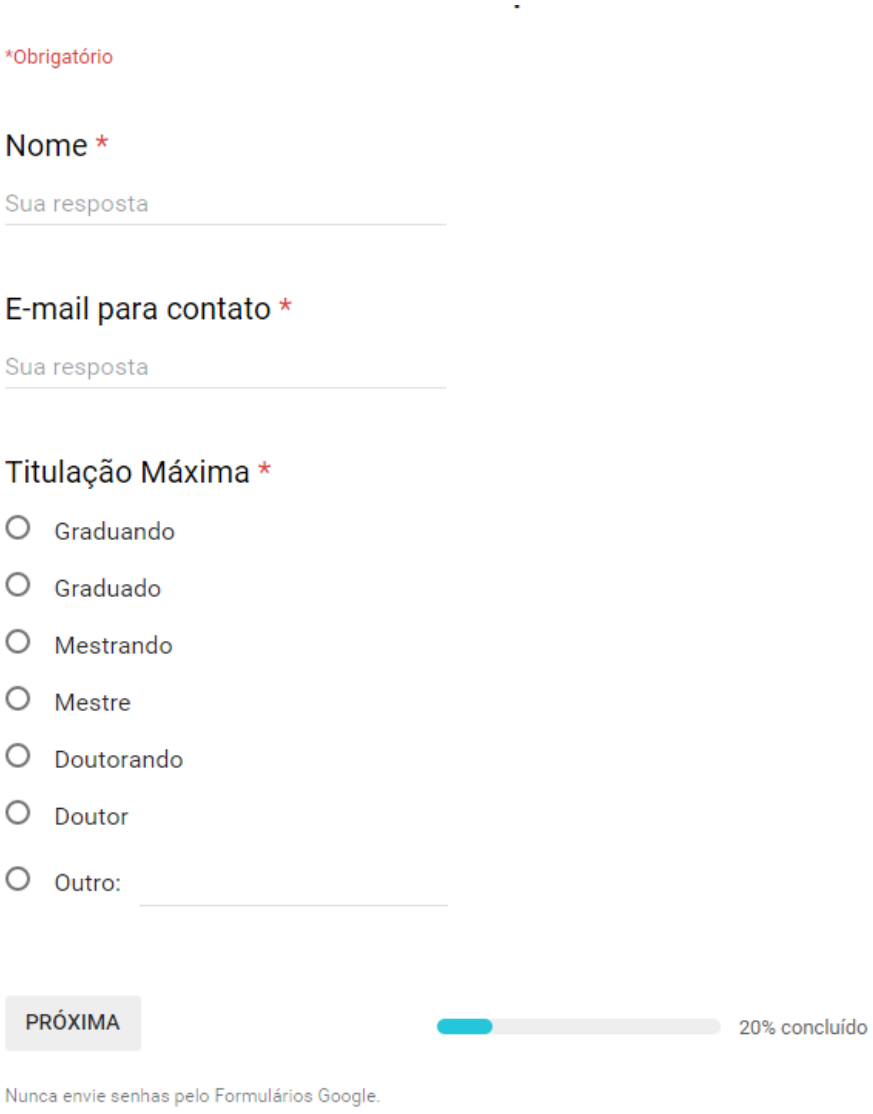


Figura 66: Primeira parte do Protocolo de simulação das tarefas.

