UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO PELOS MODELOS DE OBJETOS PERSISTENTES E DE NEGÓCIO

JOÃO PAULO MOREIRA DOS SANTOS

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

São Carlos – SP

Março/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO PELOS MODELOS DE OBJETOS PERSISTENTES E DE NEGÓCIO

JOÃO PAULO MOREIRA DOS SANTOS

Dissertação apresentada 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. Antonio Francisco do Prado

São Carlos – SP

Março/2017

Primeiramente, dedico esse trabalho a Deus, por ser essencial em minha vida. De modo especial, dedico esse trabalho aos meus pais Bernadete e Tadeu, que me ensinaram a importância do estudo e sempre batalharam para dar condições para que eu pudesse chegar até aqui. Agradeço também aos meus parentes e amigos pelo companheirismo e apoio.

AGRADECIMENTOS

Agradeço a Deus por ter me guiado em toda minha trajetória durante o mestrado. Apesar das dificuldades e momentos de aflição, em nenhum momento fiquei desamparado e, graças a Ele, com muita felicidade pude concluir meus objetivos.

Obrigado a todos os meus familiares, por confiarem no meu esforço e pela força que me deram ao longo dessa jornada. Em especial, um agradecimento a minha mãe Bernadete e minhas irmãs Natália e Naize, por serem meu porto seguro. Sem o apoio de vocês, nada disso poderia ter se concretizado.

De modo especial, agradeço ao meu orientador, professor Prado, por sempre confiar na minha capacidade, pela amizade e ensinamentos durante minha formação.

Agradeço os meus amigos Renato, Danillo, Pedro, Diógenes, Weider e Ricardo, que estamos juntos nessa jornada desde a graduação.

Obrigado aos amigos e colegas do DC, que enfrentaram o mestrado junto comigo: Elias, Guido, Pacini, Steve, Fernando, Abade, Gaspar, Gastaldi, Jésus, Bento, Odair, Elis, Ivan, Cleiton, Vitinho, Sanches, Porto, Flávio, Paulão, Rafael, Lucas, Maykon, Marcela e Suzane.

Aos amigos que moram em São Carlos e com quem pude compartilhar bons momentos: Wilmax, Gisele, FCarlos, Aline, Adam, Victor, Mello, Iohan, Nilton, Pneu, Edvaldo, Geovanna e Mantovani.

Por fim, a todos que ajudaram, torceram, ou de alguma forma, contribuíram para meu sucesso na realização deste trabalho. **De coração, muito obrigado a todos!**

Nunca diga que aquilo que você está fazendo ou tentando fazer está péssimo, pois o processo de aprendizagem admite risco, erros e falhas.

Steve Ataky T.M.

RESUMO

A Engenharia de Software tem como objetivo apoiar em todos os aspectos do desenvolvimento de software, tendo como alguns dos seus desafios a diminuição do tempo de entrega e as melhorias que facilitam o desenvolvimento do software. Assim, é necessário criar ferramentas, métodos e mecanismos que apoiam e melhoram o processo de desenvolvimento do software, incluindo a sua manutenção, considerando as mudanças de requisitos ao longo do tempo. Essas manutenções podem ocorrer devido às evoluções de tecnologias e as novas necessidades dos seus usuários. Com o objetivo de apoiar o Processo de Desenvolvimento de Software (PDS), técnicas de geração de código e de gerenciamento de processos de negócio têm sido bastante utilizadas. Com esse foco, e visando melhorar o PDS, pesquisou-se um conjunto de técnicas que, combinadas, resultou numa abordagem de desenvolvimento, cujas bases são os Modelos do Negócio e de Classes de Objetos Persistentes, e a Geração de Código. Essas técnicas, atualmente bastante utilizadas pelos desenvolvedores, integradas na abordagem proposta, possibilitou apoiar e orientar o desenvolvedor tanto na construção como na manutenção do software. A abordagem define atividades que deverão ser realizadas durante o desenvolvimento do software, e emprega conceitos, técnicas, e ferramentas, que apoiam essas atividades. A abordagem foi avaliada através do desenvolvimento de um estudo experimental, e verificou-se que sua utilização proporcionou melhorias no tempo e facilidades no PDS, não só durante a construção do software, mas também na sua manutenção.

Palavras-chave: Desenvolvimento de Software, BPMN, Geração de Código

ABSTRACT

The Software Engineering aims to support all aspects of software development, having as some of its challenges the reduction of delivery time and the improvements that facilitate the development of the software. Thus, it is necessary to create tools, methods and mechanisms that support and improve the software development process, including its maintenance, considering the changes of requirements over time. These maintenances can occur due to the evolution of technologies and the new needs of its users. In order to support the Software Development Process (SDP), code generation techniques and business process management have been widely used. With this focus, and aiming to improve the SDP, a set of techniques were researched which, combined, resulted in a development approach, whose bases are the Business Models and Persistent Object Classes and Code Generation. These techniques, widely used by developers today, integrated in the proposed approach, allowed to support and guide the developer in both the construction and maintenance of the software. The approach defines activities that should be performed during software development and employs the concepts, techniques, and tools that support those activities. The approach was evaluated through the development of an experimental study and it was verified that its use provided improvements in the time and facilities in the SDP, not only during the construction of the software, but also in its maintenance.

Keywords: Software Development, BPMN, Code Generation

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta a introdução ao tema deste trabalho destacando os objetivos e os desafios que motivaram a realização da pesquisa. A Seção 1.1 apresenta as motivações para a realização deste trabalho, bem como a Seção 1.2 descreve os objetivos, e por fim, a Seção 1.3 destaca a estrutura deste documento.

1.1 Motivação

Com o desenvolvimento de software orientado a objetos surgiram novos desafios em diferentes áreas da computação, e principalmente na Engenharia de Software. Diferentes arquiteturas surgiram, usando diferentes plataformas de software. As linguagens de programação orientadas a objetos evoluíram proporcionando mais facilidades e recursos para a implementação de software orientado a objetos.

Por outro lado, muitos dos sistemas de gerenciamento de banco de dados utilizados no mercado são Relacionais ou Relacionais Estendidos. Utilizando esses gerenciadores, os sistemas de software orientados a objetos precisam transformar os objetos em dados. Assim, Modelos de Objetos são convertidos em Modelos de Dados, e vice-versa, no armazenamento e recuperação das informações persistentes.

Para apoiar o Processo de Desenvolvimento Orientado a Objetos diversas tecnologias, ferramentas e outros recursos computacionais foram construídos. Atualmente, esse processo utiliza frameworks, bibliotecas e ferramentas que facilitam a conversão entre tais modelos. Dessa forma é possível implementar sistemas de software orientados a objetos com persistência em banco de dados.

Ao ser colocado em produção, é necessário a realização de contínuas manutenções no software de forma a adaptá-lo às novas funcionalidades e requisitos do ambiente inserido, para que o mesmo não se torne obsoleto. Em suma, é essencial manter o controle sobre as manutenções, com maior atenção no desempenho e capacidade de evolução do sistema em geral, possibilitando que novas funções não contidas no projeto inicial sejam implementadas com maior facilidade (PRESSMAN, 2010).

Diante do exposto, destaca-se a importância da criação de técnicas, ferramentas e abordagens, que apoiam as etapas do processo de desenvolvimento de software, desde o levantamento de requisitos, até a etapa de testes e manutenções do software em produção. Considerando a grande quantidade de técnicas e ferramentas disponíveis no mercado, é interessante conhecelas e utilizá-las, e se possível saber integrá-las corretamente no processo de desenvolvimento de software, para que possam facilitar as tarefas do desenvolvedor e melhorar a produção do software.

Outra motivação para essa pesquisa vem do crescente mercado de desenvolvimento de software que hoje conta com inúmeras ferramentas que facilitam a implementação de sistemas de software orientados a objetos, mas que muitas vezes não são integradas com as demais etapas, modelagem do banco de dados e dos processos de negócio, no processo de desenvolvimento de software.

1.2 Objetivo

Motivados por essas ideias, este trabalho tem como objetivo propor uma abordagem de desenvolvimento capaz de orientar o desenvolvedor na criação de um software flexível às mudanças, através da integração entre o modelo de classes de objetos persistentes e os modelos de negócio. A abordagem proposta, também, contribui para automatizar parte das tarefas do desenvolvedor por meio da técnica de metaprogramação na geração de código.

A abordagem proposta é dividida em três atividades. Na primeira atividade, o enfoque é construir o modelo de classes de objetos persistentes a partir do modelo físico do banco de dados. A segunda atividade tem como objetivo construir o modelo do processo de negócio a partir dos documentos de requisitos. Por fim, a terceira atividade é responsável por realizar a integração automática das funcionalidades CRUD com o BPMN, dando origem ao software implementado.

A abordagem proposta considera a arquitetura adotada por grande parte dos processos de desenvolvimento, e que em resumo tem suas bases em:

- Sistemas orientados a objetos e que têm suas persistências em banco de dados relacionais;
- Utilizam Modelos de Classes de Objetos Persistentes em Banco de Dados Relacionais, e adotam frameworks de persistência, para conversão do Modelo de Dados em Modelos de Objetos, e vice-versa; e
- Utilizam Modelos de Classes de Objetos não persistentes para especificar os requisitos de software que não persistem em banco de dados. Esses modelos complementam os modelos de classes de objetos persistentes.

É importante destacar essas premissas básicas para direcionar o leitor no entendimento da abordagem proposta, pois foi para essa arquitetura que foram desenvolvidos os estudos e pesquisas deste projeto. Destaque-se ainda que, essa arquitetura, normalmente é estruturada com o padrão MVC (MEDEIROS, 2015) bastante conhecido no mercado e adotado pelos diferentes processos de desenvolvimento de software.

1.3 Estrutura da Dissertação

Como mostra a Figura 1.1, esta dissertação está organizada em seis capítulos, incluindo este capítulo introdutório. O conteúdo de cada capítulo é brevemente descrito a seguir:

- O **Capítulo 2** apresenta a fundamentação teórica relacionada aos principais conceitos e técnicas abordados nesta pesquisa: Desenvolvimento de Software, Desenvolvimento Dirigido a Modelos, Modelos de Classes de Objetos Persistentes, Processo de Negócio e Metaprogramação;
- O **Capítulo 3** apresenta e detalha a abordagem proposta neste trabalho para apoiar o desenvolvimento de software. Cada uma das etapas e atividades do processo são definidas, apresentando suas respectivas entradas, saídas, controles e mecanismos que dão suporte aos desenvolvedores;

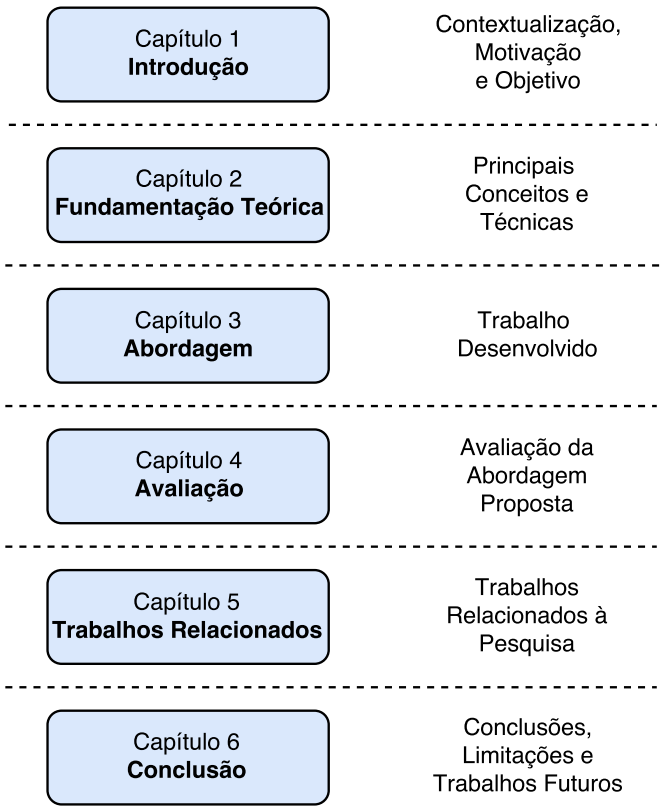

Figura 1.1: Organização da dissertação.

- O **Capítulo 4** trata de uma avaliação experimental realizada para analisar a aplicabilidade do processo proposto por uma população de desenvolvedores de software, representada por meio de um conjunto de alunos de pós-graduação;

- O **Capítulo 5** apresenta alguns trabalhos encontrados na literatura que se relacionam com o trabalho desenvolvido;

- O **Capítulo 6** discute as conclusões do trabalho desenvolvido, apresentando as contribuições, limitações e possíveis trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos relacionados a este projeto, destacando seus principais pontos. A Seção 2.1 apresenta os conceitos relacionados ao Desenvolvimento de Software. A Seção 2.2 apresenta os conceitos relacionados ao desenvolvimento dirigido a modelos. A Seção 2.3 descreve sobre modelo de classe de objetos persistentes. A Seção 2.4 apresenta sobre os conceitos de processo de negócio. A Seção 2.5 descreve sobre a técnica de metaprogramação. Finalizando, a Seção 2.6 apresenta algumas considerações finais referentes a este capítulo.

2.1 Desenvolvimento de Software

Pesquisas estão cada vez mais sendo realizadas na área de Engenharia de Software com o intuito de propor uma variedade de técnicas, métodos e ferramentas de apoio às empresas de desenvolvimento de software. As metodologias criadas orientam as equipes que produzem software, em meio aos desafios ocasionados pelas mudanças nas demandas dos usuários. Tais metodologias aumentam significativamente a flexibilidade almejada por essas constantes mudanças, durante o processo de desenvolvimento, permitindo que os desenvolvedores envolvidos execute-os com mais eficiência (ALTAMEEM, 2015).

Processo de Desenvolvimento de Software (PDS), ou processo de software, é o conjunto das atividades e diretrizes referentes ao desenvolvimento de sistemas computacionais. Embora não exista um processo ideal, grande parte das organizações desenvolvem suas próprias abordagens de processos de software, com as quais são complexas e dependem de pessoas para tomarem decisões (SOMMERVILLE, 2011).

Segundo Pressman (2010), um processo não é uma descrição rígida de como desenvolver

um software, e sim uma abordagem simples que faça com que toda a equipe envolvida compreenda e escolha as ações e tarefas a serem executadas, tendo sempre como intenção, entregarem o software com qualidade suficiente ao usuário e dentro do prazo estipulado. Existem diversos modelos de processo de desenvolvimento de software, e geralmente, incluem ao menos as quatro etapas descritas a seguir:

- **Análise**: Etapa de levantamento dos requisitos funcionais do sistema.
- **Projeto**: Etapa de planejamento da implementação do sistema, incluindo descrição, modelos, estruturas e cronograma de trabalho.
- **Implementação**: Etapa de conversão das especificações do sistema em um sistema executável.
- **Teste**: Etapa de verificação e validação do sistema com intuito de mostrar se o mesmo adéqua-se a suas especificações e as do usuário.

Os modelos de processos são representações simplificadas de um processo de desenvolvimento de software, nos quais segundo Sommerville (2011) são abstrações que podem ser usadas para explicar as diferentes abordagens de processo. O mesmo classifica os processos de software em duas categorias, dirigidos a planos ou processos ágeis, e destaca alguns modelos mais consolidados, o modelo em cascata, incremental e orientado a reúso.

Independente do modelo usado no desenvolvimento de software, as organizações buscam sempre ferramentas e técnicas para apoiarem as etapas do processo, visando automatizá-las e consequentemente minimizar os recursos humanos. De acordo com Papotti et al. (2012), ao automatizar o processo, torna-se possível que os esforços do desenvolvedor sejam filtrados para outras tarefas que não são passíveis de automatização, obtendo como ganho, eficiência na utilização dos recursos e um processo mais ágil. Desta forma, a técnica de geração de código torna-se um mecanismo bastante promissor.

Para contextualizar melhor, neste trabalho, o foco no ciclo de vida do PDS, foi nas etapas de Projeto e Implementação. Portanto, a etapa inicial de análise dos requisitos não foi explorada e sim considerada como realizada previamente. Também com relação às técnicas utilizadas, o PDS da abordagem proposta, concentrou-se na utilização de Gerador de Código, e teve suas bases nos Modelos de Negócio e de Classes de objetos Persistentes. Essa decisão foi motivada considerando que essas técnicas já vêm sendo utilizadas por grande parte dos desenvolvedores, incluindo a Secretaria de Informática da UFSCar, onde o orientador deste projeto trabalhou de 2010 a 2015.

2.2 Desenvolvimento Dirigido a Modelos

A modelagem sempre foi uma atividade essencial no processo de desenvolvimento de software, uma vez que a utilização de modelos com maior nível de abstração colabora para o entendimento de um sistema como um todo (RUMBAUGH et al., 2005). De fato, a facilidade em trabalhar com modelos contribui para a realização de debates e discussões entre os membros envolvidos no processo de desenvolvimento de software. No entanto, a utilização dos modelos na construção de um novo sistema pode ser ainda mais relevante, trazendo diversos benefícios.

Um modelo pode ser definido como uma descrição ou especificação das funcionalidades, estrutura e comportamento de um sistema e seu ambiente para determinado propósito, representado comumente pela combinação de elementos gráficos e textuais (MILLER; MUKERJI, 2003). Muitas vezes, essa combinação de elementos é realizada seguindo uma especificação formal, ou seja, tendo como base uma linguagem com sintaxe e semântica bem definidas, chamada de linguagem de modelagem.

Apesar de ocorrência de avanços a cada dia na Engenharia de Software, muitas dificuldades e problemas relacionados ao desenvolvimento de software são recorrentes até os dias atuais. Dentre esses problemas existentes, uma parte está relacionada com a dificuldade em manter a modelagem em conformidade com o estado atual do sistema. À medida que as manutenções são realizadas diretamente no código, um grande esforço é necessário para atualizar as mudanças na modelagem, de forma que, frequentemente, esta acaba sendo abandonada com o tempo.