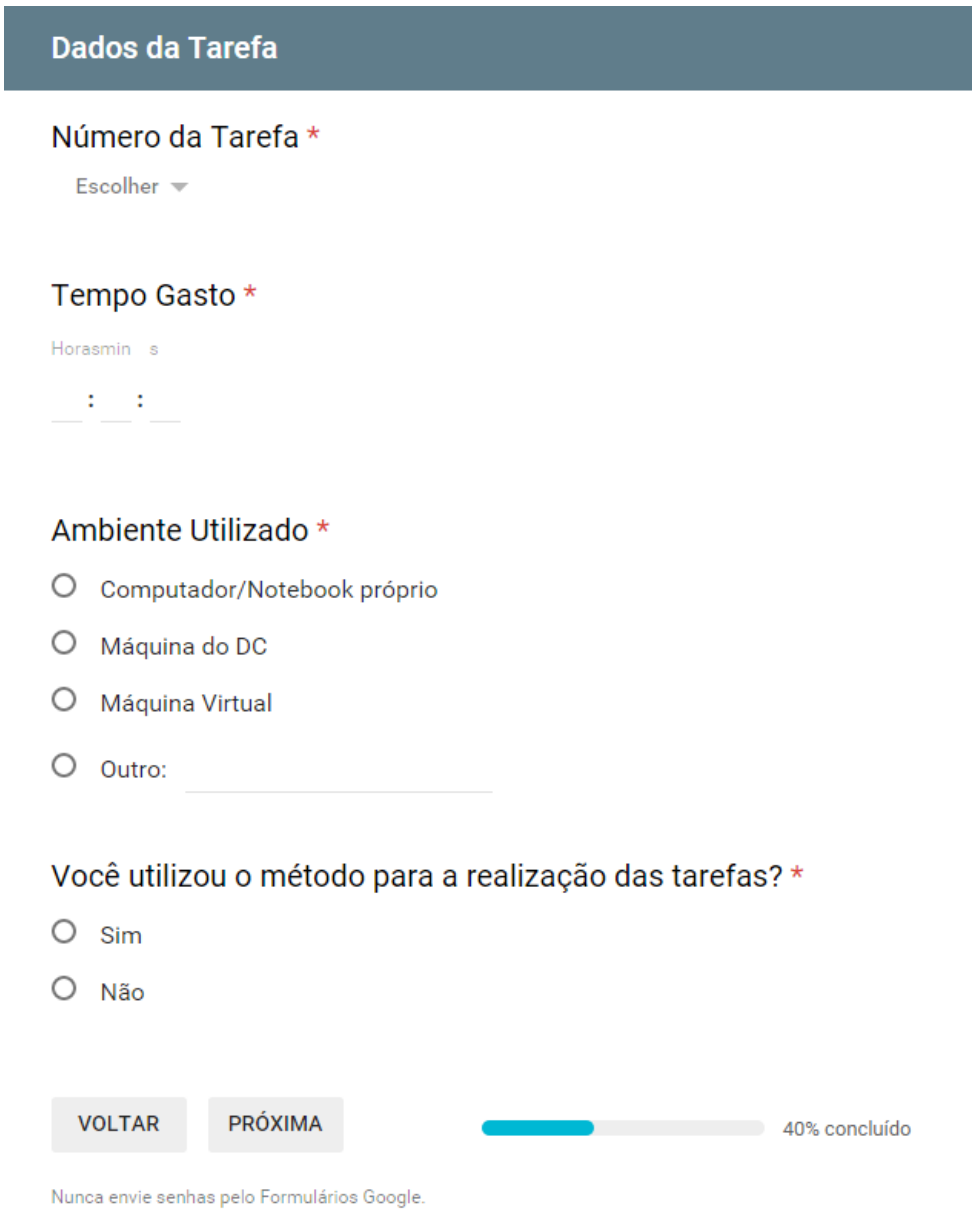


Figura 67: Segunda parte do Protocolo de simulação das tarefas.