Buscando solucionar esses problemas, o desenvolvimento dirigido a modelos (*Model-Driven Development* - MDD), também conhecido como MDE (*Model-Driven Engineering*) (SCHMIDT, 2006) ou MDSD (*Model-Driven Software Development*) (VöELTER; GROHER, 2007), tem sido pesquisado.

O MDD propõe que o engenheiro de software se concentre na elaboração de modelos de maior nível de abstração, ao invés de ter que realizar a interação manual com todo o código fonte do sistema. Nesse sentido, os modelos construídos, no desenvolvimento, não são usados somente na fase de compreensão dos requisitos do software, mas também possuem papel relevante na construção do sistema, por meio de mecanismos que realizam a geração total ou parcial do código do sistema a partir dos modelos (BITTAR et al., 2009).

A principal motivação para a utilização do MDD é melhorar a produtividade, ou seja, aumentar o retorno proveniente do esforço do desenvolvimento de software. Esse benefício é oferecido de duas maneiras aos desenvolvedores: o aumento da produtividade em curto prazo (realizada por meio da geração de funcionalidades do software) e o aumento da produtividade em

longo prazo (diminuindo a taxa com que o software se torna obsoleto) (ATKINSON; KüHNE, 2003).

2.2.1 Vantagens do uso do MDD

Entre as vantagens existentes na utilização do MDD (KLEPPE et al., 2003) (DEURSEN et al., 1998) (BHANOT et al., 2005) (MERNIK et al., 2005) (LUCRÉDIO, 2009), destacam-se:

- **Produtividade:** a curto prazo, menos tempo é dispendido para a construção dos modelos de alto nível e a aplicação de mecanismos de transformações de modelo, destinados à geração de código, automatizam parte das tarefas repetitivas no desenvolvimento. A longo prazo, o MDD provê uma diminuição da taxa com que o software se torna obsoleto e, dessa forma, menos tempo é gasto na realização de manutenções.
- **Portabilidade:** a partir de um mesmo modelo, é possível transformá-lo em código para a execução em diferentes plataformas.
- **Interoperabilidade:** cada parte do modelo pode ser transformada em código executável, que realiza comunicação entre diferentes plataformas, resultando num ambiente de execução heterogêneo e mantendo a funcionalidade global do software.
- **Manutenção e documentação:** ao contrário do que geralmente ocorre no desenvolvimento convencional, os modelos da especificação do software permanecem atualizados ao longo do tempo. As alterações no sistema são realizadas diretamente nos modelos, mantendo a documentação atualizada e tornando mais fácil as tarefas de manutenção do software.
- **Comunicação:** dado o alto grau de abstração existente nos modelos, os profissionais envolvidos têm maior facilidade para identificar e discutir as regras de negócio associadas ao sistema, não exigindo conhecimento técnico relativo à plataforma.
- **Reúso:** a reutilização não ocorre em nível de código-fonte, mas sim em nível de modelo. Um mesmo modelo pode ser reutilizado para gerar código para diferentes contextos, por meio de mecanismos de transformação.
- **Verificação e otimizações:** modelos oferecem mais recursos para que verificações semânticas e otimizações automáticas, específicas de cada domínio, possam ser executadas. Dessa forma, é possível realizar implementações mais eficientes e com menos erros semânticos.

- **Corretude:** a utilização de geradores resulta em um código testado e sem eventuais erros de sintaxe, garantindo que falhas não serão introduzidas dentro do software.

Ao encerrar essa seção destaca-se que, nesse projeto de pesquisa, foram adotadas ideias e conceitos, relacionada com MDD e Geração de Código. Assim, procurando tirar proveito das ideias do Desenvolvimento Dirigido por Modelos (MDD), a abordagem proposta emprega os Modelos de Negócios e de Classes de Objetos Persistentes que, combinados, orientam e melhoram o processo de desenvolvimento do software, inclusive na fase de sua manutenção.

2.3 Modelo de Classes de Objetos Persistentes

Um modelo de programação ou paradigma de programação é um conjunto de princípios, ideias, conceitos e abstrações utilizado para o desenvolvimento de uma aplicação. O modelo de programação mais adotado no desenvolvimento de sistemas corporativos é o modelo Orientado a Objetos (OO). Esse modelo é utilizado com o intuito de obter alguns benefícios específicos, em que normalmente, o principal benefício desejado é facilitar a manutenção das aplicações (PEREIRA, 2015).

Em geral, os conceitos do modelo de programação orientado a objetos diminuem a complexidade do desenvolvimento de sistemas que possuem como características grandes quantidades de funcionalidades desenvolvidas por uma equipe, ou que serão utilizados por um longo período de tempo e sofrerão alterações constantes.

Um objeto em programação orientada a objetos, é uma instância de uma classe que contém seus estados (valores de seus atributos) e comportamentos (métodos). O modelo de objetos é uma coleção de todas as definições de classes de uma aplicação que podem representar, por exemplo, elementos de interface do usuário, recursos do sistema, eventos da aplicação, abstrações dos conceitos de negócio, entre outros.

Os objetos que abstraem conceitos de negócio modelam o domínio onde a aplicação específica irá operar. Desta forma, eles são coletivamente chamados de Modelo de Objetos de Domínio, pois representam os principais estados e comportamentos da aplicação. Normalmente, tais objetos são compartilhados por vários usuários simultaneamente nos quais são armazenados e recuperados entre as execuções do sistema.

A capacidade desses objetos de permanecerem armazenados além do tempo de execução da aplicação é chamada de Persistência de Objetos. A persistência de objetos não é exclusividade

dos objetos contidos no modelo de domínio, porém a maioria dos objetos de uma aplicação encontra-se em tal modelo. Para que a aplicação possa utilizar os objetos em tempo de execução e ao final eles permanecerem armazenados para recuperá-los futuramente, é necessário que a persistência armazene o estado dos objetos em algum repositório, como, por exemplo, num Banco de Dados relacional.

Atualmente, com os *frameworks* de persistência como Hibernate[1], TopLink[2], e outros, é usual a utilização dos geradores de código para converter os objetos persistentes em dados e vice-versa. Assim, é possível a persistência, e demais operações básicas, denominadas *Create, Retrieve, Update and Delete* (CRUD) de objetos em Sistemas de Gerenciamento de Banco de Dados Relacionais. Essa conversão se faz necessária porque, embora a programação seja orientada a objetos, a persistência mais utilizada, ainda é em banco de dados relacionais ou relacionais estendidos.

2.4 Processo de Negócio

Inúmeras definições de Processo de Negócio (*Business Process* - BP) são encontradas na literatura, porém todos giram em torno da mesma descrição, em que são conjuntos de atividades ou tarefas conectadas entre si de forma organizadas para atingir objetivos específicos de negócios (RAGHU; VINZE, 2007). As organizações em geral utilizam deste recurso para especificarem a execução de todo o fluxo de negócio, tornando assim seus processos mais gerenciáveis.

Os processos de negócio precisam ser flexíveis e adaptáveis para acompanhar as mudanças rápidas das condições de mercado (BITKOWSKA, 2012). Para isso, originou-se uma área de pesquisa denominada Gerenciamento de Processo de Negócio (*Business Process Management* – BPM), com a qual preocupa-se em criar métodos, técnicas e ferramentas para apoiar a concepção, a criação, gestão e análise de processos de negócios operacionais (AALST, 2013).

Uma gestão adequada dos processos de negócio de uma organização é muito importante para obter resultados mais eficazes e eficientes. Para isso é necessário que a equipe de gestores se atentem aos recursos disponíveis e saibam utilizá-las de acordo com suas necessidades. Neste contexto, a técnica de modelagem no gerenciamento de processos de negócio é essencial, pois facilita a compreensão e a comunicação sobre os processos organizacionais, e servindo também

[1] Hibernate - http://hibernate.org/
[2] TopLink - http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

como base para realizar a análise e melhorias nos processos em questão (ALVES et al., 2013).

Os modelos de processo de negócio podem ser utilizados tanto em áreas gerenciais quanto em técnicas, como por exemplo na configuração dos sistemas de informação, em que podem permitir melhorias significativas de produtividade, redução de custos e redução de tempo de fluxo (AALST, 2013). Sendo assim, diversas notações gráficas baseadas em fluxograma foram criadas para especificações de processo de negócio. As principais notações atualmente são: Diagrama de Atividades da *Unified Modeling Language* (UML), *Event-driven Process Chains* (EPC) e o BPMN, o qual é o foco deste trabalho.

2.4.1 *Business Process Model and Notation* (BPMN)

O BPMN é um padrão para modelagem de fluxos de processos de negócio, onde seu principal objetivo é fornecer uma notação que seja facilmente compreensível pelos usuários de negócios, incluindo os analistas que esboçam os rascunhos iniciais dos processos para os desenvolvedores técnicos, responsáveis pela implementação da tecnologia que irá executar e gerenciar tais processos (OMG, 2011).

Baseado no Diagrama de Atividade da UML, o BPMN foi desenvolvido pela *Business Process Modeling Initiative* (BPMI) em 2004 como uma notação gráfica para representar processos de negócio, e posteriormente adotado como padrão *Object Management Group* (OMG) em 2006 pela crescente utilização por parte de empresas (CHINOSI; TROMBETTA, 2012).

Um outro objetivo do BPMN é assegurar que as linguagens *eXtensible Markup Language* (XML) projetadas para a execução de processos de negócio, como o *Web Services Business Process Execution Language* (WSBPEL) e *Business Process Modeling Language* (BPML), possam ser visualmente expressos com uma notação comum (OWEN; RAJ, 2003).

A modelagem em BPMN é feita por meio de diagramas com um pequeno conjunto de elementos gráficos. Os diagramas são simples e intuitivos, no entanto, capazes de representar processos complexos. Isso facilita para que os usuários de negócio e desenvolvedores entendam o fluxo e o processo. A Figura 2.1 ilustra um exemplo simples de um diagrama BPMN representando o processo de aprovação de férias de uma determinada organização.

O retângulo mais externo que contém todo o fluxo é denominado *Pool*. Os três retângulos internos ao *Pool* são denominados *Lane* e são utilizados para organizar as atividades entre os diversos participantes do processo. As circunferências são representações de eventos, onde a circunferência localizada ao lado esquerdo representa o evento inicial e ao lado direito repre-

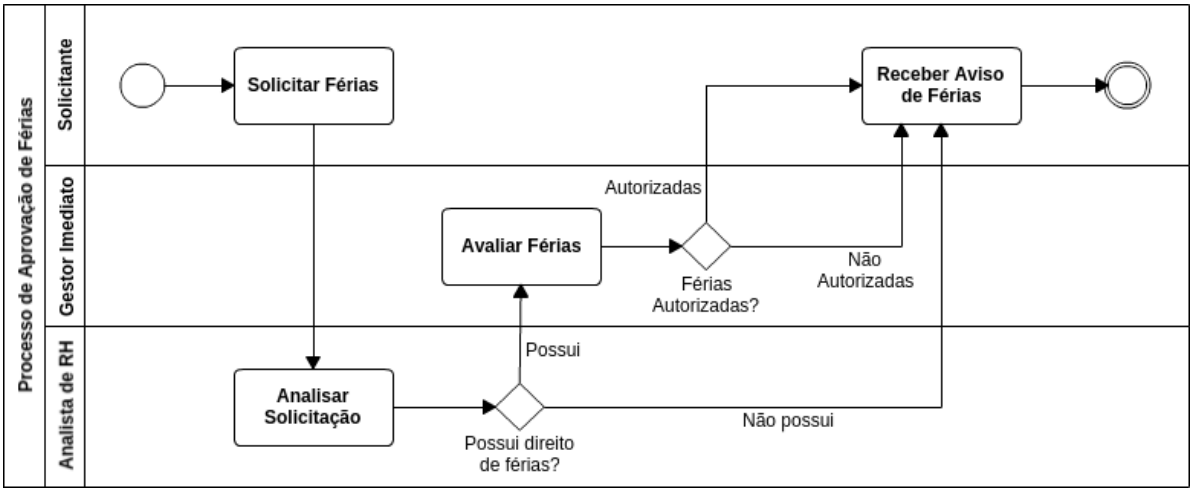

Figura 2.1: Exemplo de um modelo BPMN representando o processo de aprovação de férias.

senta o evento final. As atividades do processo são representadas pelos retângulos menores em que são conectados por setas que determinam o fluxo a ser seguido. Por fim, os losangos são as representações dos *gateways*, os quais são decisões a serem tomadas no decorrer do processo, definindo qual caminho a ser seguido.

Existem diversos tipos de representações das atividades, bem como dos *gateways* e eventos, onde cada um determina sua função no processo. Tais representações e suas definições podem ser encontradas em OMG (2011).

Atualmente existem várias ferramentas *Computer-Aided Software Engineering* (CASE) que permitem a construção, execução e monitoramento de modelos de processos de negócio, como por exemplo, jBPM[3], Bizagi[4], entre outras. Para a realização deste trabalho, será utilizado a ferramenta de modelagem Camunda Modeler[5]. Algumas ferramentas também possibilitam a criação dos diagramas em um editor gráfico por meio de um arquivo XML, seguindo as especificações do BPMN.

2.5 Metaprogramação

Por muitos anos, a metaprogramação foi conhecida e utilizada sobretudo na programação lógica formal (SHEARD, 2001). Atualmente, o escopo da aplicação das técnicas de metaprogramação é bem mais amplo, tais como: implementação de linguagem de programação e compiladores (CHLIPALA, 2010); geradores de software e aplicações (ŠTUIKYS et al., 2015);

[3]jBPM - http://www.jbpm.org/
[4]Bizagi - http://www.bizagi.com/
[5]Camunda Modeler - http://camunda.org/bpmn/tool/

linhas de produto (BATORY, 2006); transformações de programas (PALMER; SMITH, 2011); manutenção, evolução e configuração de software (CZARNECKI; EISENECKER, 2000b), dentre outras.

Metaprogramação pode ser definida como sendo uma técnica de programação, na qual um programa de computador é escrito para gerar código ou manipular outro programa ou a si mesmo, visando solucionar uma dada tarefa (CORDY; SHUKLA, 1992). Essa técnica é largamente utilizada no ciclo de desenvolvimento de software, tendo papel essencial nos processadores de programas, interpretadores e compiladores. Além disso, como abordagem conceitual, está em constante evolução e seus princípios são adaptados a níveis de abstração cada vez mais altos, como por exemplo, o MDD (SCHMIDT, 2006).

Outro conceito relacionado ao de metaprogramação é o de metalinguagem. Qualquer linguagem ou sistema simbólico usado para discutir, descrever, analisar outro sistema de linguagem simbólica é uma metalinguagem (CZARNECKI; EISENECKER, 2000a). Dessa forma, uma metalinguagem fornece uma estrutura formal para a escrita de programas, denominados metaprogramas. Dentre outros, alguns exemplos de metaprogramas são geradores de aplicação e geradores de analisadores (parsers).

Exemplos de metaprograma são os *makefiles* (programas que produzem outros programas). Um *makefile* é um programa que opera em um nível de abstração maior, sendo composto por um ou mais scripts que criam arquivos pela execução de ferramentas em uma ordem específica, visando manter a consistência desses arquivos na ocorrência de modificações (BATORY, 2006).

De modo geral, os metaprogramas podem ser divididos em duas categorias: metaprogramas estáticos ou metaprogramas dinâmicos. Os primeiros são construídos por meio da metaprogramação estática, a qual permite a escrita de código que é executado pelo compilador em tempo de compilação, ou seja, antes da execução do programa de fato. Um exemplo desse tipo de metaprogramação é o uso dos mecanismos de *templates* na linguagem C++. Já os metaprogramas dinâmicos são criados por meio da metaprogramação dinâmica, de modo que o conhecimento das definições das estruturas do programa gerado ocorre em tempo de execução. Um exemplo desse tipo de metaprogramação é o suporte à Reflexão da linguagem Java, que permite um programa listar métodos, construtores, atributos das classes e outros elementos durante sua execução (REEUWIJK, 2003).

2.6 Considerações Finais

Este capítulo apresentou uma revisão da literatura sobre os principais conceitos e técnicas que serviram de base para o trabalho desenvolvido. Ao longo do capítulo, foram abordados os conceitos fundamentais relacionados ao processo de desenvolvimento de software, desenvolvimento dirigido a modelos e suas vantagens, processo de negócio e a técnica de modelagem BPMN, modelo de objetos persistentes e sobre a metaprogramação. Esta revisão auxilia no entendimento do trabalho elaborado, apresentado no capítulo a seguir.

Capítulo 3

ABORDAGEM

Este capítulo descreve a abordagem desenvolvida neste trabalho. A Seção 3.1 descreve as considerações iniciais do capítulo; a Seção 3.2 e suas Subseções (3.2.2, 3.2.1 e 3.2.3) apresentam de forma detalhada a abordagem proposta e todas suas atividades, ferramentas e técnicas utilizadas em suas etapas; por fim, a Seção 3.4 descreve as considerações finais deste capítulo.

3.1 Considerações Iniciais

Dentro das etapas do PDS, a abordagem proposta tem foco no Projeto e Implementação, definindo atividades que baseiam em dois principais artefatos: Modelo de Classes de Objetos Persistentes e Modelo do Negócio. A abordagem reúne conceitos e técnicas que orientam e apoiam o desenvolvedor, visando facilitar o PDS na redução do tempo gasto com a construção e evolução do software.