APÊNDICE D – Questionário de Opinião sobre o Método e Tutorial Proposto

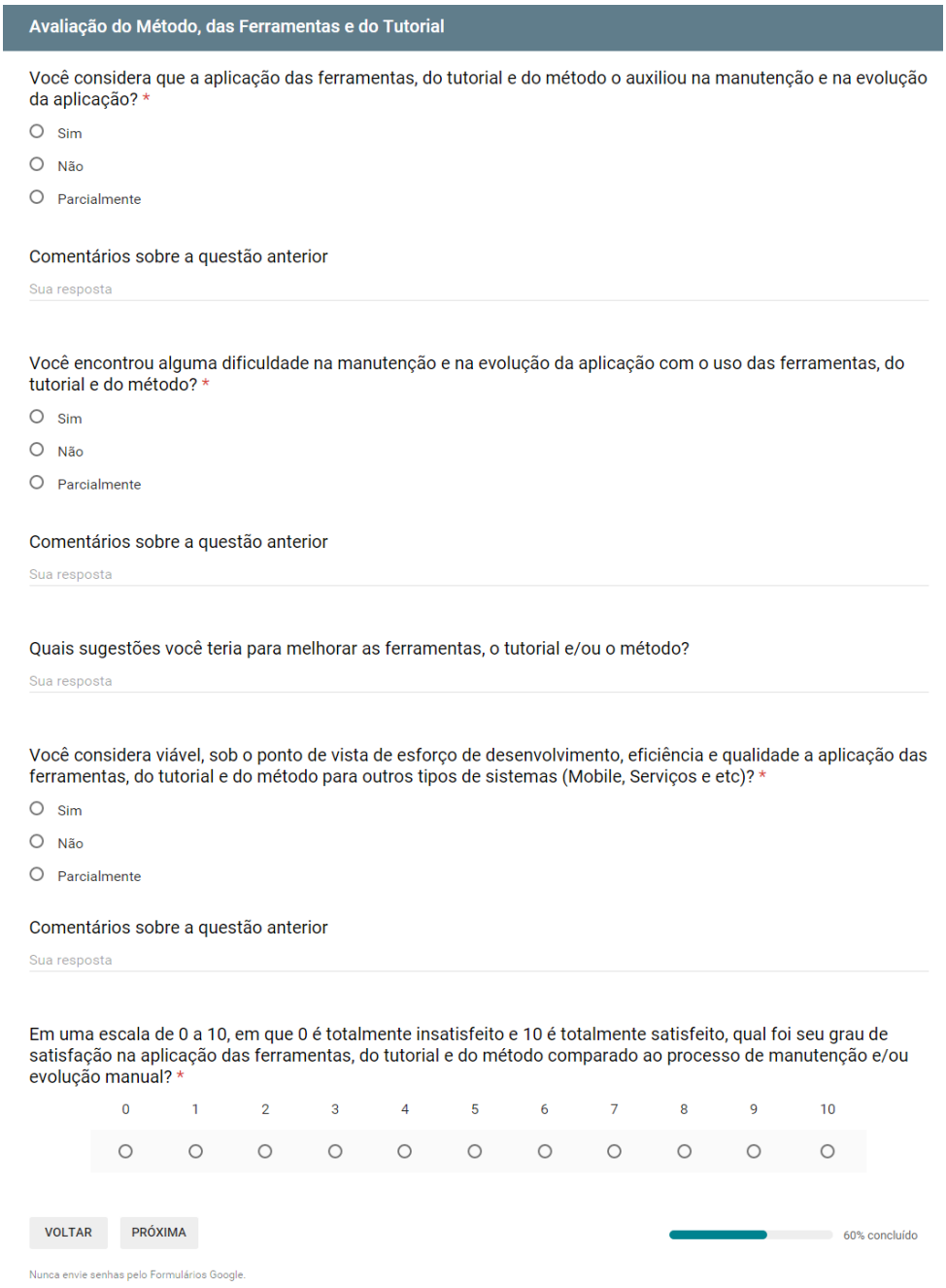


Figura 68: Primeira parte do Questionário de Opinião sobre o Método e Tutorial Proposto.

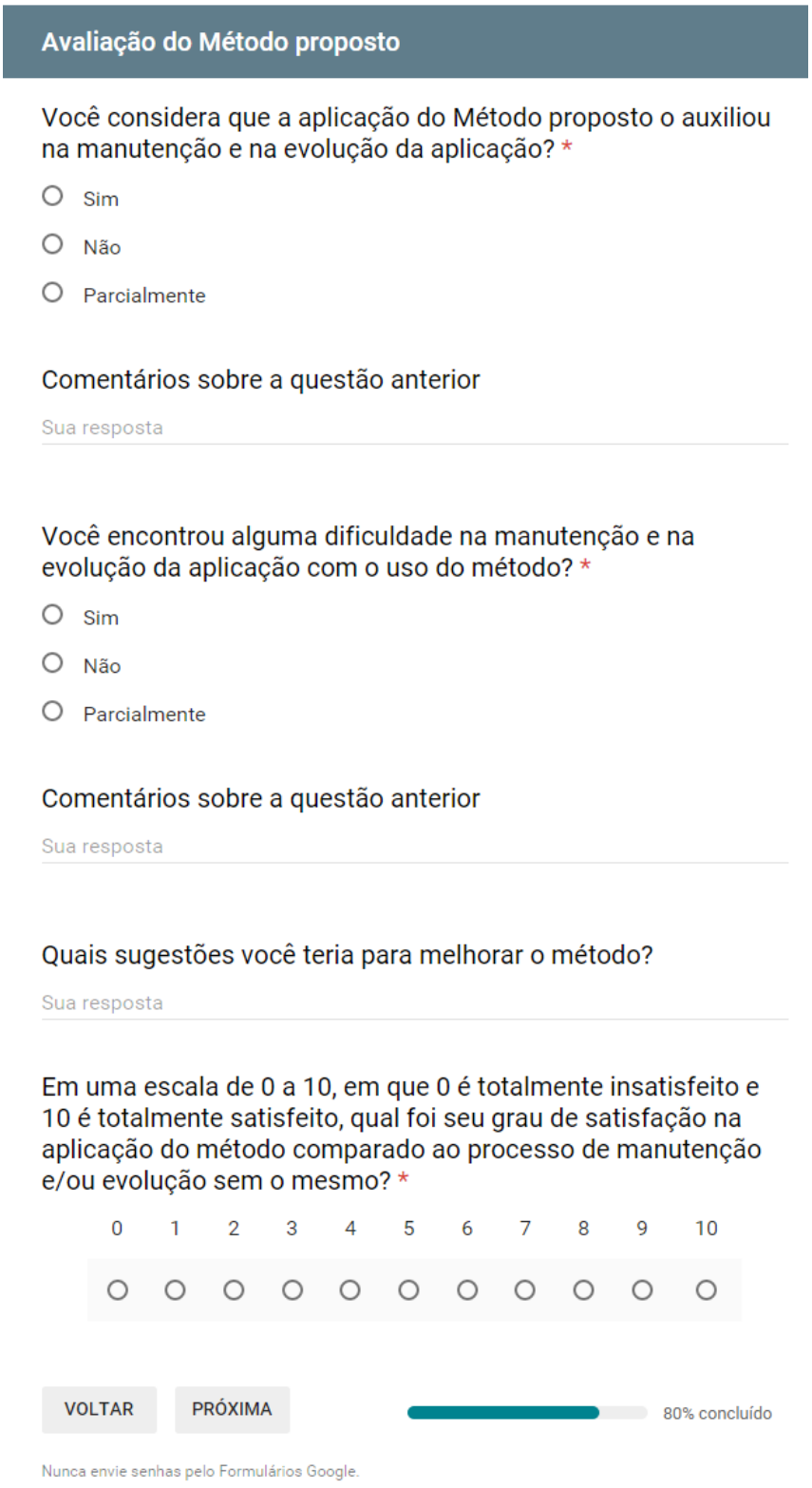