A utilização de geradores de código podem oferecer diversos benefícios no processo de desenvolvimento, entre eles a diminuição de erros, ganho de tempo e principalmente facilitar o reúso e manutenção do sistema, mantendo a qualidade e o ciclo de vida do software. A automatização de parte das tarefas do desenvolvedor, permite que seus esforços sejam focados em outras tarefas mais importantes no processo de desenvolvimento e proporciona facilidades nas manutenções futuras do software.

3.2 Abordagem

A Figura 3.1 ilustra a visão geral da abordagem, representada com a notação de diagramas *Structured Analysis and Design Technique* (SADT) (ROSS, 1977). Na notação utilizada, cada atividade pertencente a abordagem é representada por retângulos, onde as setas vindas do lado esquerdo dos retângulos representam os dados de entrada das atividades. As setas na parte superior do retângulo, representam os controles que orientam as atividades. Por sua vez, as setas na parte inferior correspondem aos mecanismos e ferramentas utilizados durante a atividade, e por fim as setas que saem dos retângulos sentido ao lado direito, representam as saídas das atividades.

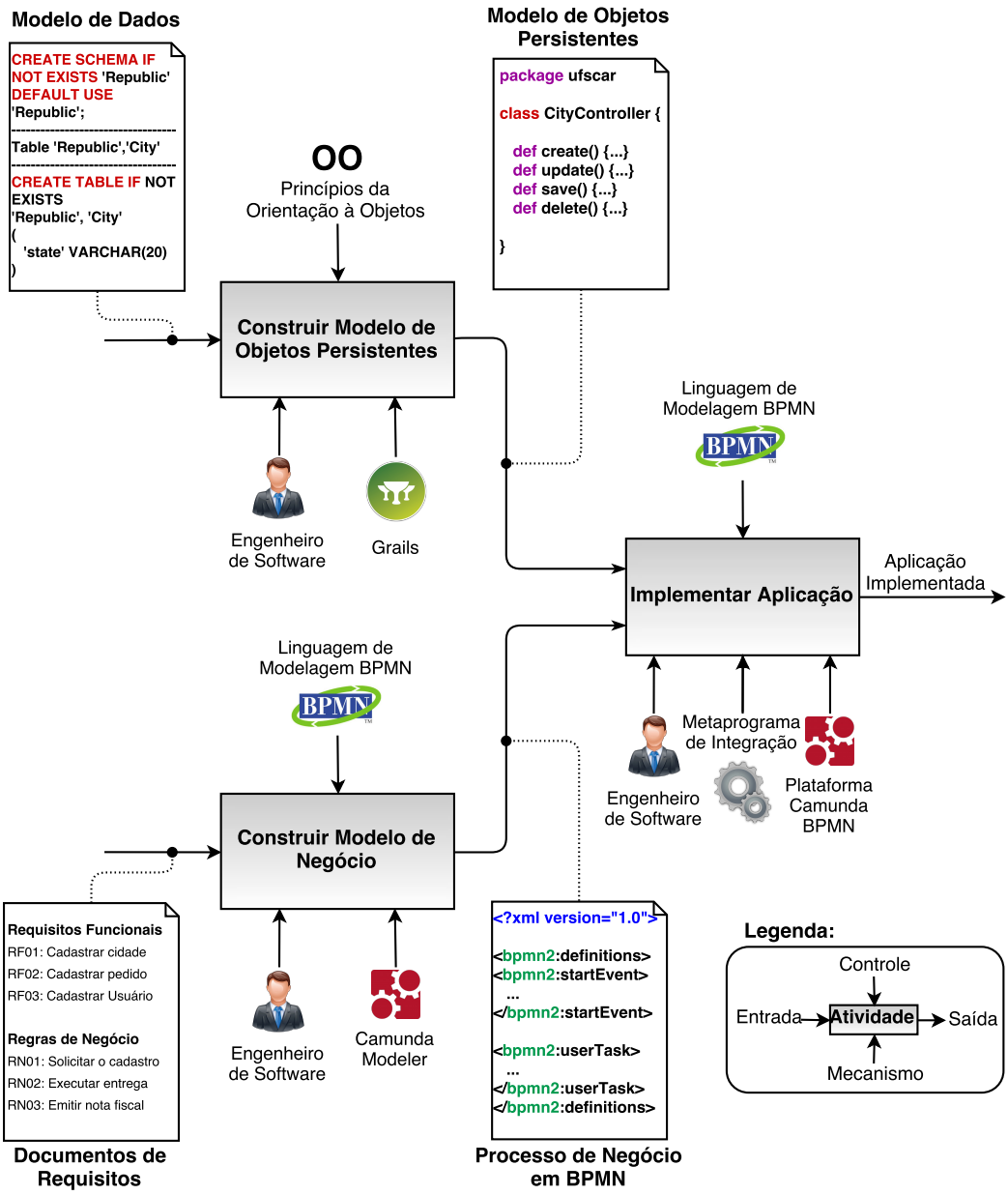

Figura 3.1: Visão geral da abordagem proposta.

Como mostra a Figura 3.1, a abordagem proposta têm três atividades: **Construir Modelo de Classes de Objetos Persistentes**, **Construir Modelo de Negócio** e **Implementar Aplicação**, as quais serão apresentadas nas Seções 3.2.1, 3.2.2 e 3.2.3 respectivamente.

Para facilitar o entendimento da abordagem proposta, a execução de suas atividades, é ilustrada com o desenvolvimento de uma aplicação, no domínio de Controle Bancário, apresentada em Beder (2016). Considerando que a fase de Análise de Requisitos não é tratada pela abordagem, as atividades dessa fase foram previamente realizadas e seus artefatos foram tornados disponíveis para uso da abordagem.

A aplicação que ilustra o uso da abordagem proposta, foi desenvolvida para servir de exemplo nos cursos lato-sensu de Desenvolvimento de Software para Web oferecidos no Departamento de Computação da UFSCar, onde o orientador desse projeto de pesquisa atua como pesquisador. Essa aplicação tem foco no gerenciamento bancário, controlando suas agências e possibilitando que seus gerentes e clientes realizem transações em contas correntes ou poupanças. As transações realizadas pelo cliente podem ser de retirada de valor de suas contas, de depósito e de transferência de valores entre contas de um mesmo banco.

Nessa aplicação, um cliente é qualquer pessoa física ou jurídica que abre uma conta em uma ou mais agências de um banco operado por essa aplicação. Assim as contas estão vinculadas às diferentes agências em diferentes endereços de várias cidades e estados. Uma conta é associada sempre a um cliente titular e a zero ou mais outros clientes. Desta forma, foram identificados os requisitos descritos resumidamente a seguir:

Administrador:

01 - Como administrador devo cadastrar os estados;

02 - Como administrador devo cadastrar as cidades;

03 - Como administrador devo cadastrar os bancos;

04 - Como administrador devo cadastrar as agências

05 - Como administrador devo cadastrar os gerentes;

06 - Como administrador devo cadastrar os caixas eletrônicos.

Gerente:

01 - Como gerente desejo cadastrar um cliente físico ou jurídico;

02 - Como gerente desejo abrir uma conta corrente ou poupança;

03 - Como gerente desejo realizar uma transação em uma conta cliente;

04 - Como gerente quero uma lista de todos os clientes ativos ou inativos;

05 - Como gerente quero uma lista de todos os clientes físicos ou jurídicos;

06 - Como gerente quero uma lista de todas as contas correntes ou poupanças;

07 - Como gerente quero uma lista de todas as contas por data de abertura;

08 - Como gerente quero uma lista de todos os clientes titulares;

09 - Como gerente quero uma lista de todos os clientes com saldo positivo ou negativo;

10 - Como gerente quero uma lista de todos os clientes por ordem alfabética.

Cliente:

01 - Como cliente desejo obter um extrato da minha conta;

02 - Como cliente desejo efetuar uma transação em minha conta.

Além dessa informações, ainda nessa fase de Análise de Requisitos é construído o modelo do banco de dados da aplicação. A etapa de modelagem do banco de dados, embora importante, não foi alvo da abordagem por ser já bastante conhecida dos desenvolvedores. Assim, para a aplicação de Controle Bancário foi construído o MER (Figura 3.2) e, a partir desse modelo, foi gerado o código SQL e criado o banco de dados da aplicação.

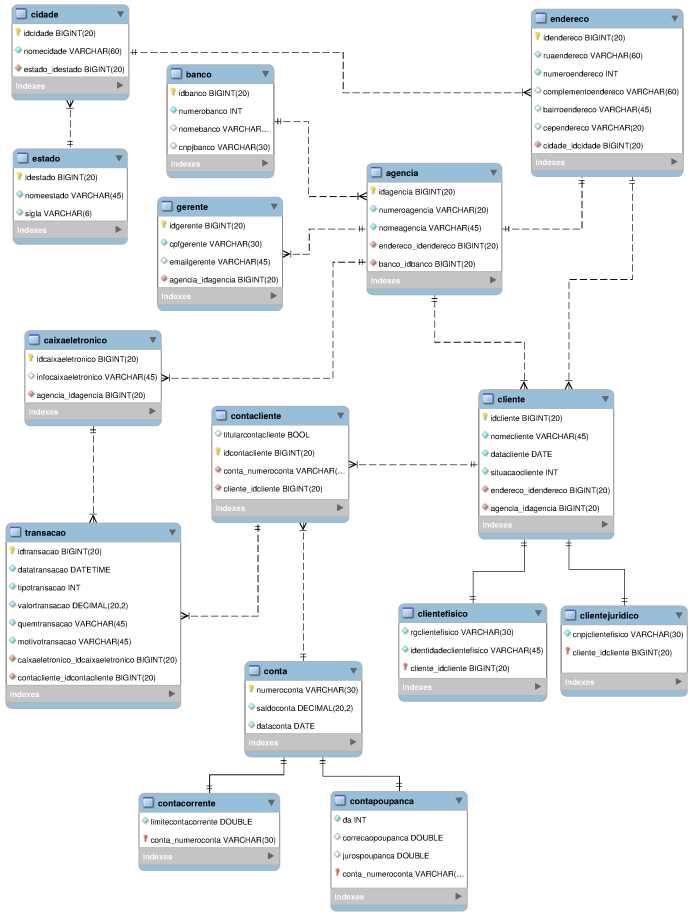

Figura 3.2: MER controle bancário

Para realizar esse processo de modelagem, no qual resultou o MER ilustrado na Figura 3.2, utilizou-se a ferramenta MySQL Workbench[1], que permite que um DBA , desenvolvedor ou engenheiro de software visualmente crie, modele e gerencie o banco de dados.

Com base nas informações e nos artefatos especificados na Análise dos Requisitos da aplicação, incia-se a abordagem proposta, construindo o modelo de classes de objetos persistentes, como é apresentado a seguir.

3.2.1 Construir Modelo de Classes de Objetos Persistentes

O objetivo dessa atividade é construir o modelo de classes de objetos persistentes a partir do modelo de dados. O modelo de classes de objetos persistentes é obtido extraindo informações do modelo físico do banco de dados, previamente construído a partir do MER especificado para o sistema de software.

No modelo de objetos, são representados as classes de objetos da camada *Model* do padrão arquitetural *Model-View-Controller* (MVC) (HATANAKA; HUGHES, 1999). O padrão MVC tem sido adotado em grande parte das arquiteturas de software, e visa facilitar o entendimento e manutenção do software. Destaca-se que, esse modelo de classes de objetos representa, no paradigma OO, as entidades persistentes no Banco de Dados. Esses modelos de objetos têm suas correspondentes implementações nas linguagens OO, como por exemplo Groovy, adotada na abordagem proposta. Dessa forma, têm-se um código descrito com Classes, Atributos e Relacionamentos, correspondentes às Tabelas, Campos e Relacionamentos das Entidades Persistidas no Banco de Dados Relacional.

Na execução dessa atividade da abordagem, o Engenheiro de Software utiliza-se de um *framework* de persistência para acessar a estrutura física do banco de dados da aplicação, e gerar o correspondente código das classes dos objetos persistentes na linguagem de programação OO. Essas classes são denominadas de classes de domínio e são organizadas na camada *Model*. Essa camada fornece as bases para o desenvolvimento das demais camadas *Controller* e *View*, do padrão MVC. A Figura 3.3 ilustra a execução dessa atividade, na qual o código da linguagem Groovy da classe de domínio Endereço, é gerado a partir de sua respectiva tabela do banco de dados.

Na abordagem proposta, adotou-se como *framework* de persistência um *plug-in* específico do *framework* Grails[2] denominado **db-reverse-engineer**[3]. Ressalte-se que existem diversas

[1] Workbench - https://www.mysql.com/products/workbench
[2] Grails - https://www.grails.org/
[3] db-reverse-engineer - http://www.grails.org/plugin/db-reverse-engineer

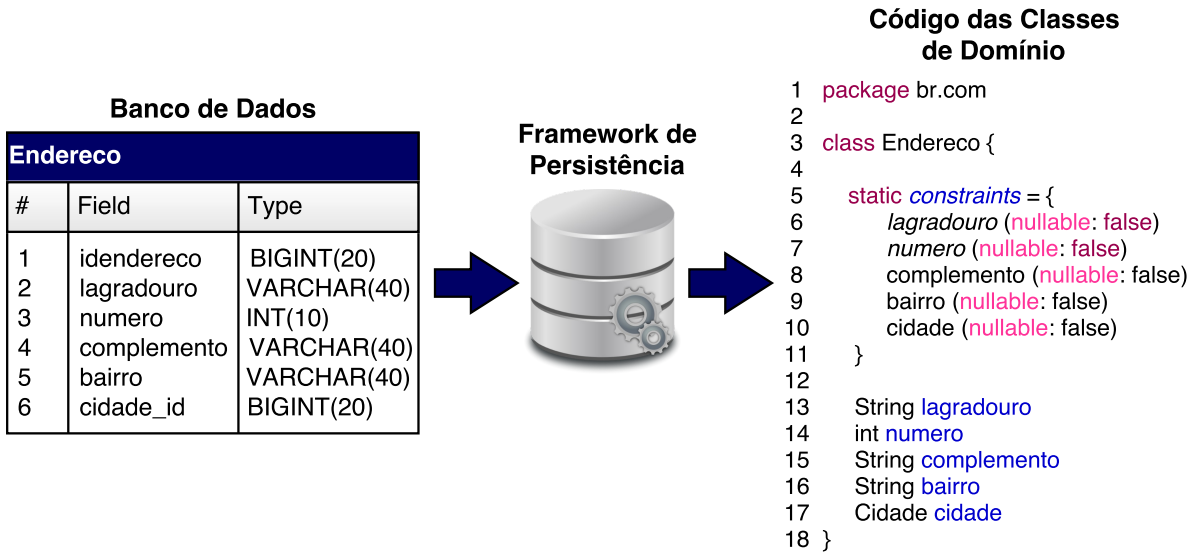

Figura 3.3: Geração de uma classe de domínio a partir do banco de dados.

ferramentas de persistência disponíveis para outras linguagens de programação.

O *plug-in* **db-reverse-engineer**, é responsável por realizar a engenharia reversa, lendo as informações das tabelas do banco de dados e criando as classes de domínio da aplicação. A utilização desse *plug-in* traz como benefício o ganho de tempo para implementar tais códigos gerados e permite uma facilidade maior em manutenções futuras, uma vez que alterado ou acrescentado alguma tabela do banco, é necessário apenas executar novamente o *script* para realizar as alterações.

O Trecho de código 3.1, apresenta o código Groovy apresentado na Figura 3.3, gerado da tabela Endereço do banco de dados da aplicação após a execução do *plug-in*. O mesmo extraiu todas as informações do SQL do banco e gerou o código da classe de domínio de sua respectiva tabela.Essa geração das classes ocorre, ao mesmo tempo, para todas as tabelas do banco de dados. No final, tem-se todas as classes de domínio, correspondentes às tabelas do banco de dados da aplicação.

```
1 package br.ufscar.dc.dsw
2
3 class Endereco {
4
5     static constraints = {
6         logradouro (blank: false, size: 1..40)
7         numero (min: 0)
8         complemento (nullable: true, size: 1..40)
```

```
9           bairro (blank: false, size: 1..40)
10          cidade (nullable: false)
11      }
12
13      String logradouro
14      int numero
15      String complemento
16      String bairro
17      Cidade cidade
18
19      String toString() {
20          return logradouro + ", " + numero +
21                  (complemento == null ? "" : " " + complemento) + ". " +
22                  bairro + " " + CEP + " " + cidade
23      }
24  }
```

Código 3.1: Classe de Domínio Endereço

Os códigos das classes de domínio gerados nessa atividade, serão utilizados como base para implementar as camadas *Controllers* e *Views* do padrão MVC, completando uma implementação básica que possibilita realizar as funcionalidades CRUD, responsáveis pela criação e manutenção dos objetos persistentes no banco de dados relacional.

3.2.2 Construir Modelo de Negócio

O objetivo dessa atividade é modelar os processos de negócio com base nos documentos de requisitos, previamente especificados na etapa de análise da aplicação. Apoiam essa atividade ferramentas que permitem a construção de modelos de processos de negócio.

Na Figura 3.4 é ilustrada a execução da atividade, na qual o Engenheiro de Software recebe como entrada o documento de requisitos e, com o conhecimento da linguagem de modelagem BPMN, modela o processo de negócio em uma ferramenta CASE, para que seja integrado posteriormente com o código do software, funcionando como o fluxo de execução do sistema.