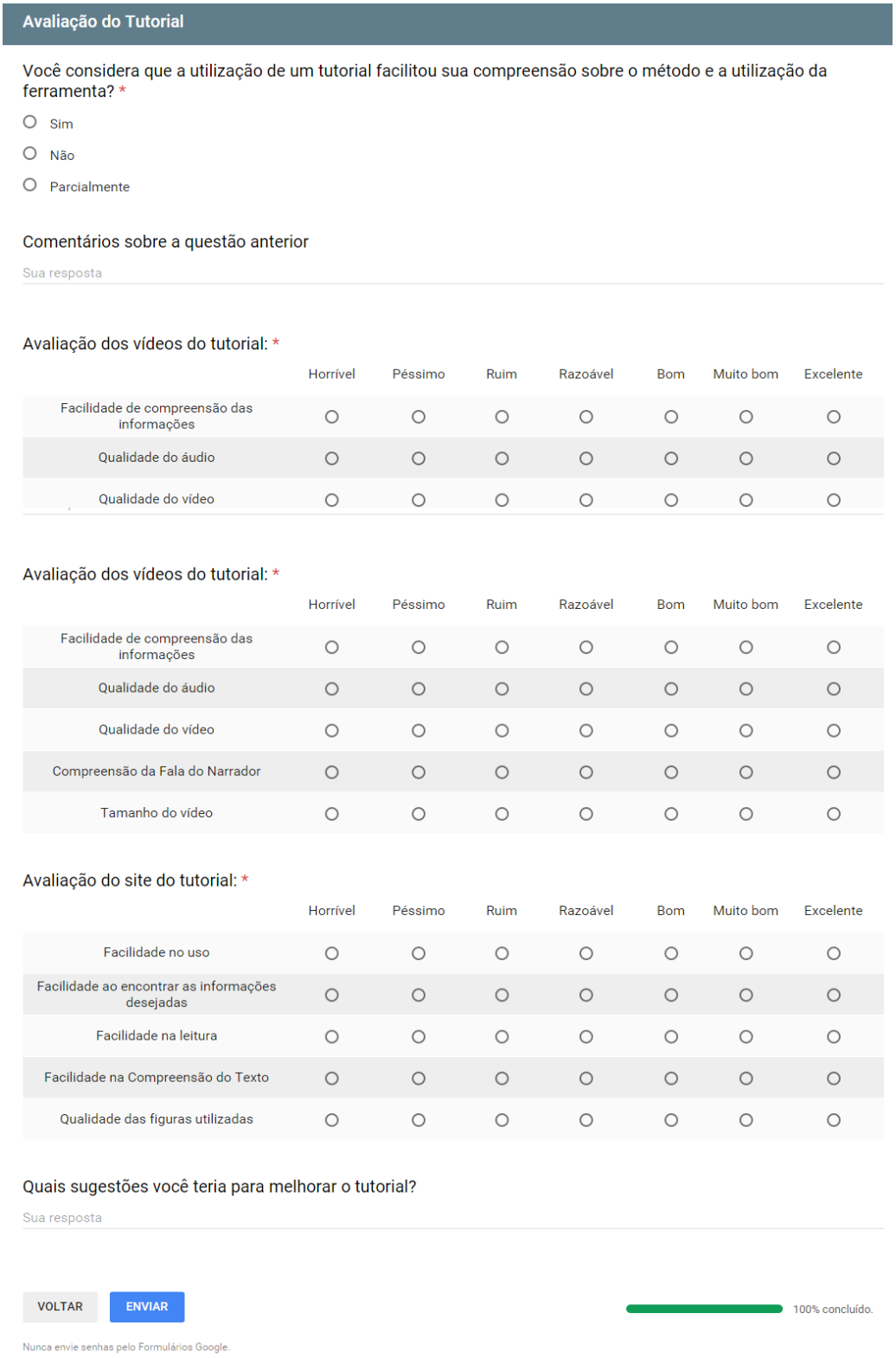


Figura 69: Segunda parte do Questionário de Opinião sobre o Método e Tutorial Proposto.



Avaliação do Tutorial

Você considera que a utilização de um tutorial facilitou sua compreensão sobre o método e a utilização da ferramenta? *

- Sim
- Não
- Parcialmente

Comentários sobre a questão anterior

Sua resposta

Avaliação dos vídeos do tutorial: *

	Horrível	Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente
Facilidade de compreensão das informações							
Qualidade do áudio							
Qualidade do vídeo							

Avaliação dos vídeos do tutorial: *

	Horrível	Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente
Facilidade de compreensão das informações							
Qualidade do áudio							
Qualidade do vídeo							
Compreensão da Fala do Narrador							
Tamanho do vídeo							

Avaliação do site do tutorial: *

	Horrível	Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente
Facilidade no uso							
Facilidade ao encontrar as informações desejadas							
Facilidade na leitura							
Facilidade na Compreensão do Texto							
Qualidade das figuras utilizadas							

Quais sugestões você teria para melhorar o tutorial?

Sua resposta

VOLTAR ENVIAR 100% concluído.

Nunca envie senhas pelo Formulários Google.

Figura 70: Terceira parte do Questionário de Opinião sobre o Método e Tutorial Proposto.

APÊNDICE E – Questionário de Opinião sobre o Experimento

Avaliação do Experimento

Você encontrou dificuldade na execução do experimento? *

- Sim
- Não
- Parcialmente

Comentários sobre a questão anterior

Sua resposta

Você acredita que a criação de um método contendo um passo-a-passo facilitaria a manutenção/evolução neste ambiente? *

- Sim
- Não
- Parcialmente

Comentários sobre a questão anterior

Sua resposta

Avaliação do Experimento *

	Horrível	Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente
Facilidade de compreensão das informações	○	○	○	○	○	○	○
Facilidade no entendimento das tarefas	○	○	○	○	○	○	○
Facilidade ao encontrar as informações necessárias para a execução da tarefa	○	○	○	○	○	○	○

Quais sugestões você teria para melhorar o experimento?

Sua resposta

Figura 71: Questionário de Opinião sobre o Experimento.

Anexos

ANEXO A – Tipos de Mapeamento

Este anexo apresenta os tipos de mapeamento utilizados pelas ferramentas de Possatto (2013) e Perini (2015). Esses tipos de mapeamento foram propostos no estudo realizado por Lucrédio e Fortes (LUCRÉDIO e FORTES 2010).

A.1 Tipo de Mapeamento 1 - Cópia Simples

Esse é o tipo mais simples de mapeamento, onde um trecho de código de um *template* é copiado para Implementação de Referência (IR), conforme exemplificado na Figura 72. Nesse tipo de mapeamento a relação entre *template* e IR é de um-para-um e não há necessidade de consulta ao modelos de entrada.

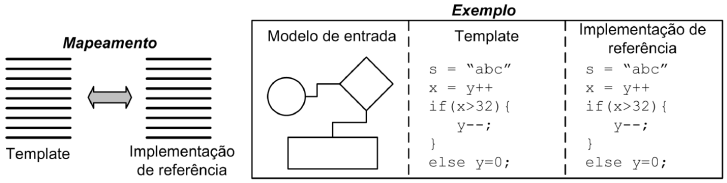


Figura 72: Esquematização do Mapeamento Tipo 1 (LUCRÉDIO e FORTES 2010).

A.2 Tipo de Mapeamento 2 - Substituição Simples

Uma chamada de consulta ao modelo de entrada é inserida no *template*, sendo que essa chamada é substituída pelo valor obtido do modelo após a geração do código-fonte, conforme exemplificado na Figura 73. Nesse tipo de mapeamento é possível a realização do caminho inverso, portanto, é possível rastrear o valor substituído até o modelo de entrada. A relação entre *template* e IR é um-para-um.