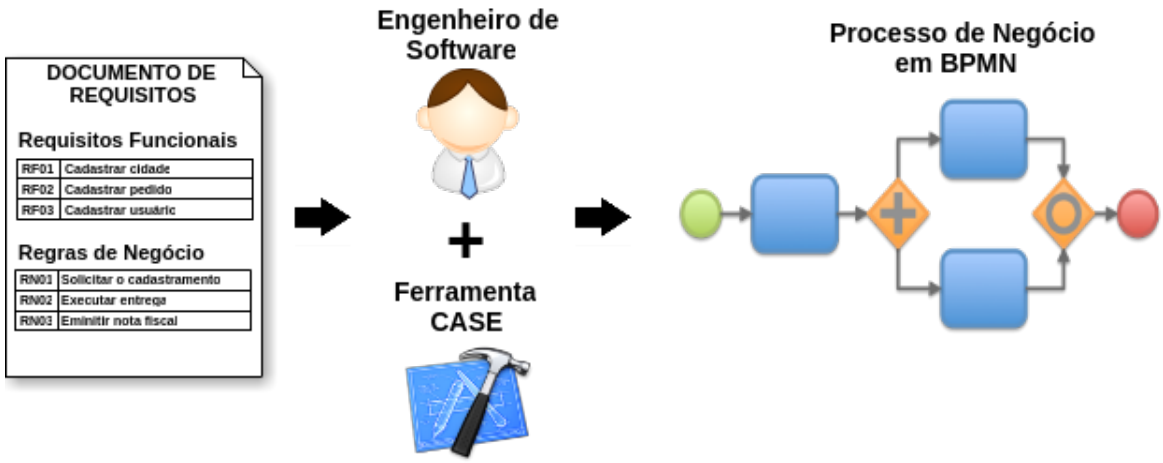

Figura 3.4: Construção do modelo de processo de negócio.

Nessa atividade, o desenvolvedor é apoiado por uma ferramenta CASE, denominada Camunda Modeler, para modelar o processo de negócio. A Figura 3.5, mostra o modelo de negócio do processo de cadastro de um gerente pelo administrador da aplicação bancária. Para cadastrá-lo, obrigatoriamente o administrador deverá cadastrar também a cidade, o endereço, a agência, e seu banco.

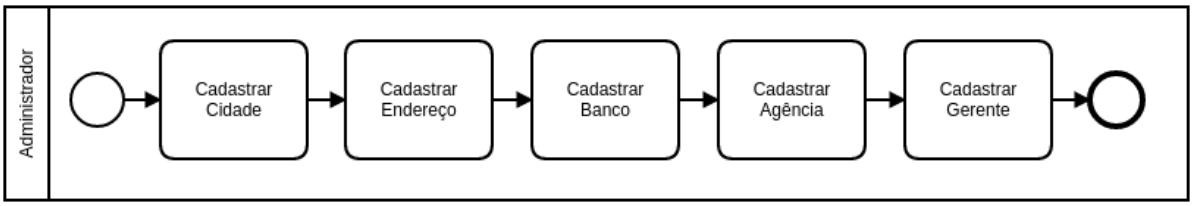

Figura 3.5: Processo de negócio cadastrar gerente bancário.

O modelo de processo de negócio gerado de acordo com o padrão BPMN é descrito em XML, e é utilizado na atividade seguinte da abordagem para orientar a implementação do processo. Essa implementação considera o código já produzido na atividade que constrói o modelo de classes de objetos persistentes. O código XML do processo ilustrado na Figura 3.5 é apresentado no trecho 3.2 abaixo.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpmn:definitions
3 xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
4 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
5 xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
6 xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
7 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="Definitions_1"
```

```
8  targetNamespace="http://bpmn.io/schema/bpmn"
9  exporter="Camunda Modeler" exporterVersion="1.6.0">
10   <bpmn:collaboration id="Process_CadGerente">
11     <bpmn:participant id="Participant_Administrador"
12     name="Administrador" processRef="Process_CadastrarGerente" />
13   </bpmn:collaboration>
14   <bpmn:process id="Process_CadastrarGerente"
15   name="Cadastrar Gerente Bancário" isExecutable="false">
16     <bpmn:startEvent id="Start_CadGer">
17       <bpmn:outgoing>SequenceFlow_1</bpmn:outgoing>
18     </bpmn:startEvent>
19     <bpmn:task id="Task_CadCid" name="Cadastrar Cidade">
20       <bpmn:incoming>SequenceFlow_1</bpmn:incoming>
21       <bpmn:outgoing>SequenceFlow_2</bpmn:outgoing>
22     </bpmn:task>
23     <bpmn:task id="Task_CadEnd" name="Cadastrar Endereço">
24       <bpmn:incoming>SequenceFlow_2</bpmn:incoming>
25       <bpmn:outgoing>SequenceFlow_3</bpmn:outgoing>
26     </bpmn:task>
27     <bpmn:task id="Task_CadBan" name="Cadastrar Banco">
28       <bpmn:incoming>SequenceFlow_3</bpmn:incoming>
29       <bpmn:outgoing>SequenceFlow_4</bpmn:outgoing>
30     </bpmn:task>
31     <bpmn:task id="Task_CadAg" name="Cadastrar Agência">
32       <bpmn:incoming>SequenceFlow_4</bpmn:incoming>
33       <bpmn:outgoing>SequenceFlow_5</bpmn:outgoing>
34     </bpmn:task>
35     <bpmn:task id="Task_CadGer" name="Cadastrar Gerente">
36       <bpmn:incoming>SequenceFlow_5</bpmn:incoming>
37       <bpmn:outgoing>SequenceFlow_6</bpmn:outgoing>
38     </bpmn:task>
39     <bpmn:endEvent id="EndEvent_CadGer">
40       <bpmn:incoming>SequenceFlow_6</bpmn:incoming>
41     </bpmn:endEvent>
42     <bpmn:sequenceFlow id="SequenceFlow_1" sourceRef="Start_CadGer"
```

```
43      targetRef="Task_CadCid" />
44      <bpmn:sequenceFlow id="SequenceFlow_2" sourceRef="Task_CadCid"
45      targetRef="Task_CadEnd" />
46      <bpmn:sequenceFlow id="SequenceFlow_3" sourceRef="Task_CadEnd"
47      targetRef="Task_CadBan" />
48      <bpmn:sequenceFlow id="SequenceFlow_4" sourceRef="Task_CadBan"
49      targetRef="Task_CadAg" />
50      <bpmn:sequenceFlow id="SequenceFlow_5" sourceRef="Task_CadAg"
51      targetRef="Task_CadGer" />
52      <bpmn:sequenceFlow id="SequenceFlow_6" sourceRef="Task_CadGer"
53      targetRef="EndEvent_CadGer" />
54    </bpmn:process>
55  </bpmn:definitions>
```

Código 3.2: XML do processo de cadastro de gerente.

3.2.3 Implementar Aplicação

Esta atividade é responsável pela implementação da aplicação. Esse código é obtido pela integração do código oriundo do Modelo de Classes de Objetos Persistentes, com o código oriundo do Modelo de Processos de Negócio descrito em BPMN. Essa etapa da abordagem é apoiada por uma ferramenta denominada *Engine* BPMN, que oferece recursos para executar e monitorar o software de acordo com os processos de negócio descritos em BPMN.

Inicialmente, são implementados os Controladores e as Visões das classes de domínio, obtidas na atividade Construir Modelos de Classes de Objetos Persistentes. Como mostra a Figura 3.6, o Engenheiro de Software utiliza o modelo de classes de objetos persistentes, obtido anteriormente e, com a ajuda de uma IDE, aplica a técnica de geração de código. Essa geração de código, na abordagem proposta é apoiada pelo *Framework* Grails. Para todas as classes de domínio anotadas como classes de entidades, normalmente organizadas no pacote *Domain*, são geradas as classes das camadas *Controller* e *View*.

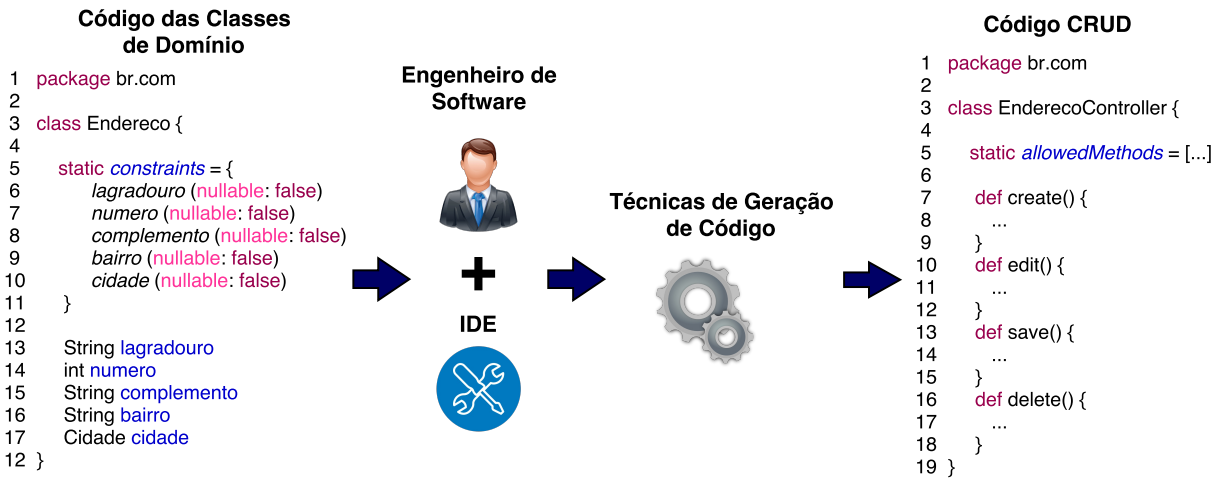

Figura 3.6: Geração das funcionalidades CRUD do software.

Essa geração inicial de código implementa as funcionalidades CRUD, dos objetos persistentes em banco de dados. Essas implementações iniciais, podem ser, ao longo do desenvolvimento, modificadas, dependendo dos requisitos da aplicação. Contudo, tem-se um ganho inicial de produtividade e também de estruturação do código segundo o padrão estrutural MVC.

Na abordagem proposta, adotou-se para a geração do código dos controladores e das visões, o gerador **scaffolding** que é um termo cunhado pelo *framework* Rails[4] e adotado pelo Grails para a geração dos artefatos (controladores, visões, etc.) que implementam as operações CRUD.

O trecho de código 3.3 apresenta o código Groovy da classe EnderecoController.groovy gerado pelo **scaffolding** a partir do código *domain*, e na Figura 3.7 sua respectiva *view* modificada, na qual também foi gerada pelo **scaffolding**.

```
1  package br.ufscar.dc.dsw
2
3  class EnderecoController {
4
5      static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]
6
7      def index(Integer max) {
8          params.max = Math.min(max ?: 10, 100)
9          respond Endereco.list(params),
10         model:[enderecoInstanceCount: Endereco.count()]
11     }
```

[4]Rails - http://rubyonrails.org/