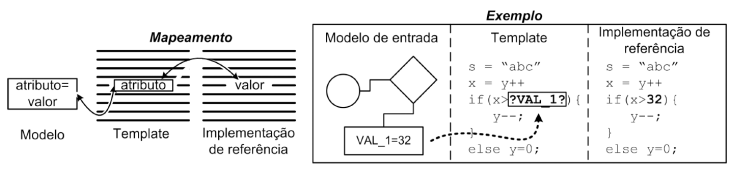


Figura 73: Esquematização do Mapeamento Tipo 2 (LUCRÉDIO e FORTES 2010).

A.3 Tipo de Mapeamento 3 - Substituição Indireta

Esse tipo de mapeamento é similar ao tipo 2 (Seção A.2), contudo não é possível a rastreabilidade do valor substituído, já que o mesmo não está explícito no modelo, pois esse valor é resultado de um cálculo realizado no *template*, conforme demonstrado na Figura 74. A relação entre *template* e IR é semelhante ao tipo 2, relação um-para-um.

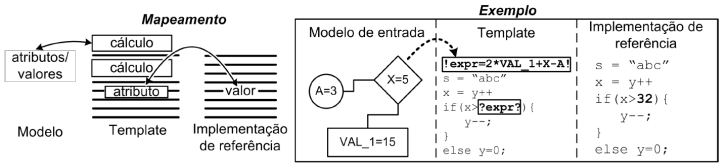


Figura 74: Esquematização do Mapeamento Tipo 3 (LUCRÉDIO e FORTES 2010).

A.4 Tipo de Mapeamento 4 – Repetição

Um trecho de código de um *template* é repetido, zero ou mais vezes, na IR, obedecendo uma condição, podendo ou não ser obtido do modelo de entrada (Figura 75), sendo assim, a relação entre *template* e Implementação de Referência é de um-para-muitos.

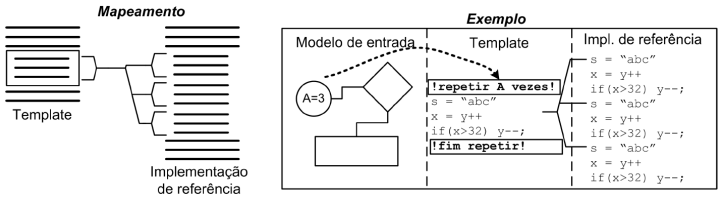


Figura 75: Esquematização do Mapeamento Tipo 4 (LUCRÉDIO e FORTES 2010).

A.5 Tipo de Mapeamento 5 - Condicional

Um trecho de código da IR pode ser mapeado para diferentes trechos de um *template*, tendo como base uma condição estabelecida no *template*, conforme apresentado na Figura 76. Essa condição pode ou não consultar o modelo de entrada. Sendo assim, a relação entre *template* e Implementação de Referência é de muitos-para-um.

A.6 Tipo de Mapeamento 6 - Inclusão

Nesse tipo de mapeamento, o *template* inclui outro *template* para a geração de um determinado trecho de código (Figura 77), sendo que é importante destacar que é sempre possível identificar, a cada linha da IR, o *template* responsável por sua geração. Dessa forma, a relação entre *template* e Implementação de Referência é de muitos-para-um.

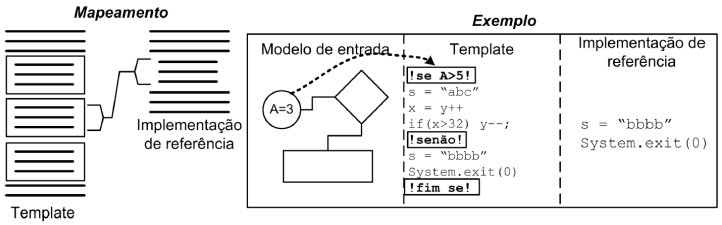


Figura 76: Esquematização do Mapeamento Tipo 5 (LUCRÉDIO e FORTES 2010).