```
12      def show(Endereco enderecoInstance) { ... }
13      def create() { ... }
14      def save(Endereco enderecoInstance) { ... }
15      def edit(Endereco enderecoInstance) { ... }
16      def update(Endereco enderecoInstance) { ...}
17      def delete(Endereco enderecoInstance) { ... }
18  }
```

Código 3.3: Classe Controladora de Endereço

Figura 3.7: *View* modificada da classe Endereço.

Após obtido os códigos das funcionalidades CRUD bem como o código XML do modelo de negócio, é necessário integrá-los para que a *engine* BPMN execute a aplicação. Como ilustra a Figura 3.8, o processo de integração é realizado manualmente pelo Engenheiro de Software, atribuindo a cada atividade do modelo de processo de negócio uma tarefa a ser executada pelo sistema. Assim que forem concluídas as atribuições, a ferramenta se encarregará de executar a aplicação de acordo com o fluxo do modelo especificado.

Visando melhorar a produtividade, e facilitar a manutenção do software, propõe-se um pro-

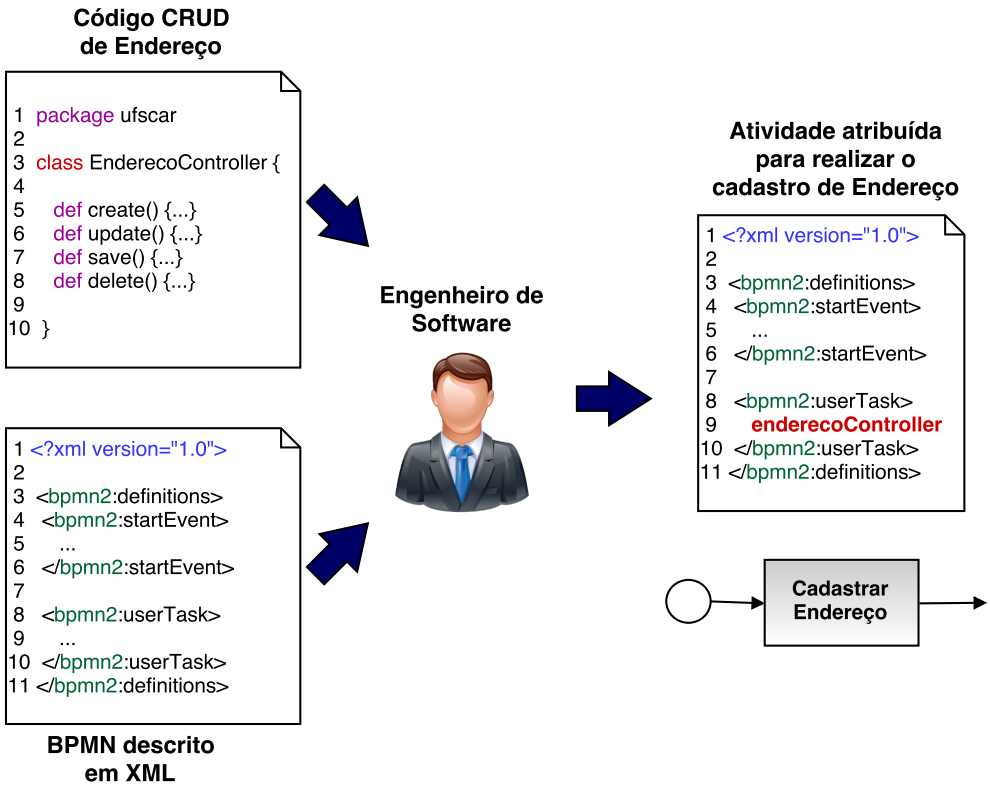

Figura 3.8: Integração manual entre o código do software com o modelo de processo de negócio.

cesso para diminuir o trabalho manual do desenvolvedor e consequentemente reduzir possíveis erros. Assim, a abordagem proposta possibilita a integração de forma automática entre o código da aplicação e o XML do modelo BPMN.

Para viabilizar essa integração, utilizou-se a metaprogramação como técnica de geração de código. Um metaprograma foi criado especificamente para este trabalho, com o objetivo de realizar uma busca por todas as atividades de usuário no XML do modelo BPMN, atribuindo para cada uma, seus respectivos códigos das funcionalidades CRUD, organizados no pacote *Controllers*.

Para que o metaprograma seja utilizado é necessário, ao modelar o BPMN, nomear os IDs das atividades (*userTask*) com as mesmas identificações especificadas nas classes de domínio. Desta forma, o metaprograma ao identificar o ID de uma *userTask*, percorrerá o XML, e adicionará seu correspondente código da camada *controller*. Essa notação possibilita ao metaprograma buscar por todos os IDs contidos no XML, e definir seus respectivos códigos. Posteriormente, a *engine* BPMN executa a aplicação de acordo com essas atribuições. Um exemplo é ilustrado na Figura 3.9.

A Figura 3.9 ilustra o uso dessa notação no desenvolvimento da aplicação de controle

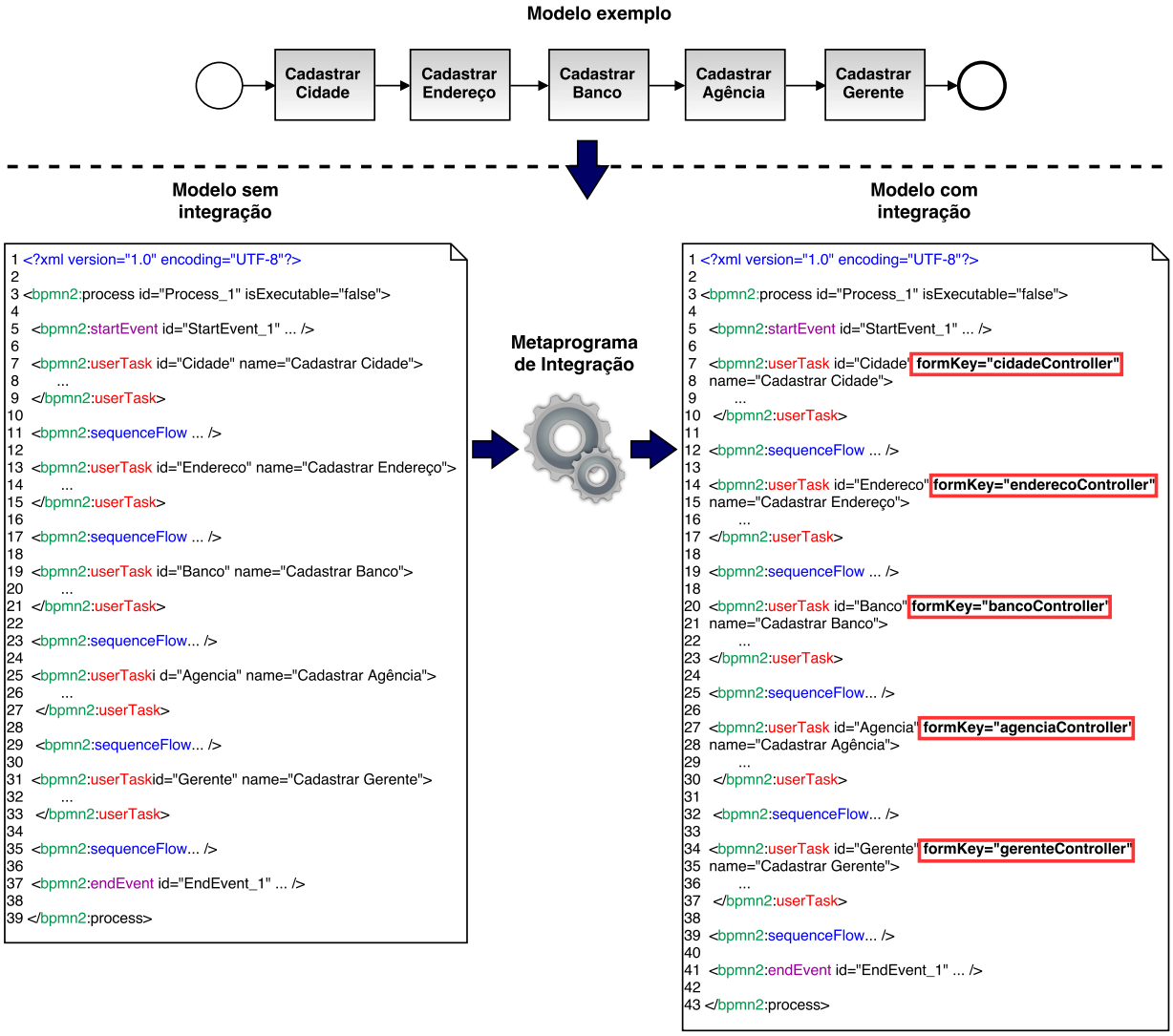

Figura 3.9: Integração automática utilizando metaprograma.

bancário. Percorrendo o Código 3.2, o metaprograma identificou a ***userTask*** com o valor **id=”Cidade”**, e adicionou ao código a notação **formKey=”cidadeController”**. Essa adição, é responsável por invocar o código *controller* da classe cidade. Da mesma forma, o metaprograma percorreu o código XML adicionando as demais notações para os IDs das classes Endereço, Banco, Agência e Gerente, possibilitando completar todas as chamadas de código dessas classes, necessários para a implementação do processo de negócio que cadastra um gerente de uma determinada agência.

Para finalizar, é necessário utilizar uma *engine* BPMN que seja capaz de executar a aplicação de acordo com o processo de negócio modelado e implementado. Para apoiar essa tarefa, a abordagem adotou a ferramenta Camunda BPMN[5]. Trata-se de uma plataforma *open source* para gerenciamento de processos de negócio, cuja execução é realizada diretamente em um *browser*.

[5]Camunda BPMN - https://camunda.com/

Após concluir toda a implementação, é preciso implantar a aplicação em um servidor Apache Tomcat[6], onde a mesma será executada.

No estudo de caso realizado, o Camunda executa a aplicação seguindo o processo de negócio definido no exemplo ilustrado na Figura 3.5. Conforme descrito no processo de negócio, para cadastrar um gerente, o Administrador segue uma sequencia cadastrando, a cidade, o endereço, o banco em que prestará serviço, a agência da qual ficará responsável e finalmente o gerente.

O Camunda é responsável por realizar o gerenciamento do processo, invocando o controlador especificado em cada atividade (Figura 3.9). O controlador por sua vez, invoca a sua respectiva *view*, apresentando ao usuário o formulário de cadastro a ser preenchido. Assim que concluídas todas as atividades, o Camunda encerra o processo e aguarda uma nova iniciação.

3.3 Processo de Manutenção

A abordagem proposta também se aplica na etapa de manutenção do software. Sua utilização no PDS pode-se obter facilidades significativas nessa etapa, pois ela possibilita que o processo não seja realizado diretamente no código e sim no modelo de negócio. Por ser alto nível, o modelo de negócio pode ser abstraído por todos os envolvidos e não somente pelos desenvolvedores, assim como ocorre em relação ao código do software. Sendo assim, caso a mudança seja apenas no modelo de negócio, ao ser alterado a *engine* BPMN se encarregará de executar a aplicação conforme a alteração realizada.

Para ilustrar o uso da abordagem nessa etapa considere a seguinte alteração solicitada pelo usuário na aplicação de gerenciamento bancário. Inicialmente ao cadastrar uma agência bancária não foi solicitado pelo usuário a definição dos respectivos caixas eletrônicos. Essa mudança, para cadastrar os caixas eletrônicos juntamente com o cadastro de uma agência implica em alterar a Regra de Negócio.

Neste caso, a alteração será realizada no modelo de negócio, onde será incluído a atividade cadastrar caixa eletrônico no processo, uma vez que já existente sua tabela no banco de dados e seus respectivos códigos do modelo de objetos. A alteração solicitada foi reespecificado o modelo de negócio conforme mostra a Figura 3.10.

Em seguida, na atividade de implementação, realizou-se a manipulação em seu código XML, acrescentando a notação necessária para especificar seu respectivo código *controller* da

[6]Apache Tomcat - http://tomcat.apache.org/

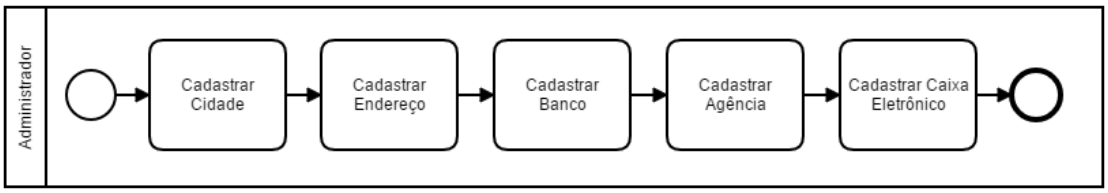

Figura 3.10: Processo de negócio cadastrar gerente bancário.

classe caixa eletrônico.

Finalmente, obteve-se o novo código da aplicação cuja execução atende a mudança de requisito solicitada pelo usuário. Ao realizar a manutenção a partir do modelo de negócio, tem-se a vantagem de facilitar o entendimento sobre a alteração solicitada, tratando-o junto ao usuário em um alto nível de abstração, num modelo de mais fácil compreensão para um usuário que conhece o negócio, mas que não é de computação.

3.4 Considerações Finais

Este capítulo apresentou a abordagem proposta, de desenvolvimento de software, que segue os princípios do MDD e utiliza a técnica de metaprogramação para realizar a geração de código na construção de aplicações. A abordagem baseia-se nos modelos de classes de objetos persistentes oriundos do banco de dados da aplicação, e nos modelos de negócios especificados em BPMN.

A abordagem é realizada em três atividades, e reune um conjunto de técnicas e ferramentas que apoiam sua execução pelo desenvolvedor. São descritos detalhes do passo a passo para execução dessas atividades, bem como os seus artefatos de entrada e saída. Essas atividades foram definidas para orientar e facilitar o desenvolvimento de aplicações web, implementadas em linguagem OO, e com persistência em banco de dados relacional.

Capítulo 4

AVALIAÇÃO

Este capítulo apresenta a avaliação realizada no intuito de validar a abordagem proposta. Desta forma, a Seção 4.1 descreve as considerações iniciais do capítulo. A Seção 4.2 apresenta o método GQM, utilizado na avaliação. As Seções 4.3 e 4.4 apresentam os estudos experimentais e as análises de seus resultados. Por fim, a Seção 4.5 apresenta as considerações finais do presente capítulo.

4.1 Considerações Iniciais

Avaliação é um processo altamente crítico para a melhoria da qualidade do processo de produção de software (CASAL et al., 1998). Através da avaliação é que se pode verificar a efetividade de novas ferramentas, métodos, processos e estratégias de desenvolvimento de software antes de colocá-los em prática. Se nenhuma avaliação é realizada, a introdução de uma nova tecnologia no desenvolvimento de software pode ser ineficiente e até mesmo ter impactos negativos na qualidade do software desenvolvido (WOHLIN et al., 2000). A avaliação na Engenharia de Software, fornece as bases para validar as teorias a respeito de uma tecnologia e confrontá-las com a realidade, com o intuito de descobrir se irão, de fato, surtir o efeito esperado tanto no produto quanto no processo (TRAVASSOS et al., 2002).

O desenvolvimento de software é, na maioria das vezes, uma tarefa complexa. Projetos de software podem se estender por longos períodos de tempo, envolver diferentes pessoas com habilidades distintas, e consistir de várias atividades que produzem diversos documentos intermediários para especificação do software a ser entregue. Essa complexidade acarreta em dificuldades para a otimização dos processos de desenvolvimento (WOHLIN et al., 2000). Em geral, os métodos de avaliação baseiam-se na premissa de que a qualidade do produto de software é

determinada pela qualidade de seu processo de desenvolvimento, considerando a máxima de que um bom produto é resultado direto de um bom processo (CASAL et al., 1998).

A literatura sugere que novas abordagens de desenvolvimento de software devem ser avaliadas no sentido de verificar sua efetividade antes de colocá-las em prática. Contudo, para que a avaliação de qualquer tecnologia seja efetiva, torna-se necessário dar enfoque aos objetivos a serem alcançados (BASILI; ROMBACH, 1994). Solingen e Berghout (1999) afirmam que pode ser interessante utilizar abordagens para guiar a definição de modelos de avaliação. Nesse sentido, uma abordagem comumente utilizada é a *Goal-Question-Metric* (GQM) (BASILI; ROMBACH, 1994).

4.2 Modelo GQM do Estudo Experimental

A abordagem GQM baseia-se na premissa de que para avaliar qualquer tecnologia, primeiramente, os próprios objetivos da avaliação devem ser conhecidos. O GQM é dividida em três níveis (BASILI; ROMBACH, 1994):

- **Conceitual:** no qual são definidos os objetivos da avaliação. O objetivo é definido para um objeto, que pode ser um produto, um processo, um serviço, entre outros. No contexto deste trabalho, o objeto é a abordagem apresentada no Capítulo 3.4;
- **Operacional:** no qual são definidas questões que caracterizam um caminho para se alcançar um determinado objetivo de avaliação; e
- **Quantitativo:** no qual são definidas as métricas que definem um conjunto de dados (objetivos ou subjetivos), que quando interpretados, permitirão ao pesquisador responder às questões estabelecidas no nível operacional.

4.2.1 Nível Conceitual: definição dos objetivos

Para este modelo de avaliação, foram criados dois objetivos, os quais são apresentados nas Tabelas 4.1 e 4.2.

Tabela 4.1: Avaliação da abordagem quanto à sua eficiência.

Objetivo 1 (O1)
Analisar a: o uso da abordagem no processo de software **Com o propósito de:** avaliar **Com respeito à:** eficiência (tempo) **Do ponto de vista de:** engenheiros de software **No contexto de:** um grupo de alunos de pós-graduação em Ciência da Computação

Tabela 4.2: Avaliação da abordagem quanto à facilidade de uso percebida por seu usuários.

Objetivo 2 (O2)
Analisar a: o uso da abordagem no processo de software **Com o propósito de:** avaliar **Com respeito à:** facilidade de uso **Do ponto de vista de:** engenheiros de software **No contexto de:** um grupo de alunos de pós-graduação em Ciência da Computação

O primeiro objetivo diz respeito à avaliação da eficiência da abordagem, e o segundo à avaliação da facilidade de uso percebidas pelos usuários.

4.2.2 Nível Operacional: definição das questões

A partir dos objetivos apresentados na Seção 4.2.1, algumas questões foram elencadas a fim de que, uma vez respondidas, possam indicar se os objetivos de avaliação foram atingidos ou não. As questões definidas para o objetivo O1 são apresentadas na Tabela 4.3.

Tabela 4.3: Questões para o objetivo O1.

Sigla	**Descrição**
Q1O1	Quão eficiente (em termos de tempo de execução) é o desenvolvimento do software com o apoio da abordagem?
Q2O1	Quão eficiente (em termos de tempo de execução) é a manutenção do software com o apoio da abordagem?

Para elencar as questões para o objetivo O2 correspondente à facilidade de uso percebida pelo usuário, foi utilizado o modelo de aceitação *Technology Acceptance Model* (TAM) (DAVIS, 1993). Esse modelo possui como objetivo explicar o comportamento das pessoas em relação à aceitação de uma tecnologia e tem sido utilizado em estudos para avaliação de produtos de software. As questões referentes ao objetivo de avaliação O2 são apresentadas na Tabela 4.4.

Tabela 4.4: Questões para o objetivo O2

Sigla	Descrição
Q1O2	Eu gostei de trabalhar com a abordagem.
Q2O2	Usar a abordagem é uma boa ideia.
Q3O2	O acesso às ferramentas utilizadas pela abordagem é simples.
Q4O2	A todo momento utilizando a abordagem estive ciente de onde estava e de como chegar aonde queria.
Q5O2	A interação com as ferramentas e técnicas utilizadas pela abordagem são claras e compreensíveis.
Q6O2	É fácil encontrar as informações que desejo na abordagem.
Q7O2	Os recursos de manipulação da abordagem estão claros e fáceis de achar.

Para que fosse possível medir o quanto uma pessoa acreditava que o uso da abordagem era fácil, os participantes respondiam para cada afirmação indicada na Tabela 4.4, uma alternativa dentre as seguintes opções: ”Discordo totalmente”, ”Discordo em grande parte”, ”Discordo parcialmente”, ”Neutro”, ”Concordo parcialmente”, ”Concordo em grande parte”e ”Concordo totalmente”, conforme sua opinião sobre tal afirmação.

4.2.3 Nível Quantitativo: definição das métricas

Após definidas as questões para cada objetivo de avaliação, deve-se especificar um conjunto de métricas capazes de indicar valores que, quando interpretados, fornecerão as informações necessárias para que o avaliador responda a essas questões. Na Tabela 4.5 estão as métricas que serão utilizadas para responder as questões dos objetivos O1 e O2 da avaliação, correspondentes à eficiência e facilidade de uso da abordagem no processo de software.

Tabela 4.5: Métricas para avaliação da abordagem proposta.

Sigla	**Descrição**
M1	Tempo gasto (em minutos) para o desenvolvimento do software com o auxílio da abordagem.
M2	Tempo gasto (em minutos) para a manutenção do software com o auxílio da abordagem.
M3	Porcentagem de usuários que escolheram a opção “Discordo totalmente”.
M4	Porcentagem de usuários que escolheram a opção “Discordo em grande parte”.
M5	Porcentagem de usuários que escolheram a opção “Discordo parcialmente”.
M6	Porcentagem de usuários que escolheram a opção “Neutro”.
M7	Porcentagem de usuários que escolheram a opção “Concordo parcialmente”.
M8	Porcentagem de usuários que escolheram a opção “Concordo em grande parte”.
M9	Porcentagem de usuários que escolheram a opção “Concordo totalmente”.

Na Figura 4.1 sumarizam-se os relacionamentos entre as métricas descritas anteriormente e as questões especificadas para o objetivo de avaliação O1. Como pode ser visto nesta figura, cada questão está relacionada diretamente com as métricas que são efetivamente utilizadas para respondê-la.

Sendo assim, os retângulos de cor verde, especificam o nível conceitual onde são definidos os objetivos da avaliação. Da mesma forma, os retângulos de cor azul, especificam o nível operacional onde definem as questões referentes a cada objetivo, e por fim, os retângulos vermelhos representam o nível quantitativo no qual definem as métricas utilizadas para responder cada questão.

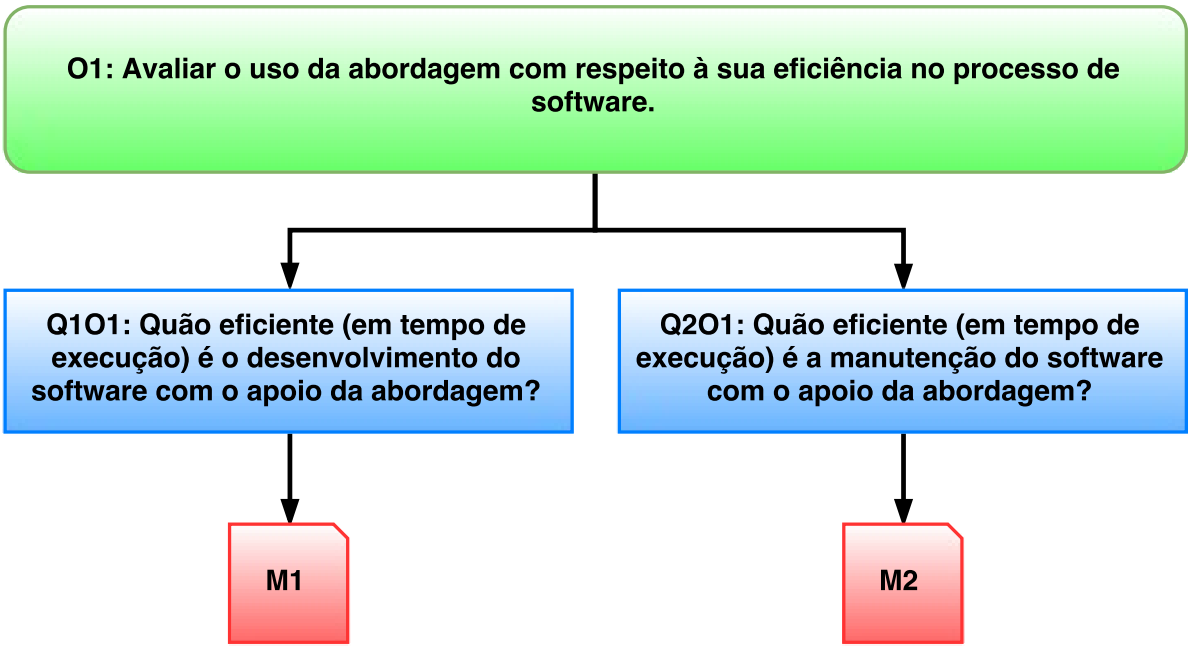

Figura 4.1: Relacionamento entre o objetivo O1 e suas respectivas questões e métricas.

Analogamente ao que foi feito para o objetivo O1, a Figura 4.2 apresenta o relacionamento existente entre o objetivo O2 e suas respectivas questões e métricas.

Uma etapa importante do nível quantitativo do modelo GQM é especificar a interpretação das métricas de cada questão, pois é por meio dela que o avaliador poderá estabelecer hipóteses, bem como extrair conclusões dos resultados obtidos após a execução de seu modelo avaliativo. Dessa forma, na Tabela 4.6 é apresentada como se deve interpretar o resultado das principais métricas elencadas para este modelo de avaliação.

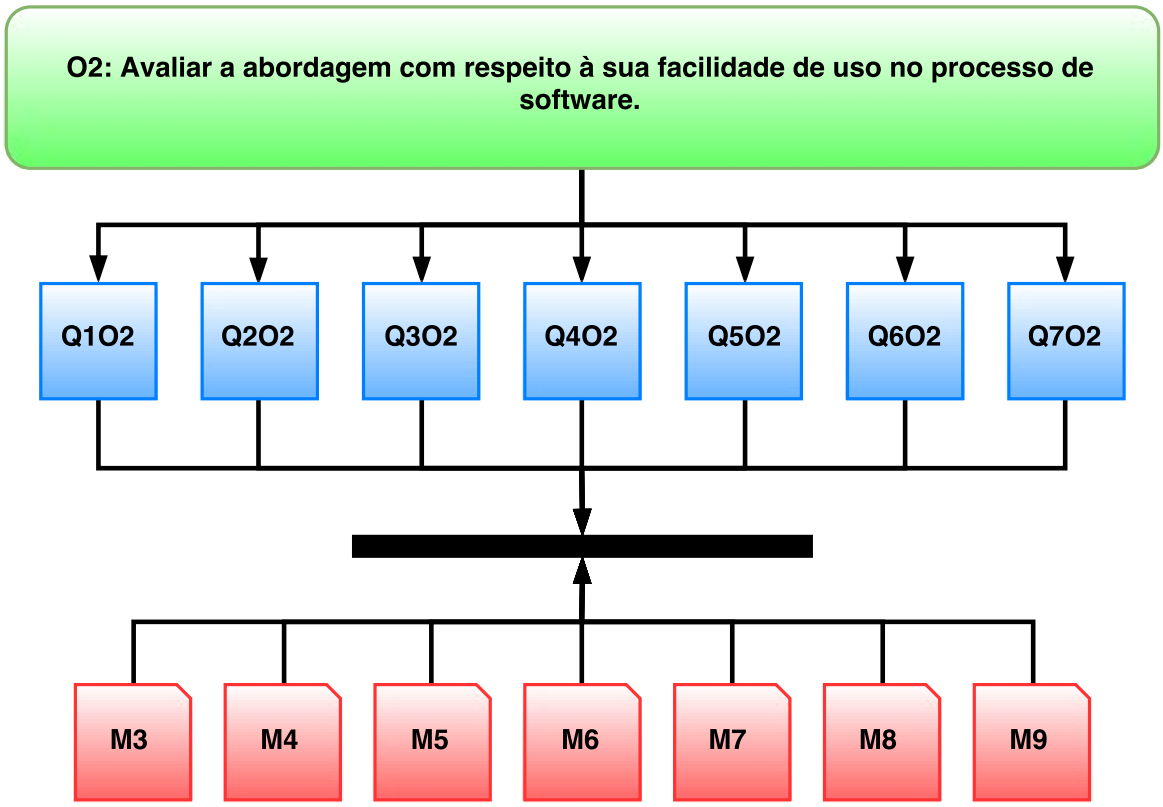

Figura 4.2: Relacionamento entre o objetivo O2 e suas respectivas questões e métricas.

Tabela 4.6: Interpretação das métricas do modelo de avaliação proposto.

Métrica	Interpretação
M1, M2	Quanto menor o valor dessas métricas, mais efetiva é a abordagem para o processo de software.
M3, M4, M5, M6, M7, M8 e M9	A facilidade de uso percebida pelos usuários da abordagem pode ser considerada satisfatória, caso M3 + M4 + M5 + M6 $<$ M7 + M8 + M9, ou seja, a porcentagem de opiniões negativas/neutra a respeito desse construto for menor que a porcentagem de opiniões positivas. Caso contrário, isto é, se M3 + M4 + M5 + M6 $\geq$ M7 + M8 + M9, então a facilidade de uso da abordagem em análise é insatisfatória.

4.3 Estudo Experimental I - Eficiência

O propósito do estudo experimental foi prover dados que, quando interpretados, permitem que os pesquisadores respondam às questões referentes aos objetivos apresentados nas Seções 4.2.2 e 4.2.1 respectivamente. Para isso um grupo de participantes foi orientado a desenvolver novas funcionalidade em uma determinada aplicação.

4.3.1 Planejamento

O planejamento deste experimento foi realizado de acordo com o modelo proposto por Wohlin et al. (2000), que envolve as fases descritas a seguir.

1. Seleção do contexto. O estudo foi realizado com dez alunos de pós-graduação em Ciência da Computação da Universidade Federal de São Carlos (PPG-CC UFSCar). O software cujo os documentos de requisitos foram utilizados referem-se a uma aplicação de controle de cotações de produtos apresentado em Weissmann (2014), que será referenciado como ConCot.

Considerando que a abordagem tem foco nas etapas de Projeto e Implementação, a fase de Identificação e Análise dos Requisitos foi realizada previamente, e seus artefatos foram utilizados como entradas para as etapas da abordagem proposta. Assim, para a aplicação ConCot foram desenvolvidos o MER, que deu origem ao banco de dados da aplicação, e os Requisitos correspondentes às Funcionalidades da aplicação foram elicitados e especificados. Essa Especificação dos requisitos e o MER do banco de dados, constam nos Anexos A e B. Esses artefatos foram as principais fontes de informações para entendimento da aplicação, para os participantes do experimento.

2. Seleção de variáveis. Variáveis independentes são aquelas manipuladas e controladas durante o estudo. Neste estudo, as variáveis independentes consistem na abordagem proposta, descrita no Capítulo 3.4, e na abordagem tradicional ”ad-hoc”. As variáveis dependentes são aquelas sob análise e cujas variações, com base nas mudanças feitas nas variáveis independentes, devem ser observadas. Neste experimento, o tempo (métricas M1 e M2) é considerado como variável dependente.

3. Projeto experimental. A distribuição dos participantes foi realizada com o intuito de formar dois grupos homogêneos (Grupo 1 e Grupo 2), com respeito ao nível de experiência dos participantes, e com cinco participantes em cada. 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. Além disso, o experimento foi planejado em fases (treinamento e execução) para minimizar ainda mais o efeito do conhecimento dos participantes.

Para a avaliação, foi criado um cenário onde os participantes partiram de uma aplicação com as funcionalidades básicas previamente implementadas, e dispondo do seu código fonte, dos seus modelos de negócio e de objetos persistentes. Para essa aplicação, os participantes desenvolveram novas funcionalidades conforme os requisitos solicitados para a sua evolução.

Como preparação da avaliação, foram disponibilizados todos os artefatos que documentam e facilitam o entendimento da aplicação na sua versão atual. Assim, o foco da avaliação foi medir a eficiência e facilidade com relação ao desenvolvimento e manutenção da aplicação, nas fases de projeto e implementação do ciclo de vida do software, não incluindo a fase de análise de requisitos. A medição compara o desenvolvimento, nas etapas de projeto e implementação, "com" e "sem" o uso da abordagem, entendendo que "sem"uso da abordagem é o desenvolvimento seguindo o processo "ad hoc", sem usar os Modelos de Negócios e de Classes de Objetos Persistentes, e os recursos de apoio, disponibilizados na abordagem proposta.

4. Execução do experimento. Antes de iniciarem o experimento, os participantes assinaram um termo de consentimento (Apêndice B) concordando com a utilização dos dados coletados neste estudo. A execução do experimento foi dividida em duas etapas onde foram avaliados aspectos do desenvolvimento e manutenção de um software utilizando ou não a abordagem em questão.

- **Etapa 1:** Nessa etapa, o Grupo 1 iniciou o desenvolvimento sem a utilização da abordagem. Em seguida foi apresentada uma alteração na regra de negócio com a qual os participantes deveriam realizar a manutenção necessária junto à aplicação. O Grupo 2 iniciou o desenvolvimento da aplicação utilizando a abordagem e, da mesma forma que o Grupo 1, realizaram a mesma alteração apresentada.

- **Etapa 2:** O Grupo 1 passou a utilizar a abordagem no desenvolvimento da mesma aplicação, e em seguida, realizou a mesma alteração apresentada na primeira etapa. Já o Grupo 2 iniciou novamente o desenvolvimento, porém sem utilizar a abordagem e também realizou a alteração necessária.

Desta forma, foi possível comparar os resultados obtidos afim de evidenciar as contribuições proporcionadas pela abordagem. Ressalta-se que, as duas etapas foram realizadas em dias diferentes para não sobrecarregar os participantes.

No desenvolvimento com a abordagem, os participantes seguiram suas atividades e tiveram apoio das suas ferramentas, enquanto que no desenvolvimento sem apoio da abordagem, os participantes seguiram o processo "ad-hoc"de desenvolvimento, partindo dos mesmos requisitos fornecidos.

4.3.2 Análise dos Resultados

Como pode ser observado na Tabela 4.7 o uso da abordagem resultou na redução do tempo de desenvolvimento do software. Esse ganho deve-se ao fato dos desenvolvedores seguirem as atividades da abordagem com o apoio das ferramentas nela integradas. Dessa forma, não despenderam tempo pesquisando quais atividades devem ser realizadas para desenvolver a aplicação, bem como quais ferramentas e tecnologias necessárias para auxiliar nas atividades.

Tabela 4.7: Tempos de desenvolvimento cronometrados de cada participante em minutos.

Grupo 1			**Grupo 2**		
Part.	**Tempo c/ Abordagem (min) - M1**	**Tempo s/ Abordagem (min) - M1**	**Part.**	**Tempo c/ Abordagem (mim) - M1**	**Tempo s/ Abordagem (mim) - M1**
P1	162	222	P6	141	207
P2	148	208	P7	158	227
P3	142	212	P8	183	247
P4	174	233	P9	171	199
P5	133	192	P10	131	238
Média	151,8	213,4	**Média**	156,8	223,6

Com o intuito de fazer uma estimativa da eficiência (Ed) proporcionada pela abordagem em relação ao tempo de desenvolvimento do software, calculou-se a diferença do tempo do desenvolvimento sem o apoio da abordagem (Tdsa), com o tempo do desenvolvimento com o apoio da abordagem (Tdca). Essa estimativa pode ser observada na equação 4.1 descrito a seguir.

$$E_d = T_{dsa} - T_{dca} = 218.5 - 154.3 = 64.2min \tag{4.1}$$

Portanto, na etapa de desenvolvimento, o uso da abordagem apresentou uma redução de tempo de 64.2 minutos com relação à não utilização da abordagem. Sendo assim, houve uma melhoria na eficiência de aproximadamente 29,38%.

Na Tabela 4.8, é apresentado o tempo individual que cada participante utilizou para realizar a manutenção do software. É possível observar que a abordagem também proporcionou uma redução no tempo, ao realizarem esta etapa, independente de qual grupo o participante pertencia. Tal redução, deve-se ao fato da facilidade com que a abordagem possibilita ao realizar as alterações necessárias, pois não são realizadas diretamente no código e sim no modelo do processo de negócio. É possível observar também, que todos os participantes levaram mais tempo para realizar a manutenção no software nos momentos em que não utilizaram a abordagem.

Tabela 4.8: Tempos de manutenção cronometrados de cada participante em minutos.

Grupo 1			Grupo 2		
Part.	**Tempo c/ Abordagem (min) - M1**	**Tempo s/ Abordagem (min) - M1**	**Part.**	**Tempo c/ Abordagem (mim) - M1**	**Tempo s/ Abordagem (mim) - M1**
P1	27	42	P6	20	37
P2	19	34	P7	23	30
P3	22	40	P8	36	62
P4	33	55	P9	30	57
P5	17	35	P10	18	38
Média	23,6	41,2	**Média**	25,4	44,8

Da mesma forma, realizou-se uma estimativa da eficiência (Em) proporcionada pela abordagem em relação ao tempo de manutenção do software. Calculou-se a diferença do tempo da manutenção sem o apoio da abordagem (Tmsa), com o tempo da manutenção com o apoio da abordagem (Tmca). Essa estimativa pode ser observada na equação 4.2 descrita a seguir.

$$E_m = T_{msa} - T_{mca} = 43 - 24.5 = 18.5min \tag{4.2}$$

Sendo assim, na etapa de manutenção, o uso da abordagem apresentou uma redução de tempo de 18.5 minutos com relação à não utilização da abordagem. Sendo assim, houve uma melhoria ainda maior na eficiência de aproximadamente 43,02%.

4.4 Estudo Experimental II - Facilidade de uso

Este estudo experimental refere-se ao objetivo de avaliação O2 (Seção 4.2.1), que visa avaliar a abordagem deste trabalho, com respeito à facilidade de uso percebida por seus usuários.

4.4.1 Planejamento

O planejamento deste experimento também foi realizado de acordo com o modelo proposto por Wohlin et al. (2000).

1. Seleção do contexto O contexto deste estudo consiste em uma situação de uso da abordagem criada, visando gerenciar todos os recursos, ferramentas e técnicas necessários para o processo de software. Este estudo foi realizado com os mesmos 10 (dez) participantes do experimento descrito na Seção 4.3 logo após os mesmos utilizarem a abordagem proposta.

2. Seleção de variáveis. As variáveis sob análise neste experimento são as porcentagens

de respostas para cada questão do questionário elaborado com base no modelo TAM citado na Seção 4.2.2.

3. Projeto e execução experimental. A distribuição dos participantes permaneceu a mesma que a descrita na Seção 4.3.1 fase 3 (três). Ao concluírem todo experimento, e adquirirem todo conhecimento após realizarem todas as etapas utilizando ou não a abordagem, solicitou-se aos participantes do estudo que respondessem a um questionário contendo as questões da Tabela 4.4 (Seção 4.2.2).

4.4.2 Análise dos Resultados

Na Tabela 4.9 é apresentado, os resultados obtidos por meio do formulário respondido pelos participantes, com relação aos facilidade de uso percebida. A primeira coluna dessa tabela contém as afirmações dadas aos participantes; as colunas 2 a 8 apresentam a porcentagem de participantes que escolheram as opções 1 a 7, respectivamente; por fim, a nona e décima coluna apresentam, respectivamente, a porcentagem de participantes que escolheram opções negativas ou neutras (1, 2, 3 ou 4) e positivas (5, 6 ou 7).

Tabela 4.9: Resultados para o construto facilidade de uso percebida.

	Opções (%participantes)								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**-**	**+**
Eu gostei de trabalhar com a abordagem.	0	0	0	0	20	30	50	0	100
Usar a abordagem é uma boa ideia.	0	0	0	0	30	50	20	0	100
O acesso às ferramentas utilizadas pela abordagem é simples.	0	0	0	0	10	30	60	0	100
A todo momento utilizando a abordagem estive ciente de onde estava e de como chegar aonde queria.	0	0	10	10	30	30	20	20	80
A interação com as ferramentas e técnicas utilizadas pela abordagem são claras e compreensíveis.	0	0	20	40	30	10	0	60	40
É fácil encontrar as informações que desejo na abordagem.	0	0	0	10	30	40	20	10	90
Os recursos de manipulação da abordagem estão claros e fáceis de achar.	0	0	0	20	40	20	20	20	80

De modo geral, pode-se notar que a abordagem obteve um porcentual maior de opções positivas do que negativas ou neutras no que diz respeito à facilidade de uso percebida pelo usuário. As três primeiras afirmações foram as mais aceitas pelos participantes, nas quais alegam terem gostado de trabalhar com a abordagem, que utilizá-la seria uma boa ideia, bem como apontam a favor da simplicidade no acesso às ferramentas utilizadas pela abordagem.

Da mesma forma, os participantes manifestaram-se em relação a interação com as ferramentas e técnicas utilizadas pela abordagem, alegando não serem claras e compreensíveis. Portanto a afirmação não obteve um grau de aceitação positiva, indicando que tais recursos deverão ser repensados e reavaliados para agregarem mais valor à abordagem proposta.

As demais afirmações obtiveram um porcentual de aceitação considerado satisfatório, pois as opções positivas escolhidas (5, 6 e 7) foram maiores que as negativas ou neutras (1, 2, 3 e 4). Desta forma, os participantes afirmaram que a abordagem demonstra facilmente os caminho a seguirem no processo de software, de maneira que o usuário não se perca ao realizar as etapas, bem como encontram facilmente as informações desejáveis na abordagem. Por fim, os mesmos afirmam que os recursos de manipulação também encontram-se de forma clara e fácil.

4.5 Considerações Finais