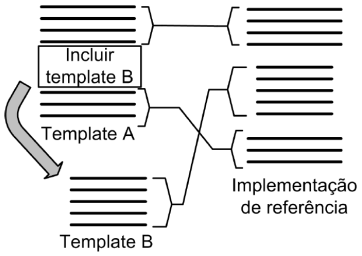


Figura 77: Esquematização do Mapeamento Tipo 6 (LUCRÉDIO e FORTES 2010).

A.7 Tipo de Mapeamento 7 - Novo Arquivo

Um *template* solicita a criação de um novo arquivo e todo o código gerado por esse *template* é direcionado para o novo arquivo (Figura A.7), e, conforme o tipo de mapeamento 6 (Seção A.6), sempre é possível identificar, a cada linha da IR, o *template* responsável por sua geração. Nesse tipo de mapeamento, a relação entre *template* e Implementação de Referência é de um-para-muitos.

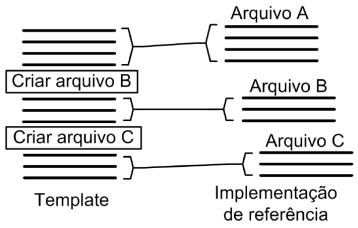


Figura 78: Esquematização do Mapeamento Tipo 7 (LUCRÉDIO e FORTES 2010).

ANEXO B – Métodos de Verificação de Qualidade do Código-Fonte

De acordo com os estudos realizados por Meirelles (2008) e Lima (2013), a partir do código-fonte de um software, é possível a realização de medições sobre a sua estrutura e organização interna, com o intuito de avaliar sua qualidade. Essas medições são possíveis graças a métodos padronizados, chamados de métricas. Há vários tipos diferentes de métricas:

- Quanto ao objeto: métricas de produto e de processo;
- Quanto aos critérios: métricas objetivas (exemplo: Número de linhas de código) e subjetivas (exemplo: Modelo de estimativa de custo); e
- Quanto ao método de obtenção: primitivas (exemplo: Erros indicados em um teste de unidade e número de linhas de código) ou compostas (exemplo: Número de erros encontrados a cada mil linhas de código).

Os atributos complexidade e tamanho final do programa são exemplos de métricas de produto, e a metodologia utilizada para se determinar esses atributos são exemplos de métricas de processo (MEIRELLES 2008; LIMA 2013).

A seguir, a descrição das métricas adotadas neste trabalho.

B.1 Número de linhas de código

Esta métrica, também conhecida como *Lines of Code* (LOC), é classificada como uma métrica de produto, objetiva e primitiva. Sendo que ela quantifica o número de linhas de código dentro de um ou mais arquivos do código-fonte. Essa métrica avalia a medida de tamanho de software (linhas de código), comparando produtos maiores com menores, podendo ser utilizada para indicar um maior tempo de desenvolvimento e, consecutivamente, um custo maior, além de um trabalho de produção mais complexo (KOSCIANSKI e SOARES 2007; MEIRELLES 2008; LIMA 2013).

B.2 Complexidade ciclomática de McCabe

Esta métrica é classificada como uma métrica de produto, objetiva e primitiva. Sendo que ela quantifica o número de caminhos em um trecho de código que é calculado

apenas para os métodos. Como por exemplo, cada vez que uma ramificação ocorre (*if*, *for*, *while*, *do*, *case*, *catch* e operadores lógicos em expressões condicionais) o valor dessa medida é incrementado em um, desta forma, há uma forte correlação entre o número de erros encontrados em um software e o tamanho e a complexidade interna do mesmo. Os estudos mais recentes sobre essa métrica apontam que 15 é o valor máximo aceitável de complexidade, além desse valor, deve-se particionar o código-fonte (KOSCIANSKI e SOARES 2007; MEIRELLES 2008; LIMA 2013).