Este capítulo apresentou o planejamento e a execução dos estudos experimentais que objetivaram verificar a eficiência da abordagem no processo de desenvolvimento de software e a aceitação por parte dos usuários com relação à sua facilidade de uso.

Desta forma, utilizou-se para conduzir a avaliação o modelo GQM, na qual foi definido os objetivos, as questões e as métricas que forneceram dados para realizar a análise da abordagem e verificar sua viabilidade em meio ao processo de software.

Após tais definições, conduziu-se um experimento em que dez participantes desenvolveram uma aplicação a partir de alguns requisitos pré levantados e fornecidos, como por exemplo os requisitos funcionais, modelos de negócio e o banco de dados com suas entidades relacionais. Em seguida, os participantes realizaram a manutenção da aplicação de acordo com uma alteração na regra de negócio pré definida.

Por fim, os participantes responderam um questionário com o intuito de avaliar a facilidade de uso da abordagem com relação as ferramentas, técnicas e mecanismos utilizadas pela mesma no processo de software, mais especificamente nas etapas de desenvolvimento e manutenção.

Capítulo 5

TRABALHOS RELACIONADOS

Este capítulo contextualiza os trabalhos relacionados. Desta forma a Seção 5.1 apresenta as considerações iniciais do capítulo. As Seções 5.2, 5.3 e 5.4, apresentam os trabalhos encontrados na literatura. Por fim, a Seção 5.5 apresenta as considerações finais deste capítulo.

5.1 Considerações Iniciais

Diversos trabalhos de pesquisa têm sido propostos pela comunidade acadêmica e estão relacionados com os temas abordados nesta pesquisa. Este capítulo contextualiza esses trabalhos, descrevendo brevemente as características daqueles que possuem maior relação com a pesquisa desenvolvida. Tais trabalhos, inspiraram e estão relacionados de diferentes formas com o trabalho proposto.

5.2 An Approach to Support Legacy Systems Reengineering to MDD Using Metaprogramming

An Approach to Support Legacy Systems Reengineering to MDD Using Metaprogramming (PAPOTTI et al., 2012) propõe uma abordagem para o desenvolvimento de software, visando apoiar a reengenharia de sistemas legados a partir de seus bancos de dados e código fonte. Assim como mostra a Figura 5.1, tal abordagem combina os conceitos de Desenvolvimento Dirigido a Modelos (*Model-Driven Development – MDD*), engenharia reversa a partir do banco de dados, metaprogramação como técnica de geração de código e mapeamento objeto-relacional, com o objetivo de oferecer uma forma automatizada e prática de suporte na reconstrução de

sistemas legados.

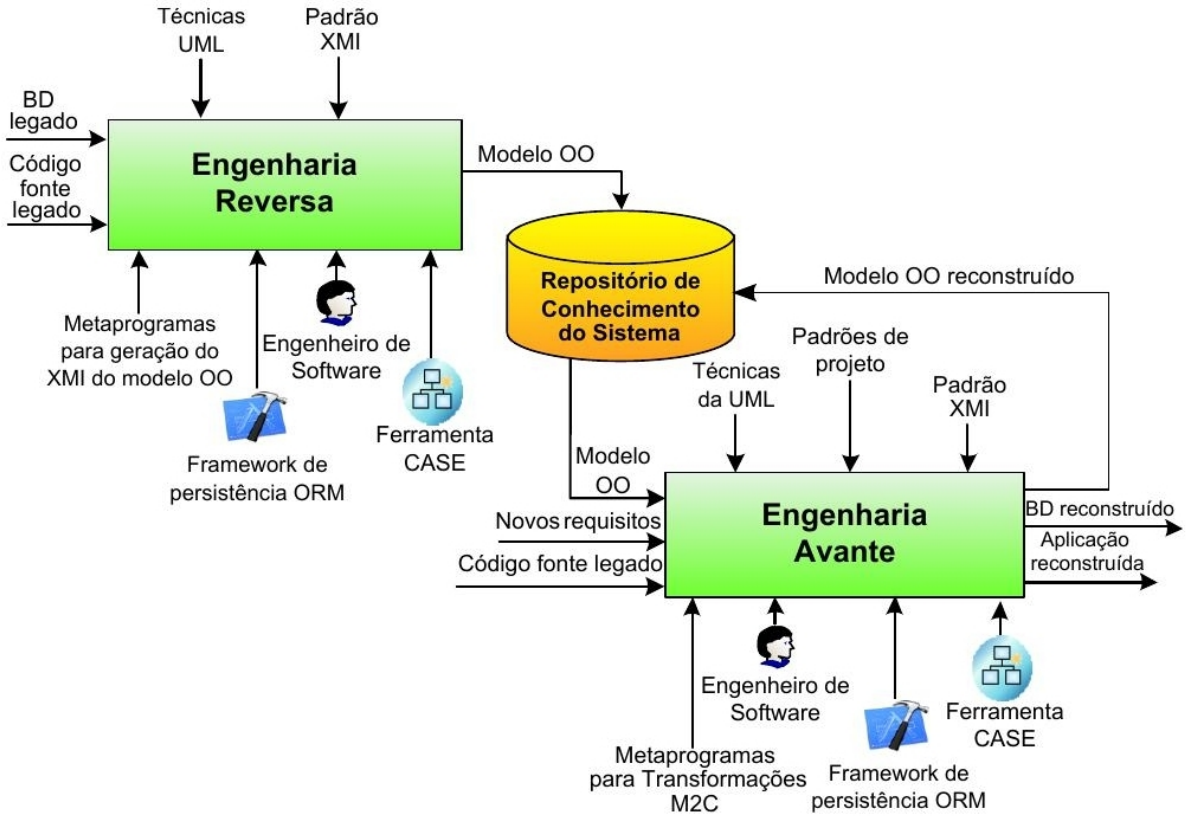

Figura 5.1: Visão geral da abordagem proposta por (PAPOTTI et al., 2012).

A abordagem é dividida em duas etapas: Engenharia Reversa (ER), cujo o objetivo é realizar a análise de uma aplicação legada, identificando seus componentes e inter-relacionamentos, criando representações da mesma em outra forma ou em um nível mais alto de abstração, e a etapa de Engenharia Avante (EA), responsável por levar abstrações de alto nível até a implementação física de um sistema.

A abordagem proposta por Papotti et al. (2012) inicia realizando uma ER com a utilização de um *framework* de persistência para gerar código das classes de entidades, por meio do banco de dados da aplicação. Obtendo o código gerado, metaprogramas analisam a estrutura das classes e geram um modelo UML Orientado a Objetos correspondente. Na etapa da EA, o modelo OO é refinado com base nos novos requisitos da aplicação, na qual metaprogramas realizam a geração de código de estruturas das classes contidas nas especificações do novo modelo, reduzindo os esforços na implementação da nova aplicação.

Da mesma forma que a abordagem proposta, a abordagem Papotti et al. (2012) visa automatizar grande parte das tarefas do desenvolvedor e tornar o processo de software mais ágil, usando a metaprogramação como técnica para realizar a geração de código automática. Portanto, ambas

as abordagens buscam reduzir a ocorrência de erros por parte dos desenvolvedores, tornando o processo ferramental. Entretanto, enquanto a abordagem Papotti et al. (2012) é específica para sistemas legados, a abordagem proposta nesse projeto de pesquisa tem foco no desenvolvimento e manutenção de software, e integra técnicas com esse foco.

Outra diferença entre as abordagens, é que a presente proposta não utiliza da UML como utensílio para a geração de código, e sim o próprio código do software gerado em atividades anteriores, tendo como partida inicial o banco de dados. Além disso, utiliza-se da notação de modelagem BPMN para realizar a integração com o código do software, para que posteriormente, a *engine* BPMN execute a aplicação final.

5.3 Software Programmed by Artificial Agents Toward an Autonomous Development Process for Code Generation

O trabalho *Software Programmed by Artificial Agents Toward an Autonomous Development Process for Code Generation* (INSAURRALDE, 2013) propõe uma abordagem que vai além das técnicas de geração de código automática para o desenvolvimento de software, apresentando um processo de desenvolvimento auto-dirigido capaz de tomar decisões por conta própria para gerar o código de software.

O processo automatizado de tomada de decisão baseia-se em um raciocinador de ontologia. O mesmo obtém informações de um banco de dados ontológico, de como executar tarefas específicas de domínio com base nas habilidades dos desenvolvedores humanos, bem como as informações relacionadas à configuração dinâmica e capacidades do sistema em desenvolvimento. O conhecimento adquirido permite o alto nível de raciocínio para interpretar, criar e sintetizar a lógica de controle do código gerado.

Portanto, o trabalho apresentado propõe uma abordagem de geração de código a partir das decisões tomadas por um agente artificial, que realiza o processo de desenvolvimento autônomo, diferente da presente proposta que utiliza a metaprogramação como mecanismo de geração de código, a partir dos modelos de negócios e de classes de objetos persistentes.

5.4 UWE4JSF: A Model-Driven Generation Approach for web Applications

UWE4JSF: A Model-Driven Generation Approach for Web Applications (KROISS et al., 2009) é uma abordagem para apoiar a geração de aplicações Web, utilizando a plataforma JSF, com escopo na *UML-based Web Engineering (UWE)*. Essa abordagem segue o princípio de "separação de interesses" por meio da construção de modelos separados para representar o conteúdo, estrutura de navegação, regras de negócio e apresentação de aplicações Web. Para isso, os modelos são construídos utilizando a UML como linguagem de modelagem e aplicando um perfil da UWE.

Conforme mostra a Figura 5.2, o processo inicia com um modelo UML (que faz uso do perfil UWE) que contém tanto o modelo independente de plataforma *(Plataform-Independent Model - PIM)* quanto os mapeamentos específicos de elementos.

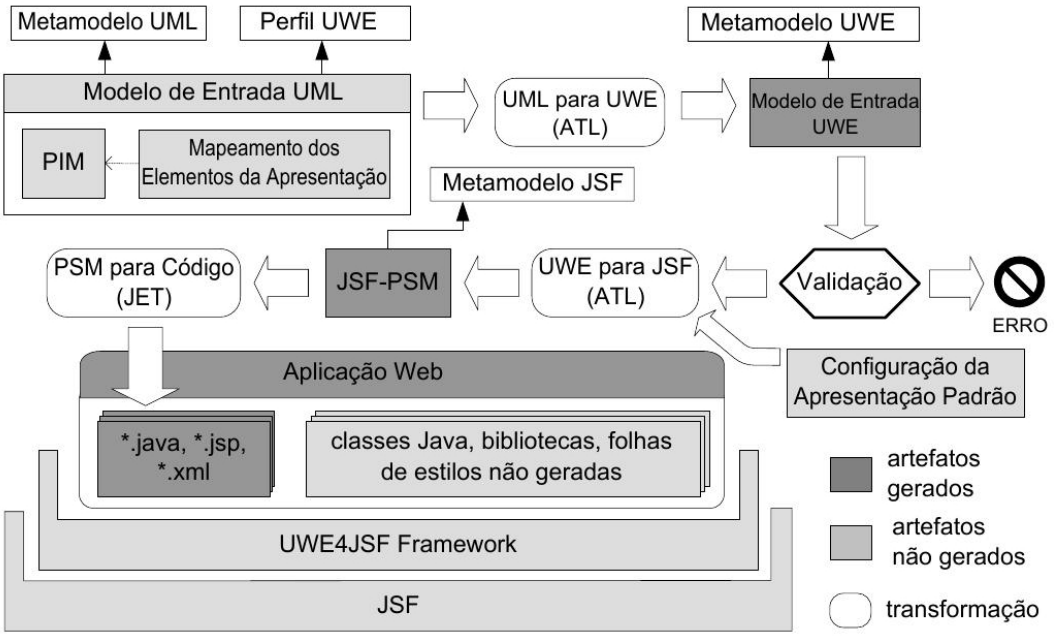

Figura 5.2: Visão geral da abordagem proposta por (KROISS et al., 2009).

Uma transformação modelo para modelo (M2M) converte esse modelo em uma instância do metamodelo da UWE, o qual é validado usando restrições *Object Constraint Language (OCL)*. Realizada a validação, outra transformação é aplicada para a geração de um modelo específico de plataforma *(Plataform-Specific Model - PSM)*. Por fim, esse PSM é usado como entrada para uma transformação M2T que gera o código de classes em Java, especificação de páginas e arquivos de configuração.

Os artefatos de código são gerados com base no *framework* UEW4JSF, funcionando como

uma plataforma intermediária que visa reduzir a complexidade do código gerado e das regras de transformação. Portanto, nota-se que o UWE4JSF propõe uma abordagem de geração de código a partir de modelos UML, destinada especificamente à geração de código para a plataforma JSF. Diferente de Kroiss et al. (2009), a abordagem proposta neste trabalho utiliza mecanismos de metaprogramação na geração de código, combina dois modelos comumente utilizados pelos desenvolvedores, e a plataforma para implementação é Grails.

5.5 Considerações Finais

Com o objetivo de situar o trabalho desenvolvido em relação a outros trabalhos presentes na literatura, este capítulo apresentou diferentes trabalhos relacionados aos tema abordados durante esta pesquisa e promoveu algumas discussões sobre as semelhanças e diferenças encontradas.

O trabalho proposto tem como base diversas características dos trabalhos correlatos, no entanto, apresenta suas próprias contribuições por meio da combinação, adequação e evolução dos conceitos e técnicas dos trabalhos com que se relaciona. De forma mais precisa, as contribuições do trabalho desenvolvido estão voltadas para o suporte ao desenvolvimento de software por meio da utilização técnicas de geração de código a partir de modelos.

Capítulo 6

CONCLUSÃO

Este capítulo apresenta as conclusões referentes a este trabalho. A Seção 6.1 descreve as considerações iniciais do capítulo. A Seção 6.2 descreve as contribuições e limitações do trabalho realizado. Por fim, a Seção 6.3 apresenta os trabalhos futuros.

6.1 Considerações Iniciais

É crescente a demanda pelo desenvolvimento de aplicações com alto nível de qualidade em um espaço de tempo cada vez menor. Nessa direção, métodos ágeis de desenvolvimento de software têm se tornado cada vez mais utilizados e aceitos pelos profissionais da Engenharia de Software. No entanto, para suprir a demanda existente e apoiar o desenvolvimento de software com qualidade e com rapidez, torna-se necessário o estudo e pesquisa de técnicas, ferramentas e processos, que viabilizem o aumento de produtividade na construção de sistemas de software.

Métodos e abordagens para auxiliar no desenvolvimento de software surgem frequentemente na tentativa de acompanhar as novas tendências referentes ao processo. Da mesma forma, as organizações empresariais estão sempre buscando alternativas para tornarem seus processos de negócio gerenciáveis, utilizando assim, mecanismos capazes de oferecer melhorias para o objetivo final.

Nesse sentido, o estudo e utilização de técnicas de modelagem e de geração de código foram os principais pontos explorados neste trabalho. Uma abordagem de processo de software foi proposto com base nas concepções de Desenvolvimento Dirigido a Modelos e de metaprogramação. A abordagem proposta define atividades e artefatos com o objetivo de auxiliar as etapas de modelagem e de geração de código ao longo do desenvolvimento de uma

aplicação.

Com o objetivo de testar a aplicabilidade e quantificar os resultados obtidos por meio da utilização da abordagem proposta, duas avaliações foram conduzidas seguindo o modelo GQM. A primeira avaliação consistiu de um estudo experimental, por meio da comparação dos tempos gastos por grupos de estudantes, na construção e alteração de uma aplicação Web do tipo CRUD, utilizando tanto a abordagem proposta quanto o processo tradicional de desenvolvimento de software. Na segunda avaliação, os participantes indicaram, por meio de um questionário, o grau de aceitação da abordagem com respeito a sua facilidade de uso.

6.2 Contribuições e Limitações

A principal contribuição deste trabalho, é o estudo realizado no intuito de identificar as principais ferramentas e técnicas existentes, que contribuem para o processo de desenvolvimento na construção de um software orientado a objetos e com persistência em banco de dados relacional, e que possibilita sua evolução e manutenção com uma maior facilidade, apoiando o desenvolvimento do software, ao longo do seu ciclo de vida.

Outra contribuição deste trabalho é a abordagem proposta, que integra técnicas de modelagem e de geração de código, com base nos conceitos e técnicas relacionados ao MDD e à metaprogramação. Em grande parte das abordagens que utilizam o MDD, o código da aplicação é gerado, por transformações, a partir dos modelos previamente construídos. Esse procedimento, muitas vezes torna essa tarefa complexa e difícil de ser operacionalizada para uso prático dos desenvolvedores de software. Com a evolução do software, as mudanças nas transformações fica, muitas vezes, onerosa e desmotivante. No caso da abordagem proposta, grande parte das manutenções, são realizadas nos modelos, e reimplementadas com apoio das técnicas e ferramentas disponíveis no mercado e integradas na abordagem.

No trabalho desenvolvido, várias etapas de geração de código foram integradas na abordagem para apoiar a construção da aplicação. O código gerado a partir de modelos é consistente com a informação especificada nos modelos construídos. Por exemplo, a partir de um diagrama de classes, os código de cada classe, seus respectivos atributos, métodos e relacionamentos é gerado, mantendo as informações do modelo atualizadas com o código. A geração de código das regras de negócio, também é consistente com os modelos de negócio, considerando que a mesma é realizada com apoio de metaprogramas específicos para cada caso (Metaprograma para geração de interfaces CRUD, metaprograma para geração da camada de persistência, e outros conforme o código a ser gerado).

Destaca-se ainda, que os geradores de código são mais modulares. Essa modularidade, facilita a manutenção dos geradores de código, em comparação com geradores únicos e complexos, que englobam a geração de todas as camadas de implementação da aplicação. Como desvantagem desse particionamento, com vários geradores, tem-se dificuldade em manter um maior número de geradores e a necessidade do Engenheiro de Software conhecer previamente os conceitos e técnicas relacionadas ao MDD e à metaprogramação.

O trabalho desenvolvido também apresentou os resultados de uma análise quantitativa a partir do experimento que avaliou a utilização da abordagem no desenvolvimento de sistemas, bem como sua facilidade de uso percebida pelos usuários no processo de desenvolvimento proposto. Os participantes da avaliação foram compostos por alunos de pós-graduação em Ciência da Computação da UFSCar, que desenvolveram uma aplicação do tipo CRUD com base em sistema de controle de cotação de produtos, seguindo tanto a abordagem proposta, quanto o processo tradicional ad-hoc.

Foram coletados dados sobre o tempo gasto no desenvolvimento das tarefas propostas em ambos os casos. A partir dos dados coletados, uma análise revelou, em média, uma redução significativa no tempo de desenvolvimento gasto pelas equipes, considerando as condições do experimento. Além disso, a análise qualitativa, sobre o uso da abordagem pelos desenvolvedores, indicou que a mesma contribuiu, dentre outros benefícios, para reduzir as dificuldades encontradas no processo de desenvolvimento e manutenção do software.

A abordagem foi planejada como sendo de propósito geral com o objetivo de não se restringir ao uso de ferramentas e tecnologias específicas no desenvolvimento de aplicações com uso de geração de código. Essa característica contribui para que o processo seja aplicável em diferentes contextos. No entanto, essa definição genérica e de alto nível de abstração requer dos usuários realizar adaptações e instanciar os elementos do processo proposto para o contexto da aplicação que está sendo desenvolvida.

Ainda, com relação aos resultados obtidos a partir do estudo experimental apresentado no Capítulo 4.5, embora evidenciem ganhos no tempo de desenvolvimento usando a abordagem proposta, é importante considerar que esses resultados estão limitados ao escopo de desenvolvedores de software em ambiente universitário e às tecnologias adotadas. Considerando questões de escalabilidade, para estender e generalizar os resultados obtidos para um contexto mais amplo, torna-se necessária a reaplicação do estudo experimental em sistemas de maior porte e complexidade, preferencialmente, com maior número de desenvolvedores atuando no mercado de trabalho.

6.3 Trabalhos Futuros

Embora diversos pontos foram explorados durante o trabalho desenvolvido, existem diferentes possibilidades de melhoria relacionadas ao estudo realizado. Dentre os trabalhos futuros que podem contribuir para evoluir a pesquisa realizada, destacam-se:

- O aprimoramento e construção de novas ferramentas de apoio à execução da abordagem proposta;
- Estudo e desenvolvimento de técnicas e mecanismos para a aplicação da abordagem em outras linguagens de programação;
- A reaplicação do estudo experimental em um ambiente industrial;
- Realização de estudos que possibilitem avaliar a abordagem proposta, com relação a outras abordagens.

REFERÊNCIAS

AALST, W. M. van der. Business process management: A comprehensive survey. *ISRN Software Engineering*, Hindawi Publishing Corporation, v. 2013, 2013.

ALTAMEEM, E. Impact of agile methodology on software development? *Computer and Information Science*, v. 8, n. 2, p. p9, 2015.

ALVES, R.; SILVA, C.; CASTRO, J. A bi-directional mapping between i* and bpmn models in the context of business process management. *Requirements Engineering@ Brazil. Available at: http://www. cin. ufpe. br/˜ erbr13/arquivos/proceedings/erbr2013_submission_33. pdf*, 2013.

ATKINSON, C.; KüHNE, T. Model-driven development: a metamodeling foundation. *IEEE software*, IEEE, v. 20, n. 5, p. 36–41, 2003.

BASILI, V. R.; ROMBACH, H. D. Goal question metric paradigm. *Encyclopedia of Software Engineering*, v. 1, p. 528–532, 1994.

BATORY, D. Multilevel models in model-driven engineering, product lines, and metaprogramming. *IBM Systems Journal*, IBM, v. 45, n. 3, p. 527–539, 2006.

BEDER, D. M. Grails: Desenvolvimento web Ágil para a plataforma java. In: . [S.l.]: UFSCar, Departamento de Computação, 2016.

BHANOT, V.; PANISCOTTI, D.; ROMAN, A.; TRASK, B. Using domain-specific modeling to develop software defined radio components and applications. In: SN. *The 5th OOPSLA Workshop on Domain-Specific Modeling, San Diego USA*. [S.l.], 2005. p. 33–42.

BITKOWSKA, A. The orientation of business process management toward the creation of knowledge in enterprises. *Human Factors and Ergonomics in Manufacturing & Service Industries*, Wiley Online Library, 2012.

BITTAR, T. J.; FORTES, R. P.; LOBATO, L. L.; WATANABE, W. M. Web communication and interaction modeling using model-driven development. In: ACM. *Proceedings of the 27th ACM international conference on Design of communication*. [S.l.], 2009. p. 193–198.

CASAL, J. A.; TUBIO, O. D.; VAZQUEZ, R. G.; FERNANDEZ, M. L.; YANEZ, S. R. Formalising the software evaluation process. In: *Computer Science, 1998. SCCC'98. XVIII International Conference of the Chilean Society of*. [S.l.]: IEEE, 1998. p. 15–24.

CHINOSI, M.; TROMBETTA, A. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, Elsevier, v. 34, n. 1, p. 124–134, 2012.

CHLIPALA, A. Ur: statically-typed metaprogramming with type-level record computation. In: ACM. *ACM Sigplan Notices*. [S.l.], 2010. v. 45, n. 6, p. 122–133.

CORDY, J. R.; SHUKLA, M. Practical metaprogramming. In: IBM PRESS. *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research-Volume 1*. [S.l.], 1992. p. 215–224.

CZARNECKI, K.; EISENECKER, U. W. *Generative Programming: Methods, Tools, and Applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 0-201-30977-7.

CZARNECKI, K.; EISENECKER, U. W. Separating the configuration aspect to support architecture evolution. In: *Workshop on Aspects and Dimensions of Concern at ECOOP*. [S.l.: s.n.], 2000.

DAVIS, F. D. User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *International journal of man-machine studies*, Elsevier, v. 38, n. 3, p. 475–487, 1993.

DEURSEN, A. V.; KLINT, P. et al. Little languages: Little maintenance? *Journal of software maintenance*, v. 10, n. 2, p. 75–92, 1998.

HATANAKA, I.; HUGHES, S. *Providing multiple views in a model-view-controller architecture*. Google Patents, 1999. US Patent 5,926,177. Disponível em: ⟨https://www.google.com/patents/US5926177⟩.

INSAURRALDE, C. C. Software programmed by artificial agents toward an autonomous development process for code generation. In: IEEE. *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.], 2013. p. 3294–3299.

KLEPPE, A. G.; WARMER, J. B.; BAST, W. *MDA explained: the model driven architecture: practice and promise*. [S.l.]: Addison-Wesley Professional, 2003.

KROISS, C.; KOCH, N.; KNAPP, A. *Uwe4jsf: A model-driven generation approach for web applications*. [S.l.]: Springer, 2009.

LUCRÉDIO, D. Uma abordagem orientada a modelos para reutilização de software. *Instituto de Ciências Matemáticas e de Computação Universidade de São Paulo*, p. 37, 2009.

MEDEIROS, H. *Introdução ao padrão MVC*. 2015.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM, v. 37, n. 4, p. 316–344, 2005.

MILLER, J.; MUKERJI, J. Mda guide version 1.0.1. OMG Framingham, 2003.

OMG, B. P. M. Notation (bpmn) version 2.0 (2011). *Available on: http://www. omg. org/spec/BPMN/2.0*, 2011.

OWEN, M.; RAJ, J. *BPMN and Business Process Management: Introduction to the New Business Process Modeling Standard*. [S.l.], 2003.

PALMER, Z.; SMITH, S. F. Backstage java: making a difference in metaprogramming. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2011. v. 46, n. 10, p. 939–958.

PAPOTTI, P.; PRADO, A. do; SOUZA, W. de. An approach to support legacy systems reengineering to mdd using metaprogramming. In: *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*. [S.l.: s.n.], 2012. p. 1–10.

PEREIRA, J. C. Orientação a objetos em java. In: . [S.l.]: K19 Treinamentos, 2015.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN 0073375977, 9780073375977.

RAGHU, T.; VINZE, A. A business process context for knowledge management. *Decision Support Systems*, Elsevier, v. 43, n. 3, p. 1062–1079, 2007.

REEUWIJK, C. V. Rapid and robust compiler construction using template-based metacompilation. In: SPRINGER. *International Conference on Compiler Construction*. [S.l.], 2003. p. 247–261.

ROSS, D. T. Structured analysis (sa): A language for communicating ideas. *IEEE Transactions on software engineering*, IEEE, n. 1, p. 16–34, 1977.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. *Unified modeling language reference manual, the*. [S.l.]: Pearson Higher Education, 2005.

SCHMIDT, D. C. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, Citeseer, v. 39, n. 2, p. 25, 2006.

SHEARD, T. Accomplishments and research challenges in meta-programming. In: SPRINGER. *International Workshop on Semantics, Applications, and Implementation of Program Generation*. [S.l.], 2001. p. 2–44.

SOLINGEN, R. V.; BERGHOUT, E. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. [S.l.]: McGraw-Hill, 1999.

SOMMERVILLE, I. *Egenharia De Software*. 9. ed. [S.l.]: Pearson Prentice Hall, 2011. ISBN 9788579361081.

ŠTUIKYS, V.; MONTVILAS, M.; DAMAŠEVIČIUS, R. Development of web component generators using one-stage metaprogramming. *Information Technology and Control*, v. 38, n. 2, 2015.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. *Introdução à engenharia de software experimental*. [S.l.]: UFRJ, 2002.

VöELTER, M.; GROHER, I. Product line implementation using aspect-oriented and model-driven software development. In: IEEE. *Software Product Line Conference, 2007. SPLC 2007. 11th International*. [S.l.], 2007. p. 233–242.

WEISSMANN, H. L. *Falando de Grails: Altíssima produtividade no desenvolvimento web*. [S.l.]: Casa do código, 2014.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering: an introduction*. [S.l.]: Kluwer Academic Publishers, 2000.

LISTA DE ABREVIATURAS E SIGLAS

BPMI – *Business Process Modeling Initiative*

BPML – *Business Process Modeling Language*

BPMN – *Business Process Model and Notation*

BPM – *Business Process Management*

BP – *Business Process*

CASE – *Computer-Aided Software Engineering*

CRUD – *Create, Retrieve, Update and Delete*

DBA – *Database Administrator*

EA – *Engenharia Avante*

EPC – *Event-driven Process Chains*

ER – *Engenharia Reversa*

GQM – *Goal-Question-Metric*

IDE – *Integrated Development Environment*

MDD – *Model-Driven Development*

MDE – *Model-Driven Engineering*

MDSD – *Model-Driven Software Development*

MER – *Modelo Entidade Relacionamento*

MVC – *Model-View-Controller*

OCL – *Object Constraint Language*

OMG – *Object Management Group*

OO – *Orientado a Objetos*

PDS – *Processo de Desenvolvimento de Software*

PIM – *Plataform-Independent Model*

PSM – *Plataform-Specific Model*

SADT – *Structured Analysis and Design Technique*

SQL – *Structured Query Language*

TAM – *Technology Acceptance Model*

UML – *Unified Modeling Language*

UWE – *UML-based Web Engineering*

WSBPEL – *Web Services Business Process Execution Language*

XML – *eXtensible Markup Language*

Anexo A

MER Controle de Cotação

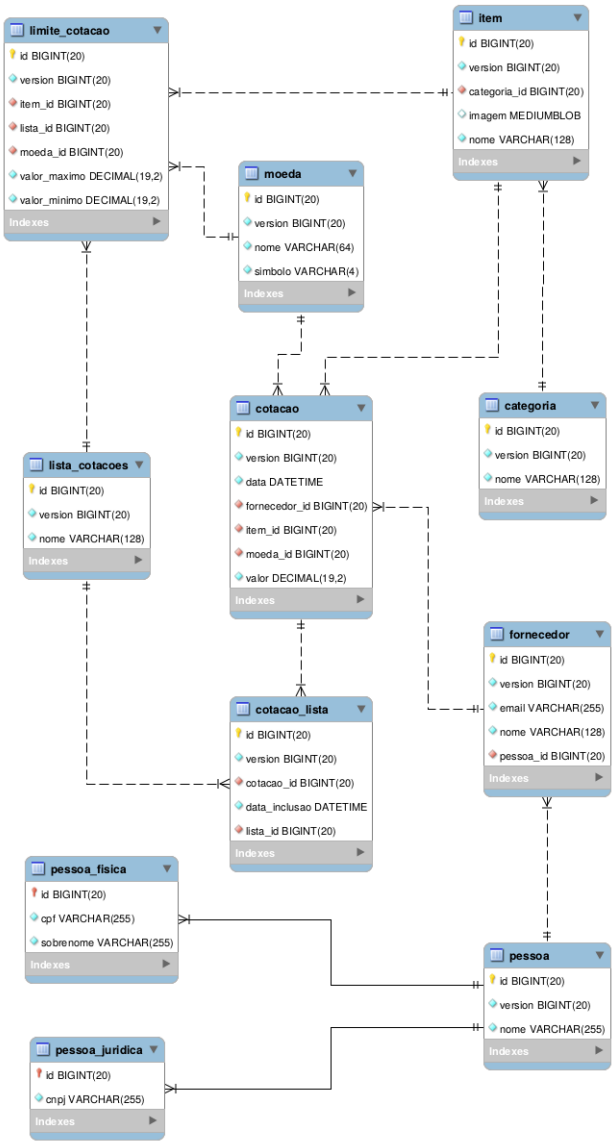

Figura A.1: Modelo Entidade Relacionamento (WEISSMANN, 2014).

Anexo B

REQUISITOS DO CONCOT

O processo de cotação consiste na execução de uma pesquisa de preços. Dado um determinado item para um determinado projeto, consultar sua lista de fornecedores em busca do preço deste produto. A cotação é o preço do item em um dado dia por um fornecedor específico, na qual o cliente escolherá o fornecedor que melhor lhe atendesse.

Administrador

01 - Como administrador devo cadastrar a categoria do produto;

02 - Como administrador devo cadastrar os itens;

03 - Como administrador devo cadastrar o tipo de valor da moeda (Real, Dólar ou outros);

04 - Como administrador devo cadastrar os fornecedores;

05 - Como administrador devo cadastrar as cotações.

Cliente

01 - Como cliente desejo uma lista de cotação de um determinado item por data;

02 - Como cliente desejo uma lista de cotação de um determinado item por fornecedor;

03 - Como cliente desejo uma lista de cotação de um determinado item por valor;

04 - Como cliente desejo uma lista de cotação de um determinado item categoria;

05 - Como cliente desejo informar qual item comprar.

Apendice A

FORMULÁRIO DE CARACTERIZAÇÃO DOS PARTICIPANTES

Este formulário tem por objetivo caracterizar sua experiência com relação a alguns aspectos da Ciência da Computação. Por favor, responda TODAS as questões o mais fielmente possível. Toda informação fornecida é confidencial e nenhum dado pessoal será divulgado em nenhuma hipótese.

1) Dados do participante.
Nome:
Idade:
Nível de formação:

2) Experiência profissional.

1. Assinale a opção que melhor reflita o seu grau de experiência com as tecnologias listadas a seguir, considerando a escala de 5 pontos:

0 = nenhum
1 = estudei em aula ou em livro
2 = pratiquei em projetos em sala de aula
3 = já utilizei em projetos pessoais
4 = utilizo em grande parte dos projetos que realizo

Linguagem de programação JAVA	0	1	2	3	4
Linguagem de programação Groovy	0	1	2	3	4
Business Process Model and Notation (BPMN)	0	1	2	3	4
Framework Grails	0	1	2	3	4
Metaprogramação	0	1	2	3	4
IDE Eclipse	0	1	2	3	4
Banco de Dados MySQL	0	1	2	3	4
Camunda Modeler	0	1	2	3	4
Camunda BPMN	0	1	2	3	4
Framework de persistência	0	1	2	3	4

2. Assinale a opção que melhor reflita sua habilidade com as seguintes atividades:

a) Modelagem de Software

( ) Nenhum ( ) Básico ( ) Médio ( ) Avançado ( ) Especialista

b) Processo de negócio

( ) Nenhum ( ) Básico ( ) Médio ( ) Avançado ( ) Especialista

c) Processo de desenvolvimento de software

( ) Nenhum ( ) Básico ( ) Médio ( ) Avançado ( ) Especialista

d) Desenvolvimento Dirigido a Modelos

( ) Nenhum ( ) Básico ( ) Médio ( ) Avançado ( ) Especialista

Apendice B

FORMULÁRIO DE CONSENTIMENTO

Experimento

Este experimento visa avaliar a aplicação da abordagem de desenvolvimento de software orientado por modelos, na construção e evolução de uma aplicação através de geração de código.

Idade

Eu declaro ser maior que 18 (dezoito) anos de idade e concordar em participar do experimento conduzido por João Paulo Moreira dos Santos na Universidade Federal de São Carlos (UFSCar).

Procedimento

Este experimento ocorrerá em duas sessões, que incluirá o desenvolvimento de uma aplicação Web a partir dos documentos de requisitos e de um banco de dados previamente estabelecidos. O desenvolvimento da aplicação se dará com e sem a utilização da abordagem proposta, conforme determinado pelo experimentador. Eu entendo que, uma vez que o experimento tenha sido concluído, os trabalhos que desenvolvi, bem como os dados coletados, serão estudados visando analisar a aplicação dos procedimentos e técnicas propostos.

Confidencialidade

Estou ciente de que toda informação coletada neste experimento é confidencial, e meu nome ou

quaisquer outros meios de identificação não serão divulgados. Da mesma forma, me comprometo a não comunicar meus resultados aos demais participantes e/ou a outros grupos enquanto não terminar o experimento, bem como manter sigilo das técnicas e documentos apresentados que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste experimento se limitam ao aprendizado do material que é distribuído e apresentado. Eu compreendo que sou livre para realizar perguntas a qualquer momento, solicitar que qualquer informação relacionada à minha pessoa não seja incluída no experimento, ou comunicar minha desistência de participação. Eu entendo que participo livremente com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisador responsável

João Paulo Moreira dos Santos
Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Professor responsável

Prof. Dr. Antonio Francisco do Prado
Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Ao preencher e assinar este formulário, dou plena ciência e consentimento com os termos acima expostos.

Nome (em letra de forma):

Assinatura:

Data: