UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM PROCESSO DIRIGIDO A MODELOS PARA GERAÇÃO DE CÓDIGO

PAULO EDUARDO PAPOTTI

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

São Carlos – SP

Maio/2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM PROCESSO DIRIGIDO A MODELOS PARA GERAÇÃO DE CÓDIGO

PAULO EDUARDO PAPOTTI

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

Maio/2013

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P218pd

Papotti, Paulo Eduardo.
Um processo dirigido a modelos para geração de código / Paulo Eduardo Papotti. -- São Carlos : UFSCar, 2013.
148 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2013.

1. Engenharia de software. 2. Software - desenvolvimento. 3. Geração de código. 4. Desenvolvimento orientado por modelos. 5. Metaprogramação. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Um Processo Dirigido a Modelos para Geração de Código"

Paulo Eduardo Papotti

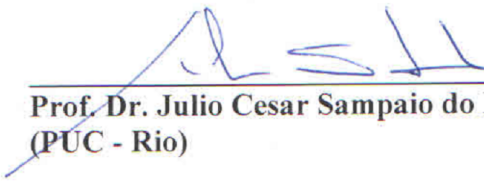
Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)

Profa. Dra. Rosângela Ap. Dellosso Penteado
(DC/UFSCar)

Prof. Dr. Wanderley Lopes de Souza
(DC/UFSCar)



Prof. Dr. Julio Cesar Sampaio do Prado Leite
(PUC - Rio)

São Carlos
Maio/2013

Primeiramente, dedico esse trabalho a Deus,
que fez de mim um ser capaz de pensar e entender o mundo em que vivo.
De modo especial, dedico esse trabalho aos meus pais Aparecida e Antonio,
que me ensinaram a importância do estudo e sempre batalharam
para dar condições para que eu pudesse vencer na vida.

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 sempre confiar no meu esforço e pela força que me deram ao longo dessa jornada. Em especial, um agradecimento aos meus pais, Aparecida e Antonio e ao meu irmão Carlos. Sem o apoio de vocês, nada disso poderia ter se concretizado.

Agradeço à Giovana, companheira e amiga fiel de todas as horas, por me acalmar nos momentos de desespero, sempre me incentivar e estar ao meu lado. Obrigado por todos esses anos.

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

Obrigado aos amigos e colegas do DC, sobretudo do LaBDES, que enfrentaram o mestrado junto comigo: Dias, Elias, Carlão, Eduardo, César, Vitinho, Léo e outros.

Aos colegas do futebol pelos momentos de descontração.

Ao Dr. Rubens, dona Rosana, Lu e Alex pelo acolhimento durante esses anos.

Aos amigos que moram em São Carlos e com quem pude compartilhar bons momentos: Mali, Herbert, Naka, Gabriel, Karina, Dhiego e tantos outros.

Aos professores do DC que contribuíram diretamente com minha formação desde a graduação e aos funcionários pelo apoio. Sobretudo, aos professores Wanderley, Luís Pires e Rosângela pelas contribuições ou sugestões dadas ao meu trabalho.

Ao CNPQ e ao PPGCC pelo apoio financeiro.

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 deixe que lhe digam que não vale a pena acreditar no sonho que se tem
Ou que seus planos nunca vão dar certo
Ou que você nunca vai ser alguém
Tem gente que machuca os outros
Tem gente que não sabe amar
Mas eu sei que um dia a gente aprende
Se você quiser alguém em quem confiar
Confie em si mesmo!
Quem acredita sempre alcança!
Quem acredita sempre alcança!

Renato Russo

RESUMO

O desenvolvimento de software é uma atividade em constante mudança ao longo do tempo. Diante da complexidade e do alto custo existente na construção de um software, existe uma grande demanda, sobretudo nas indústrias de software, por metodologias e ferramentas que possibilitem aumentar a produtividade de software, gastando menos tempo e esforços em seu desenvolvimento. Dentre os desafios existentes na Engenharia de Software, existe a necessidade de conhecer e explorar diferentes técnicas e ferramentas que viabilizem aumentar a produtividade mantendo a qualidade do software. Abordagens relacionadas ao desenvolvimento dirigido a modelos podem ser utilizadas como ponto de partida para modelar e especificar os requisitos de um sistema de software. Os modelos fornecem uma representação com maior nível de abstração do projeto do sistema e auxiliam na compreensão do sistema. Por meio da aplicação de mecanismos de transformação de modelos em código, é possível gerar código de artefatos de implementação para diferentes tecnologias a partir de dados extraídos e interpretados dos modelos. Além disso, a metaprogramação reflexiva é uma técnica que pode ser empregada para complementar a geração de código realizada por transformações de modelos, automatizando grande parte das tarefas do desenvolvedor, tais como as funcionalidades CRUD (Create, Retrieve, Update, Delete), liberando-o para atuar em outras tarefas mais importantes do processo de desenvolvimento. Dessa forma, este trabalho define um processo de software que visa orientar os profissionais no desenvolvimento de aplicações utilizando tais técnicas para realizar geração código, que tem como objetivo aumentar a produtividade de software e facilitar sua manutenção.

Palavras-chave: Processo de Software, Geração de Código, Desenvolvimento Dirigido a Modelos, Metaprogramação

ABSTRACT

Software development is an activity in constant change over time. Given the complexity and high cost that exists in software construction, there is high demand, especially in the software industries, for methodologies and tools that enable to increase software productivity by spending less time and effort in its development. Among the challenges that exist in Software Engineering, there is a need in knowing and exploring techniques and tools that enable to increase productivity and maintai software quality. Approaches related to model-driven development can be used as a starting point for modeling and to specify the requirements of a software system. Models provide a representation with higher level of abstraction of system design and assist to understand the system. Through the execution of model-to-code transformation mechanisms, implementation artifacts for different technologies can be generated based on data extracted and interpreted from models. Furthermore, reflexive metaprogramming is a technique that can be used to complement the code generation performed by model-to-code transformations, automating most of tasks of developers, such as CRUD (Create, Retrieve, Update, Delete) functionalities, freeing them to perform more important tasks in development process. Therefore, this research proposes a software process to guide professionals in developing applications using such techniques to accomplish code generation, in order to increase software productivity and facilitate maintenance.

Keywords: Software Process, Code Generation, Mode-Driven Development, Metaprogramming

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ADM *Architecture Driven Modernization* / Modernização Dirigida por Arquitetura

API *Application Programming Interface* / Interface de Programação de Aplicativos

CASE *Computer-Aided Software Engineering* / Engenharia de Software Assistida por Computador

CRUD *Create, Retrieve, Update, Delete* / Criar, Recuperar, Atualizar, Deletar

BNF *Bakus-Naur Form* / Forma de Backus-Naur

DBRE *Database Reverse Engineering* / Engenharia Reversa de Banco de Dados

DLL *Dynamic-Link Library* / Biblioteca de Vínculo Dinâmico

DOM *Document Object Model* / Modelo de Objetos de Documentos

DSL *Domain Specific Language* / Linguagem Específica de Domínio

EA Engenharia de Domínio

ED Engenharia de Aplicação

ER Engenharia Reversa

EAv Engenharia Avante

EJB *Enterprise JavaBeans*

EXE *Executable File* / Arquivo Executável

IDE *Integrated Development Environment* / Ambiente de Desenvolvimento Integrado

JET *Java Emitter Templates* / Modelos Emissores de Java

JPA *Java Persistence API* / API de Persistência Java

JSF *Java Server Faces* / Faces Servidoras de Java

JSP *Java Server Pages* / Páginas Servidoras de Java

JVM *Java Virtual Machine* / Máquina Virtual Java

LPS *Software Product Lines* / Linhas de Produto de Software

SLOC *Source Lines Of Code* / Linhas de Código Fonte

OMG *Object Management Group* / Grupo de Gerenciamento de Objetos

M2C *Model to Code* / Modelo para Código

M2M *Model to Model* / Modelo para Modelo

M2T *Model to Text* / Modelo para Texto

MDD *Model-Driven Development* / Desenvolvimento Dirigido a Modelos

MDE *Model-Driven Engineering* / Engenharia Dirigida a Modelos

MDSD *Model-Driven Software Development* / Desenvolvimento de Software Dirigido a Modelos

MD* Acrônimo que abrange todas as abordagens de desenvolvimento de software dirigidas a modelos

MOF *Meta Object Facility* / Infraestrutura de Meta-Objetos

MVC *Model-View-Controller* / Modelo-Visão-Controlador

OCL *Object Constraint Language* / Linguagem de Restrição de Objeto

OA *Orientação a Aspectos* / Orientado a Aspectos

OCL *Object Constraint Language* / Linguagem para Especificação de Restrições em Objetos

OO Orientação a Objetos / Orientado a Objetos

ORM Object Relational Mapping / Mapeamento Objeto Relacional

PHP *PHP Hypertext Preprocessor* / Pré-processador de Hipertexto PHP

PIM *Platform-Independent Model* / Modelo Independente de Plataforma

PSM *PlatformSpecific Model* / Modelo Específico de Plataforma

RIA *Rich Internet Application* / Aplicação de Internet Rica

SADT Structured Analysis and Design Technique / Técnica de Projeto e Análise Estruturada

SLOC *Source Lines of Code* / Linhas de Código Fonte

UML *Unified Modeling Language* / Linguagem de Modelagem Unificada

UWE *UML-based Web Engineering* / Engenharia Web com base na UML

WA *Web Application* / Aplicação Web

XHTML *eXtensible Hypertext Markup Language* / Linguagem de Marcação de Hipertexto Extensível

XMI *XML Metadata Interchange* / Intercâmbio de Metadados XML

XML *eXtensible Markup Language* / Linguagem de Marcação Extensível

XSLT *eXtensible Stylesheet Language for Transformation* / Linguagem de Folha de Estilo Extensível para Transformação

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Motivação

Desde muito tempo, a produtividade de software não consegue acompanhar a produtividade constante do hardware (BOEHM, 1987). A cada dia, novos processadores cada vez mais potentes, memórias e discos rígidos com maior velocidade e capacidade são lançados no mercado.

Enquanto isso, para a maioria das companhias que fabricam software, o desenvolvimento de sistemas grandes e complexos ainda é um processo complicado, caro, lento e imprevisível (BOSCH; BOSCH-SIJTSEMA, 2010).

Embora diversos avanços tenham sido realizados na área de reúso de software para melhorar esse cenário, ainda restam muitos desafios. Um dos desafios que ainda perduram está ligado à realização de especificações, arquiteturas e mecanismos para apoiar a automatização de sistemas em geral (FRAKES; KANG, 2005), mantendo a qualidade do software.

Nesse sentido, diversas partes da implementação do software, sobretudo nas fases inicias de codificação, consistem de tarefas muitas vezes estruturais e repetitivas que podem ser automatizadas (DUVALL; MATYAS; GLOVER, 2007), com o objetivo de reduzir erros e aumentar a produtividade de software.

Diante desse cenário, mecanismos reutilizáveis de geração de código tornam-se alternativas promissoras para reduzir o tempo de desenvolvimento de um software, automatizando parte das tarefas do desenvolvedor e o liberando para atuar em outras tarefas mais importantes do processo de desenvolvimento.

Abordagens com base em desenvolvimento dirigido a modelos podem auxiliar no desenvolvimento de software, tanto na modelagem dos requisitos, realizada por meio da construção de modelos com alto nível de abstração, quanto para realizar geração de código, por meio da construção e aplicação de transformações envolvendo modelos e código fonte.

Na construção de geradores de código, técnicas de metaprogramação podem ser utilizadas para construir programas com o objetivo de processar outros programas e realizar a geração de código de funções específicas, como por exemplo, a geração de código para acesso a banco de dados, construção de interfaces e aplicação de padrões de software.

O ganho de tempo, a diminuição de erros e o reúso de conhecimento propiciados por geradores de código, possibilita que os esforços do desenvolvedor sejam canalizados para outras tarefas do ciclo de vida do software que não são viáveis de serem automatizadas (e.g algoritmos e regras de negócio específicas), tornando mais eficiente o uso dos recursos e possibilitando acelerar, de forma geral, o processo de desenvolvimento de software.

Além de aumentar a produtividade no desenvolvimento de software, a geração de código pode oferecer recursos para que a manutenção do código gerado seja feita de forma mais simples. Nesse sentido, o emprego de padrões de projeto dentro do código gerado é uma medida de muita utilidade, uma vez que um código que faz uso de padrões possui um entendimento mais claro, além de facilitar o reúso e as futuras manutenções (GAMMA et al., 1995).

No entanto, a utilização de modelos e geração de código não está restrita apenas à construção de novos sistemas computacionais. A realização de melhorias em um sistema para atender novos requisitos e a utilização de novas tecnologias requerem reparos constantes na estrutura do software. Sendo assim, o fato é que, de modo geral, a estrutura de software sofre diversas mudanças ao longo do tempo, independentemente de quão cuidadosamente foi projetada (BOSCH, 2004).

Nesse sentido, alguns dos desafios da Engenharia de Software tem foco em especificações, arquiteturas e mecanismos para apoiar o processo de reengenharia de sistemas, visando reduzir tempo e esforços despendidos nesse processo. Além disso, sistemas legados, muitas vezes com alto custo de manutenção em virtude da escassez ou ausência de documentação, podem ser reestruturados e portados para uma nova e moderna arquitetura, que proporcione melhor compreensão e facilidades na

realização de futuras manutenções.

1.2 Objetivos

O objetivo principal deste trabalho é definir um processo dirigido a modelos, visando aumentar a produtividade de software por meio de geração de código, explorando a utilização de conceitos do MDD e técnicas de metaprogramação.

O processo proposto é dividido em duas fases, que são executadas em diferentes momentos. Na primeira fase, o enfoque é construir artefatos (transformações de modelos em código e metaprogramas) reutilizáveis, que serão disponibilizados para reúso futuro. A segunda fase estabelece diretrizes para a construção de novas aplicações com reúso dos artefatos construídos na primeira fase, com enfoque na geração de código referente à aplicação.

Durante a fase de reutilização dos artefatos, a geração de código é realizada em duas etapas. Na primeira etapa, é empregada uma abordagem dirigida a modelos, de modo que modelos abstratos sejam construídos para representar a modelagem da aplicação e, a partir dos modelos, o processo proposto prevê o reúso e aplicação de transformações para realizar a geração de código a partir da especificação dos modelos construídos. Na segunda etapa, o processo prevê o reúso e aplicação de metaprogramas que processam o código gerado pelas transformações de modelos e geram partes do código da aplicação.

Visando aumentar o grau de reúso e facilitar a manutenção do código gerado, o processo proposto também contempla a aplicação de padrões de projeto no código gerado.

De forma resumida, o objetivo geral deste trabalho pode ser dividido nos seguintes tópicos:

- Formular um processo de propósito geral, usando os conceitos de MDD e metaprogramação, para apoiar os profissionais da área de Engenharia de Software na construção e reengenharia de sistemas.
- Definir atividades com diretrizes para a construção de artefatos de geração de código reutilizáveis e o desenvolvimento de sistemas com reúso desses artefatos.

- Disponibilizar os artefatos construídos para reutilização em outras pesquisas.
- Avaliar a utilização do processo proposto em comparação a um processo com base no ciclo de vida clássico do software, por meio de um experimento prático realizado por indivíduos selecionados em um ambiente controlado.
- Testar a utilização do processo proposto para apoiar a reengenharia de sistemas legados.
- Servir como apoio a novas pesquisas nas áreas de desenvolvimento orientado a modelos, geração de código e metaprogramação.

1.3 Estrutura da Dissertação

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

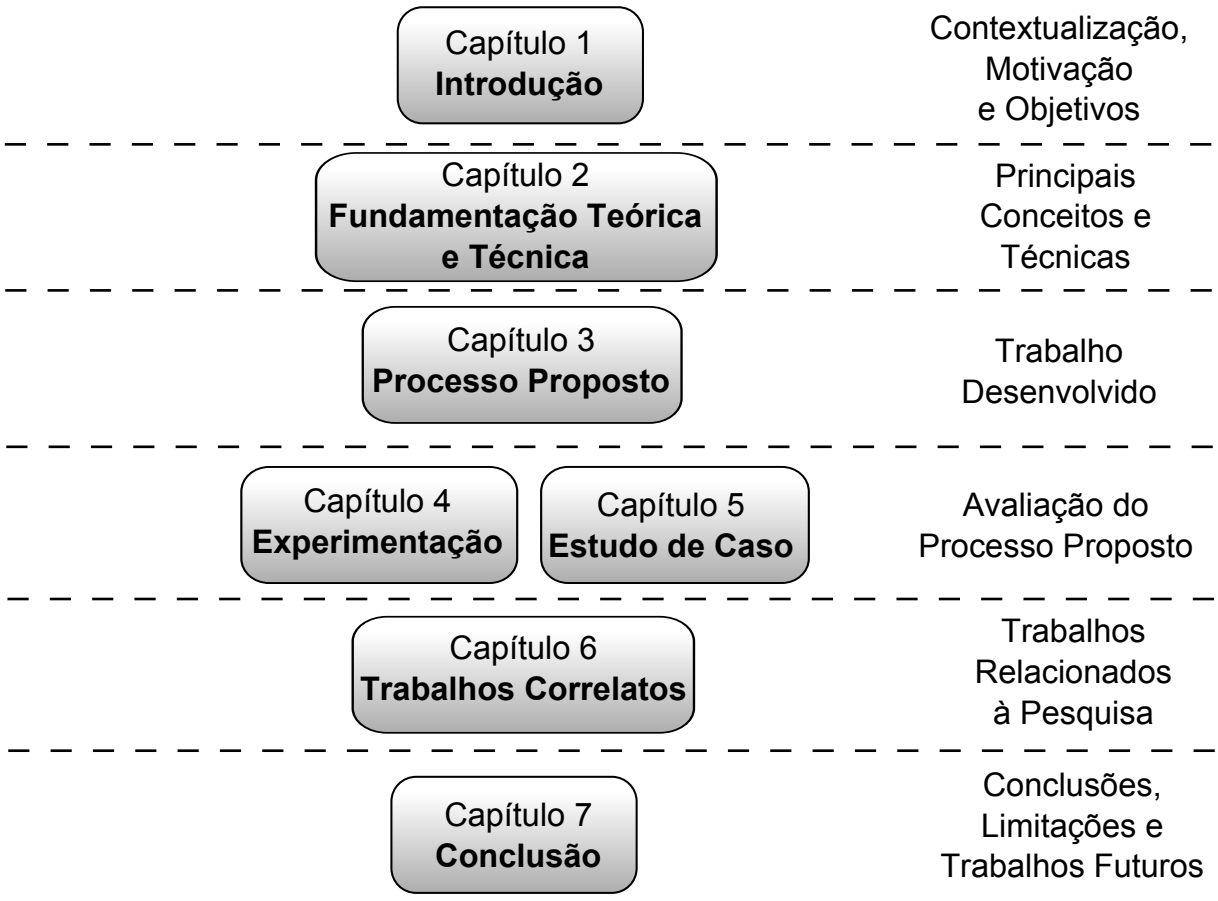


Figura 1.1: Organização da Dissertação.

- No **Capítulo 2** é apresentado o levantamento bibliográfico relacionado aos conceitos abordados nesta pesquisa: desenvolvimento dirigido a modelos, metaprogramação, padrões de projeto e reengenharia de software.

- No **Capítulo 3** é apresentado e detalhado o processo proposto neste trabalho para apoiar o desenvolvimento de software com a utilização de geração de código a partir de modelos. Cada uma das etapas e atividades do processo são definidas, apresentando suas respectivas entradas, saídas, controles e mecanismos que dão suporte aos desenvolvedores. Além disso, o capítulo apresenta uma adaptação realizada no processo para a utiliza-lo para apoiar a reengenharia de sistemas legados.

- No **Capítulo 4** é tratada 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 graduação.

- No **Capítulo 5** é descrito um estudo de caso realizado para avaliar a utilização do processo proposto na reengenharia de um sistema legado.

- No **Capítulo 6** é apresentado alguns trabalhos encontrados na literatura que se relacionam com o trabalho desenvolvido, realizando uma análise comparativa entre os trabalhos apresentados.

- No **Capítulo 7** é discutido as conclusões do trabalho desenvolvido, apresentando as contribuições, limitações e possíveis trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA E TÉCNICA

2.1 Considerações Iniciais

Neste capítulo, são apresentados os conceitos e técnicas pesquisadas para o desenvolvimento do trabalho proposto. Uma revisão da literatura foi realizada com o objetivo de fazer um levantamento do estado da arte dos conceitos de Desenvolvimento Dirigido a Modelos, Metaprogramação, Padrões de Projeto e Reengenharia de Software. Essa revisão da literatura fornece subsídios para contextualizar e os mecanismos que apoiam o trabalho desenvolvido.

Na Seção 2.2 são apresentados os conceitos relacionados ao desenvolvimento dirigido a modelos. Na Seção 2.3 é apresentado os conceitos referentes à metaprogramação. Na Seção 2.4 é discutido conceitos relativos à aplicação de padrões de projeto. Na Seção 2.5 são abordados alguns conceitos relacionados à reengenharia de software. Por fim, na Seção 2.6, este capítulo é finalizado com algumas considerações finais.

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 (BOOCH; RUMBAUGH; JACOBSON, 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.

Na Engenharia de Software, uma das linguagens de modelagem mais utilizadas pelos profissionais da área é a *Unified Modeling Language (UML)* (RUMBAUGH; JACOBSON; BOOCH, 2004). A UML é definida pelo *Object Management Group (OMG)*[1] e, atualmente, encontra-se na versão 2.4.1[2] de sua especificação, oferecendo quatorze tipos de diagramas para desenvolver a modelagem estrutural, comportamental e interacional de um software. Dentre os diagramas mais utilizados, estão o diagrama de caso de uso (comportamento), o diagrama de classes (estrutura) e o diagrama de sequência (interação), exemplificados na Figura 2.1.

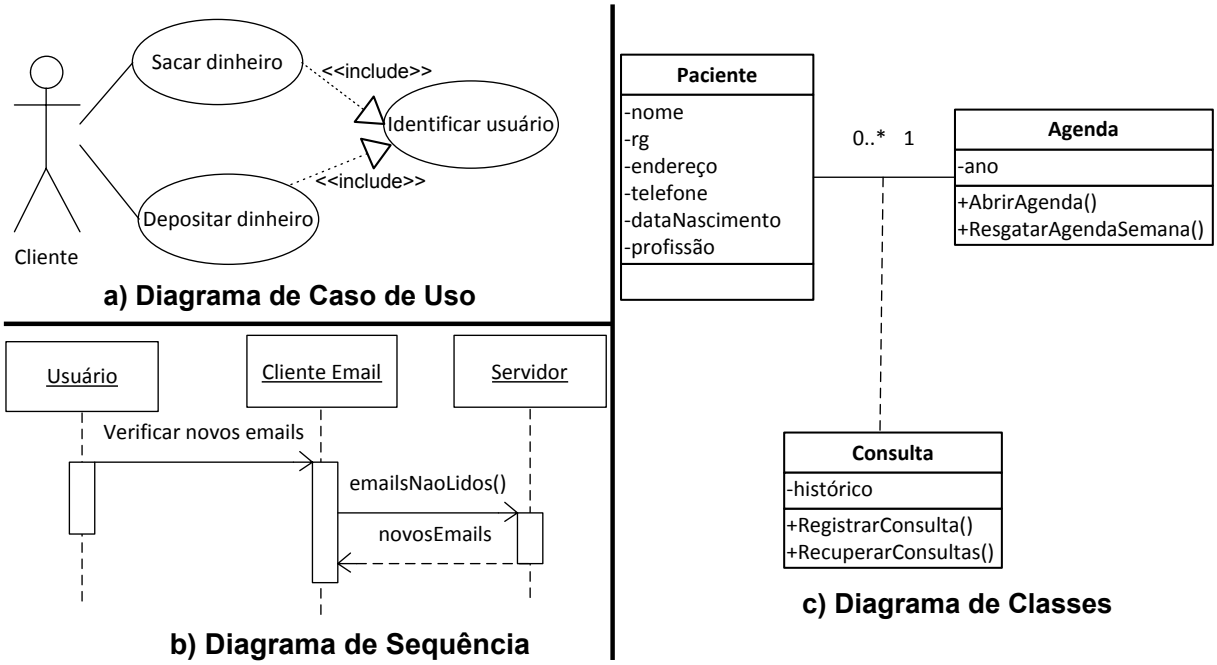


Figura 2.1: Exemplos de modelos construídos utilizando a UML.

Apesar da 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.

[1] OMG - http://www.omg.org/

[2] UML 2.4.1 - http://www.omg.org/spec/UML/2.4.1/

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), MDSD (*Model-Driven Software Development*) (VÖELTER; GROHER, 2007) ou MD* (VÖELTER, 2008), tem sido pesquisado.

O MDD, esquematizado na Figura 2.2, 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 (LUCRÉDIO, 2009). 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).

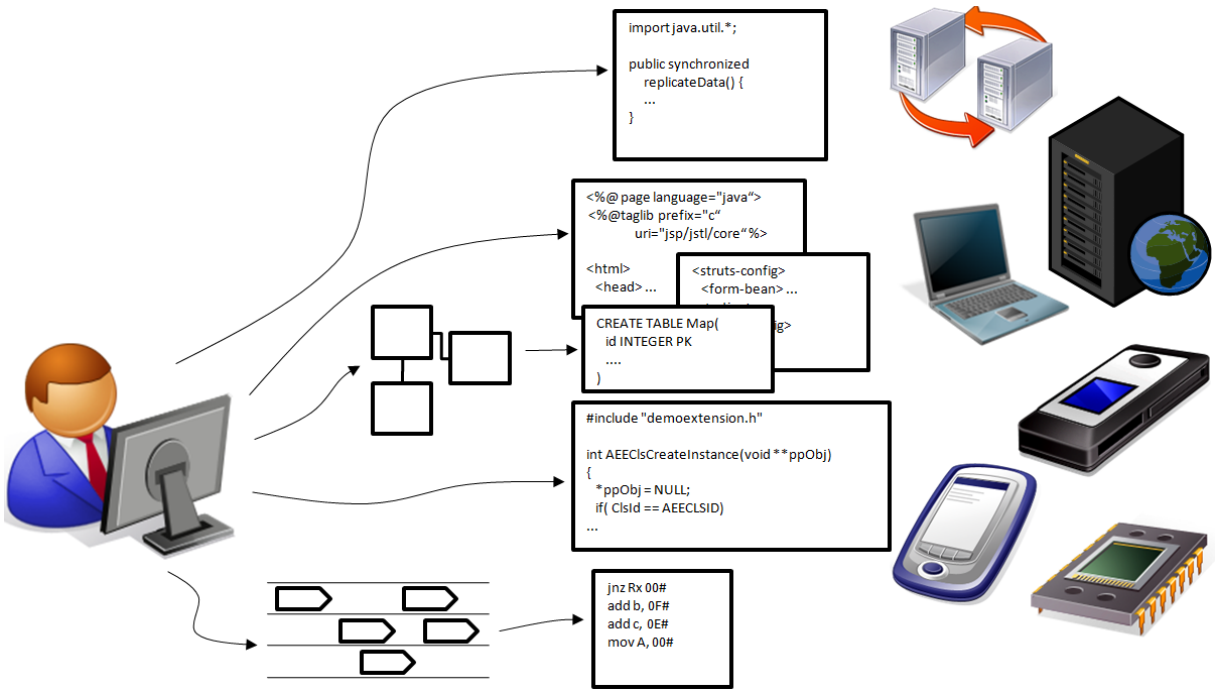


Figura 2.2: Ilustração do processo de desenvolvimento de software dirigido a modelos (LUCRÉDIO, 2009).

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 Desenvolvimento Dirigido a Modelos

Entre as vantagens existentes na utilização do MDD (KLEPPE; WARMER; BAST, 2003; DEURSEN; KLINT, 1998; BHANOT et al., 2005; MERNIK; HEERING; SLOANE, 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.

2.2.2 Transformações de Modelos

Trabalhar com diversos modelos, relacionados entre si, requer um grande esforço para mantê-los atualizados e consistentes durante todo o ciclo de vida do software. Por meio do uso de mecanismos de automatização, pode-se atingir uma redução nos esforços e tornar mais fácil a execução de atividades relativas ao desenvolvimento, tais como engenharia reversa, geração de interfaces gráficas, aplicação de padrões e refatoração.

Muitas das atividades citadas podem ser realizadas por meio de um processo automatizado, no qual, a partir de um ou mais modelos de entrada, podem ser produzidos um ou mais artefatos (outros modelos ou código-fonte) de saída, seguindo um conjunto de regras de transformação (SENDALL; KOZACZYNSKI, 2003). Esse processo, esquematizado na Figura 2.3, é conhecido como transformação de modelos.

Muitas das ideias relacionadas à transformação de modelos derivaram dos conceitos de transformação de programas (PARTSCH; STEINBRÜGGEN, 1983), tema que vem sendo pesquisado há muito tempo dentro da Ciência da Computação. Em contrapartida, existem muitas diferenças entre as duas abordagens.

A área de pesquisa que estuda transformações de programas é considerada madura e com grande tradição, estando relacionada às linguagens de programação. Esse tipo de transformação tem como base, tipicamente, conceitos matemáticos, estruturas gramaticais das linguagens e programação funcional. Por outro lado, a transformação de modelos é um campo de pesquisa relativamente novo, nascido dentro da Engenha-

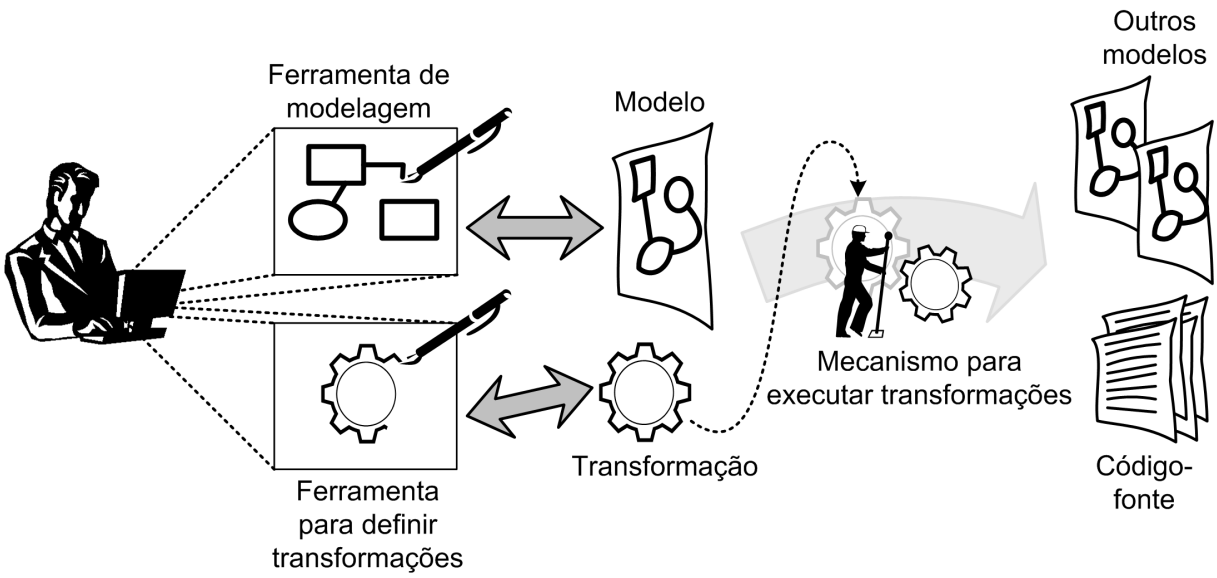


Figura 2.3: Transformação de modelos no MDD (LUCRÉDIO, 2009).

ria de Software. Nessa última, são adotados conceitos de orientação a objetos para representar e manipular os modelos (CZARNECKI; HELSEN, 2006).

Dentre os diversos tipos de abordagens para realizar transformações de modelo, uma das mais relevantes e utilizadas no MDD são as transformações Modelo para Código (*Model to Code - M2C*). Trata-se de uma transformação de modelo, com uma característica específica: código fonte e outros artefatos textuais são gerados como saída, a partir dos modelos de entrada da transformação. Esse tipo de transformação também é conhecido como transformação Modelo para Texto (*Model to Text - M2T*), dado que outros artefatos textuais, além de código fonte, também podem ser gerados (CZARNECKI; HELSEN, 2003).

As abordagens para a construção de transformações M2C podem ser divididas em: abordagens com base no padrão *Visitor* (GAMMA et al., 1995) e abordagens com base em *templates* (CZARNECKI; HELSEN, 2003).

Na abordagem que se baseia no padrão *Visitor*, a representação interna do modelo de entrada é percorrida e o código é gerado a partir de cada elemento do modelo. Um exemplo dessa abordagem é o *framework* orientado a objetos de código aberto Jamda (BOOCOCK, 2003).

Já a outra abordagem, utiliza *templates* para realizar a geração de código. Um *template* consiste de regras que são mapeadas no modelo de entrada. A estrutura de um *template* possui maior semelhança com o código gerado, comparando-se com a abordagem com base no padrão *Visitor*. Exemplos de ferramentas que utilizam essa

abordagem incluem o *Java Emmiter Templates (JET)* (VOGEL, 2009) e a ferramenta AndroMDA (BOHLEN, 2003).

2.3 Metaprogramação

Um sistema de software, tanto em sua execução quanto em sua construção, pode passar por diferentes níveis de abstração, tais como o código de máquina, código *assembly*, código em linguagem algorítmica ou orientada a objetos e diretivas de pré-processamento. Como mostra a Figura 2.4, do ponto de vista do nível de abstração, programar em maior nível de abstração é a essência que a metaprogramação visa atingir.

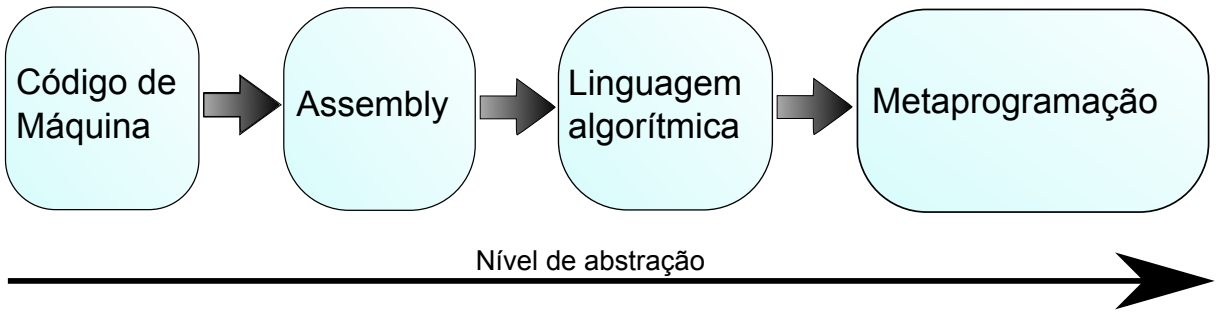


Figura 2.4: Alto nível de abstração da metaprogramação.

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; MONTVILAS; DAMAŠEVIČIUS, 2009); linhas de produtos (BATORY, 2006); transformações de programas (PALMER; SMITH, 2011); reúso generativo (TRUJILLO; AZANZA; DIAZ, 2007); manutenção, evolução e configuração de software (CZARNECKI; EISENECKER, 2000b) e aplicações *middleware* (CROSS; SCHMIDT, 2002).

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 princí-

pios são adaptados a níveis de abstração cada vez mais altos, como por exemplo, a metamodelagem (ATKINSON; KÜHNE, 2003), o *metadesign* (FISCHER; NAKAKOJI; YE, 2009) e 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. Alguns exemplos de metaprogramas são geradores de aplicação e geradores de analisadores (*parsers*).

A Figura 2.5 mostra um metaprograma construído usando a linguagem *Shell Script* (JARGAS, 2008). Esse metaprograma possui 7 linhas e sua execução gera um programa 1001 linhas na mesma linguagem com comandos para a escrita de números do 1 ao 1000. Uma vez gerado o programa, sua execução gera a saída do número 1 ao 1000 na tela do usuário.

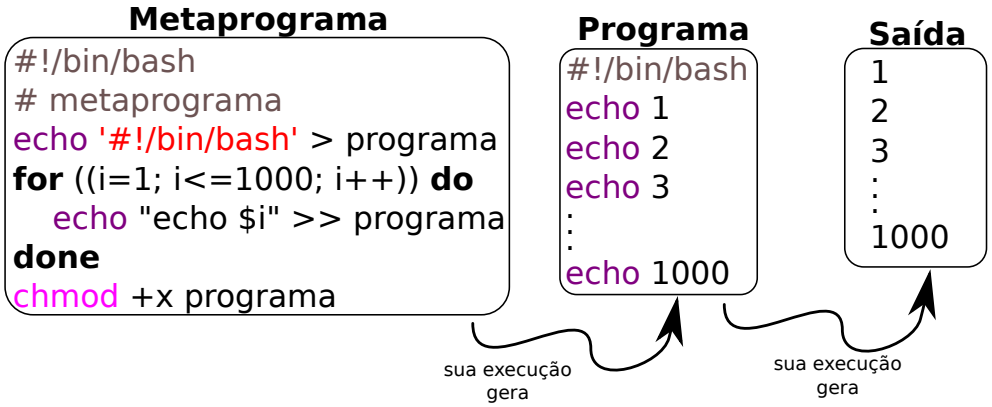


Figura 2.5: Exemplo de uma metaprograma e sua execução.

O metaprograma apresentado é apenas um exemplo simples que ilustra o uso da metaprogramação. Outros 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 meca-

nismos 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).

2.3.1 Reflexão

Existem diferentes formas de realizar a metaprogramação estática ou dinâmica. Uma das possíveis técnicas utilizadas para desenvolver metaprogramas dinâmicos é conhecida como reflexão:

> Reflexão é a habilidade integral de um programa observar ou alterar a estrutura do código bem como aspectos de sua linguagem de programação (sintaxe, semântica ou implementação), mesmo em tempo de execução. Uma linguagem de programação é dita reflexiva quando oferece reflexão a seus programas (MALENFANT; JACQUES; DEMERS, 1996, p, 3).

Durante a compilação do código fonte de um programa, a informação a respeito da estrutura do programa é normalmente perdida, restando apenas o código estritamente necessário para sua execução. Por outro lado, se um sistema suporta reflexão, a estrutura do programa pode ser preservada como metadados embutidos dentro do código compilado. Nesse caso, a metaprogramação é uma atividade reflexiva, uma vez que permite o desenvolvimento de programas que realizam a manipulação das estruturas referente ao próprio programa em execução (DAMAŠEVIČIUS; ŠTUIKYS, 2008).

Por meio da reflexão, um programa em execução pode examinar a si próprio e seu ambiente de software, por meio do acesso aos metadados, e mudar seu fluxo de execução dependendo do que encontrar. Esse autoexame, quando realizado em tempo de execução, dá-se o nome de introspecção (FORMAN; FORMAN, 2004).

Nesse sentido, a metaprogramação pode ser realizada por meio da reflexão, de modo que um metaprograma orientado a objetos pode ter acesso às definições de atributos, métodos, relacionamentos e outras informações estruturais de outros programas em tempo de execução, como exemplificado na Figura 2.6. Nesse exemplo, o código de uma classe escrita na linguagem Java é compilado pela *Java Virtual Machine (JVM)* e sua estrutura é refletida em metadados que são embutidos dentro do

código *bytecode* de saída. Dessa forma, metaprogramas podem acessar esses metadados para ter conhecimento a respeito da estrutura do código inicial.

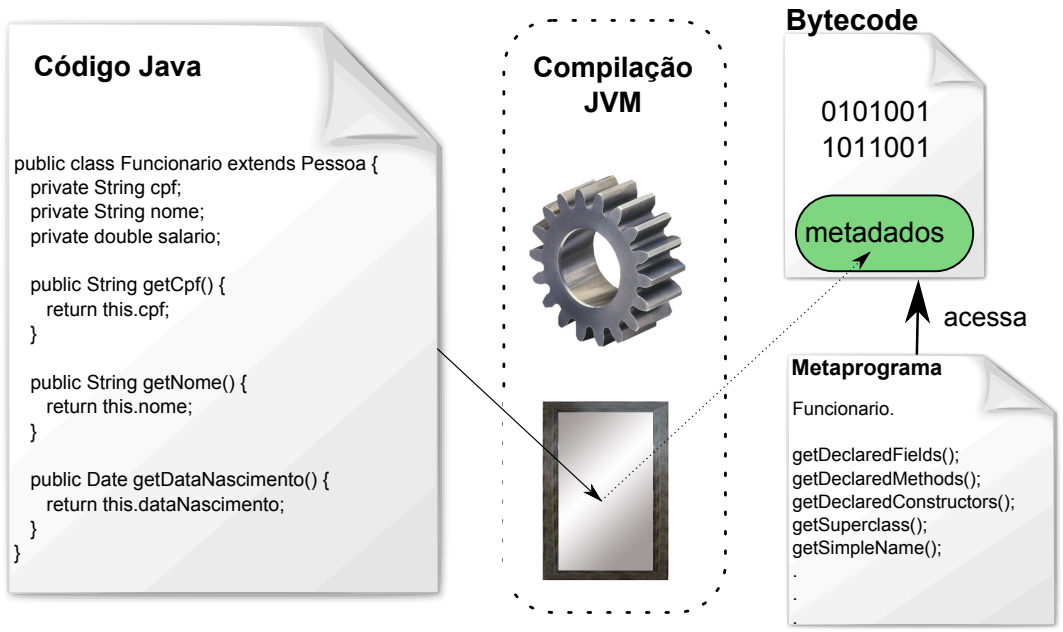


Figura 2.6: Metaprogramação por meio de reflexão.

A reflexão está presente em diversas linguagens de programação atuais, como por exemplo: Java, C#, Perl, SmallTalk, Delphi, PHP, entre outras. Dentre as inúmeras aplicações da reflexão estão, por exemplo, a extração e geração de artefatos de software para a realização de reengenharia de sistemas (CHEN et al., 2010), testes de software (FIX, 2009) e outros.

2.4 Padrões de Projeto

Projetar software orientado a objetos é uma tarefa difícil, no entanto, projetar software orientado a objetos reutilizável é ainda mais complexo (GAMMA et al., 1995). De fato, o processo de desenvolvimento de qualquer software não é simples e se alguns cuidados não forem tomados em suas fases iniciais, em pouco tempo o software pode se degradar, causando diferentes prejuízos.

O projeto e a implementação de um software precisam ser específicos para o problema que se propõe a resolver, embora também devam ser genéricos a fim de permitir mudanças e inclusões de novas funcionalidades. Todavia, desenvolver software com tais características exige muita experiência e conhecimento por parte dos profissionais de software para saber determinar quais soluções são mais adequadas para cada tipo

de problema.

Nesse sentido, os padrões de projeto captam esse conhecimento de soluções bem sucedidas e tornam mais fácil a reutilização delas por profissionais com menos experiência. Pela definição de Gamma et al.: “Padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto em um contexto particular" (GAMMA et al., 1995, p, 20).

De forma geral, um padrão é composto de quatro elementos essenciais:

- **Nome:** palavra ou sentença que faz referência a um problema de projeto, suas soluções e consequências.
- **Problema:** descreve a situação na qual o padrão deve ser aplicado.
- **Solução:** especifica todos os elementos que fazem parte do padrão, envolvendo seus relacionamentos, responsabilidades e colaborações.
- **Consequências:** descreve os resultados e análise da aplicação do padrão.

Um exemplo de padrão arquitetural muito popular é o Modelo/Visão/Controlador (*Model-View-Controller - MVC*) (KRASNER; POPE, 1988). Esse padrão, esquematizado na Figura 2.7, propõe a divisão de um sistema em três camadas de objetos. Na camada Modelo, encontram-se os objetos da lógica de negócio da aplicação. Dentro da camada Visão estão os objetos referentes à interface de apresentação e o Controlador capta as requisições do usuário e define como a interface de apresentação irá reagir. Dessa forma, essa abordagem aumenta o grau de reutilização e flexibilidade de um sistema, tornando-o mais fácil de ser mantido.

O padrão MVC, assim como muitos outros presentes na literatura, tornou-se uma solução muito bem sucedida e diferentes *frameworks* apoiam o desenvolvimento de novos softwares usando esse padrão de arquitetura. Alguns desses *frameworks* são: *Java Server Faces - JSF*[3], Struts[4] e Spring[5] para a linguagem Java, ASP.NET MVC[6] para a plataforma .NET, Django[7] para a linguagem Python e Rails[8] para Ruby.

[3] JSF - http://javaserverfaces.java.net/
[4] Struts - http://struts.apache.org/
[5] Spring - http://www.springsource.org/
[6] APS.NET MVC - http://www.asp.net/mvc
[7] Django - https://www.djangoproject.com/
[8] Rails - http://rubyonrails.org/

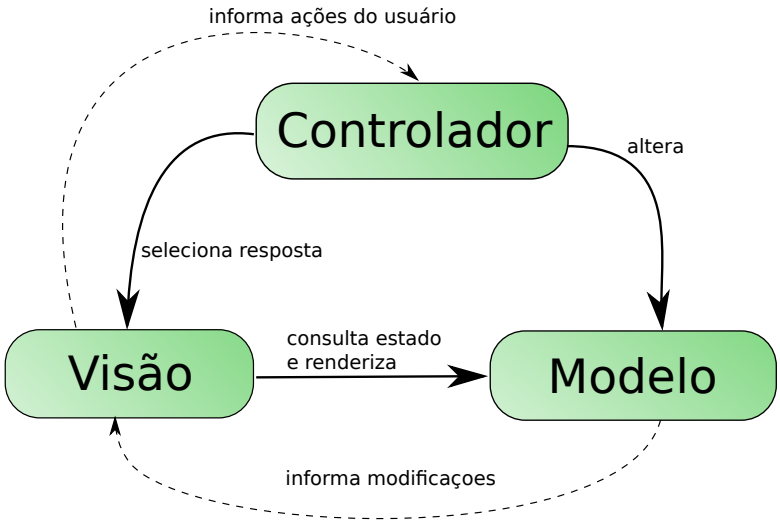


Figura 2.7: Padrão MVC.

Padrões e linguagens de padrões, entretanto, já começaram a influenciar e dar suporte outras abordagens de desenvolvimento de software (BUSCHMANN; HENNEY; SCHMIDT, 2007), como a programação orientada a aspectos (GOMES; MONTEIRO, 2010) e o desenvolvimento dirigido a modelos (FREDERICK; BOND; TILLEY, 2008).

2.4.1 Padrões de Projeto na Geração de Código

Não são apenas os pesquisadores que possuem interesse na reutilização de padrões, mas também as companhias de desenvolvimento de software. O que diferencia esse interesse é o enfoque que cada um dá para os padrões. Enquanto os pesquisadores estão mais preocupados em definir uma teoria formal para os padrões de projeto, as companhias de software estão mais preocupadas em maneiras práticas de reutilizá-los em problemas reais do cotidiano (BUDINSKY et al., 1996).

Nesse cenário, padrões de projeto é uma das áreas que está preocupada em desenvolver técnicas, mecanismos e ferramentas para apoiar a geração de código referente a diferentes tipos de padrão de projeto, visando tornar mais fácil a compreensão e reutilização desse código. Dessa forma, os usuários, sobretudo os menos experientes, não precisam começar “do zero” para realizar o desenvolvimento de novos softwares utilizando padrões de projeto.

A geração de código com base em padrões também contribui para evitar interpretações incorretas dos padrões, o que poderia levar ao desenvolvimento de implementações erradas. Nesse sentido, a utilização de padrões de projeto na geração

de código é uma vantagem, uma vez que reduz o tempo de implementação, evita a ocorrência de erros de programação, promove a reutilização de código, além de outros benefícios (MACDONALD et al., 2002).

2.5 Reengenharia de Software

Sistemas de software são construídos conforme os requisitos solicitados durante a fase de concepção. As tecnologias adotadas para suas construções normalmente são as disponíveis nessa fase do desenvolvimento. No entanto, ao longo dos anos, um sistema de software pode passar por sucessivas manutenções, seja para adicionar, excluir ou modificar seus requisitos ou para atender mudanças de plataforma de hardware e software.

As manutenções, muitas vezes, tornam-se difíceis e onerosas para acompanhar essa evolução contínua do software devido às mudanças na estrutura do software e à falta de documentação do sistema. Isso ocorre devido à mudança nos requisitos nas tecnologias que apoiaram suas plataformas, que se tornaram obsoletas.

Para acompanhar as mudanças requisitadas e não perder o conhecimento embutido nesses sistemas de software, uma solução é reconstruí-lo para atender esses novos requisitos, modernizando sua arquitetura com base em novas tecnologias. Sendo assim, para que esses sistemas continuem em operação, considerando que suas funcionalidades ainda são utilizadas por seus usuários, um processo de reengenharia pode ser necessário para modernizá-los.

Segundo Chikofsky, Cross et al. (1990) a reengenharia é um processo que visa reconstruir um sistema de software para atualizá-lo visando atender novos requisitos ou para operacionalizá-lo em novas plataformas tecnológicas.

O objetivo da reengenharia de software é adquirir conhecimento existente em sistemas antigos, denominados sistemas legados, e utilizar esse conhecimento para realizar modificações em seu código fonte, seja para adicionar novas funcionalidades ou melhorar a compreensão do sistema existente (JACOBSON; LINDSTRÖM, 1991; PENTEADO, 1996). Dessa forma, esse conhecimento serve como base para a contínua evolução do software, visando atualizá-los para novas arquiteturas, apoiadas por plataformas atuais de hardware e software. Além disso, em um processo de reengenharia, é possível incluir melhorias, modificações, exclusões e inclusões de novos

processos conforme as necessidades atuais dos seus usuários.

Na Figura 2.8 é ilustrado o relacionamento entre os elementos envolvidos durante um processo de reengenharia, considerando as etapas de Requisitos, Projeto e Implementação do ciclo de vida do software.

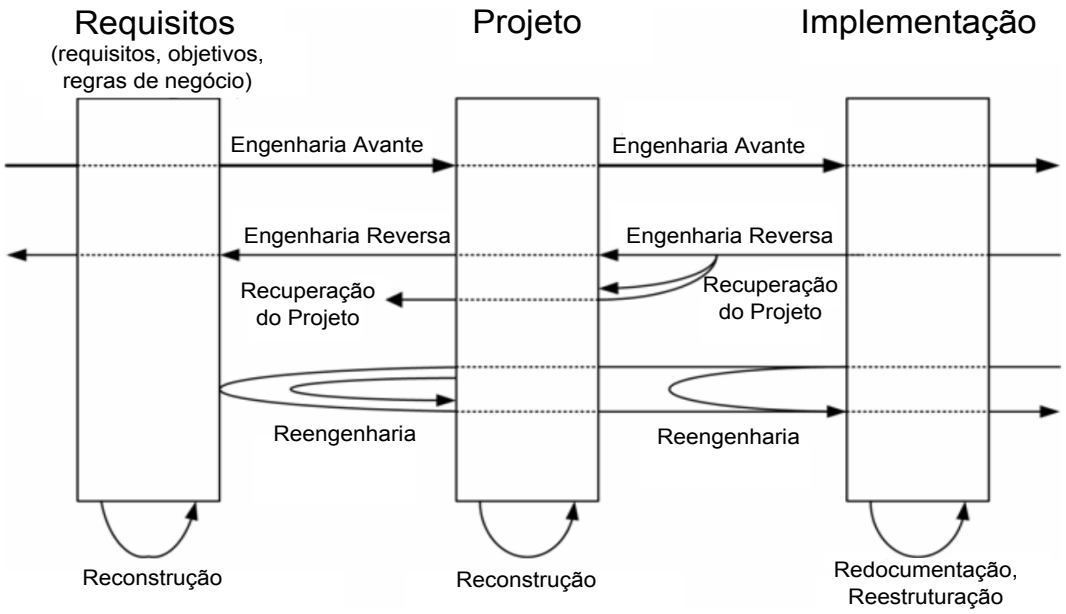


Figura 2.8: Relação entre os elementos do processo de reengenharia. Adaptado de (CHIKOFSKY; CROSS et al., 1990)

De modo geral, o processo de reengenharia de software pode ser dividido em duas etapas principais: Engenharia Reversa e Engenharia Avante. A Engenharia Reversa é o processo em que se analisa o sistema legado para identificar seus componentes e inter-relacionamentos, criando representações em um nível mais alto de abstração. A Engenharia Avante é o processo de desenvolvimento em que se parte de um alto nível de abstração, passando pelas fases do ciclo de vida do software, até a implementação física de um sistema (CHIKOFSKY; CROSS et al., 1990).

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 desenvolvimento dirigido a modelos, metaprogramação, padrões de projeto e reengenharia de software. Esta revisão auxilia o entendimento do trabalho, apresentado no capítulo a seguir.

Capítulo 3

UM PROCESSO DIRIGIDO A MODELOS PARA GERAÇÃO DE CÓDIGO

3.1 Considerações Iniciais

Processo de desenvolvimento de software, ou processo de software, corresponde ao conjunto das atividades e diretrizes relacionadas ao desenvolvimento de sistemas computacionais. Uma forma de analisar e amadurecer tal processo é por meio da sua descrição, especificando as entradas, saídas, mecanismos e controles envolvidos em cada atividade pertencente ao processo. Essa descrição permite que o processo possa ser analisado, compreendido e executado (PRESSMAN, 2005; PFLEEGER; ATLEE, 2005).

Com o objetivo de melhorar a produtividade de software, este trabalho se propõe a desenvolver um Processo Dirigido a Modelos para Geração de Código. O processo tem como base a utilização dos conceito do MDD, visando à construção de artefatos (transformações M2C e metaprogramas) reutilizáveis destinados à geração de código. Além disso, o processo é apoiado por mecanismos e controles que incluem ferramentas, *frameworks* e linguagens para modelagem, geração de código e implementação.

Este capítulo está organizado da seguinte forma: na Seção 3.2 é introduzida uma visão geral do processo proposto, apresentando seus principais elementos e suas etapas, detalhando suas respectivas atividades nas Subseções 3.2.1 e 3.2.2. Na Seção 3.3 é apresentada uma adaptação do processo proposto para apoiar a reengenharia de sistemas legados e, por fim, na Seção 3.4 o capítulo é finalizado, apresentando algumas considerações finais a respeito do processo desenvolvido.

3.2 Processo Proposto

Na Figura 3.1 é mostrada a visão geral do processo proposto, representado por meio da notação de diagramas *Structured Analysis and Design Technique (SADT)* (ROSS, 1977). Na notação utilizada no diagrama, os retângulos simbolizam cada um das atividades pertencentes ao processo e as setas possuem diferentes significados. As setas que incidem no lado esquerdo de cada retângulo representam os dados ou artefatos de entradas das atividades. As setas que saem de um retângulo no sentido direito se referem às saídas da atividade. Por sua vez, as setas que incidem na parte superior de uma atividade correspondem aos controles (e.g técnicas e linguagens) que a orientam e as setas na parte inferior representam os mecanismos e ferramentas que são utilizados durante a atividade.

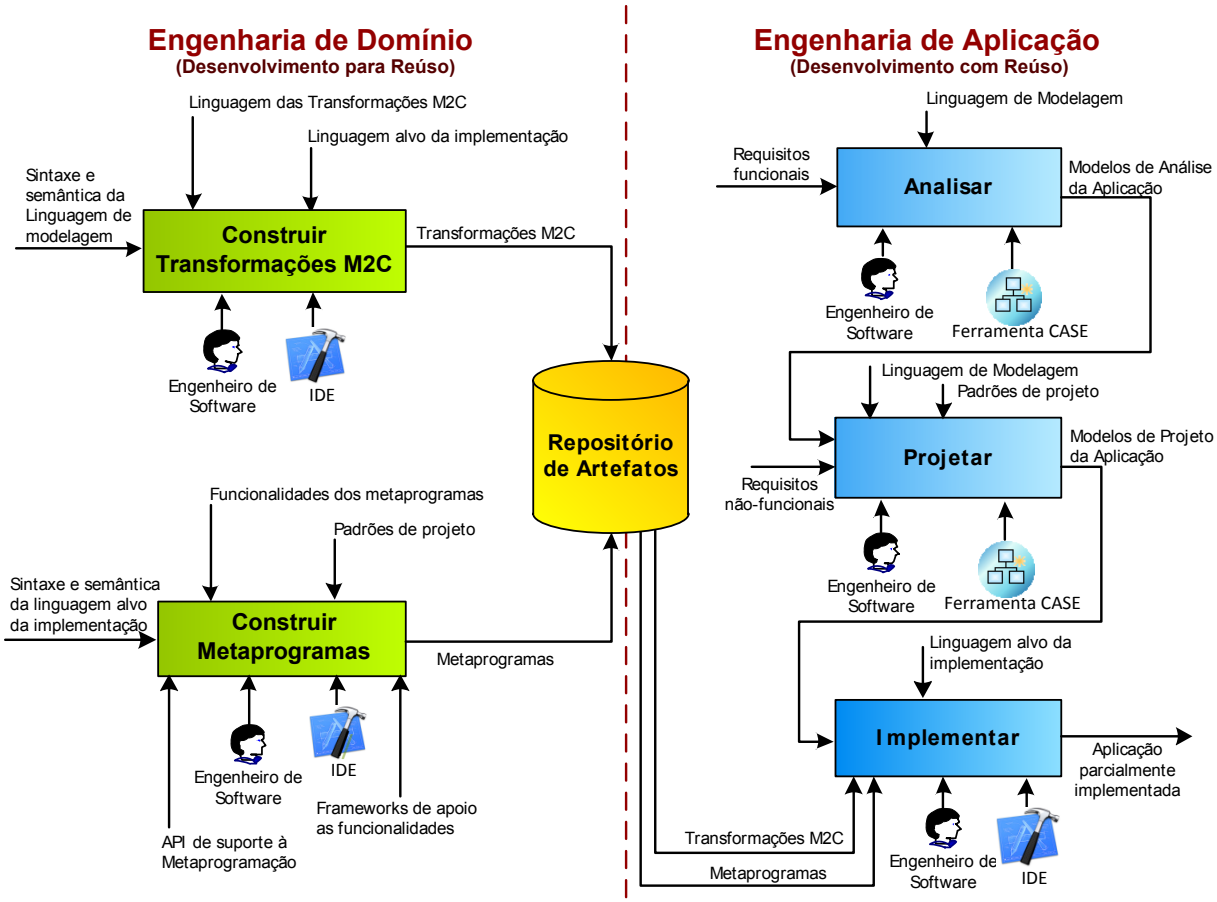


Figura 3.1: Visão geral do processo proposto.

Como mostra a Figura 3.1, o processo proposto possui cinco atividades distribuídas em duas etapas: *Engenharia de Domínio (ED)* e *Engenharia de Aplicação (EA)*. Essas etapas e nomenclatura foram inspiradas nas etapas de abordagens de *Software*

Product Lines (Linhas de Produto de Software - LPS) (GOMAA, 2004), cujo objetivo é desenvolver uma família de programas de um domínio específico.

A ED é um processo que se preocupa com a identificação e modelagem do conhecimento de um dado domínio de aplicações. O objetivo da ED é gerar modelos de domínio (e.g casos de uso do domínio, classes do domínio) para serem reutilizados em aplicações específicas (desenvolvimento *"para reúso"*) (PRIETO-DIAZ; ARANGO, 1991; BLOIS; WERNER; BECKER, 2005). Com base nisso, no processo proposto, a ED tem enfoque na construção de artefatos para reúso futuro.

A EA é um processo que identifica técnicas e métodos para realizar a construção de aplicações com base no reúso de artefatos. Os componentes de software produzidos na ED são reutilizados no desenvolvimento de aplicações de um determinado domínio do problema (desenvolvimento *"com reúso"*)(GRISS; FAVARO; D'ALESSANDRO, 1998). Durante essa etapa do processo proposto, novas aplicações são construídas com reúso dos artefatos construídos na ED.

No processo proposto, em ambas as etapas de ED e EA, a construção dos artefatos compreendem as disciplinas tradicionais do ciclo de vida do software (Análise, Projeto, Implementação e Testes), embora sejam adaptadas para a utilização dos controles e mecanismos que orientam as atividades do processo.

Um ponto importante a ser destacado é que o processo proposto possui enfoque na criação e reutilização de artefatos de software para realizar geração de código, ou de forma mais simples, geradores de código. Dentre os tipos de geradores abordados ao longo do processo, destacam-se as transformações M2C e metaprogramas.

Embora tanto transformações M2C quanto metaprogramas sejam geradores de código, existem diferenças que viabilizam realizar a distinção dos termos usados para referenciá-los. No contexto adotado para o processo proposto, transformações M2C se referem aos mecanismos de geração que utilizam descrições de modelos como entrada e código em sua saída, dando origem a programas. Por outro lado, metaprogramas se referem aos mecanismos de geração que utilizam código de programas como entrada e produzem outro código em sua saída, dando origem a novos programas.

As transformações M2C são construídas para auxiliar na automatização da escrita de código derivado diretamente a partir de modelos construídos durante o projeto de uma nova aplicação. Por exemplo, um transformação M2C pode ser construída para

que, a partir de um diagrama de classes da UML, seja gerado um código na linguagem Java das classes presentes no diagrama. Obviamente, a construção dessas transformações depende da sintaxe e semântica das linguagens usadas na modelagem e na implementação da aplicação.

Os metaprogramas desenvolvidos durante o processo possuem o objetivo de automatizar a escrita de código fonte de partes específicas na construção de uma nova aplicação, principalmente tendo com base o código gerado por transformações M2C. Por exemplo, um metaprograma pode ser desenvolvido para que, a partir de um conjunto de classes em Java geradas a partir de um diagrama de classes, seja gerado um código com interfaces gráficas para realizar as operações *Create, Retrieve, Update e Delete (CRUD)* (YODER; JOHNSON; WILSON, 1998) de uma aplicação Web. Nesse caso, a construção de metaprogramas depende da sintaxe e semântica da linguagem de implementação e de um mecanismo que possibilite o uso da metaprogramação.

A seguir, cada uma das etapas do processo proposto é detalhada, bem como suas respectivas atividades.

3.2.1 Engenharia de Domínio

A Figura 3.2 apresenta as atividades pertencentes à ED, utilizando a notação de diagramas SADT. Nessa etapa, o objetivo é produzir artefatos de software a serem reutilizados na EA, sendo realizada em duas atividades: **Construir Transformações M2C** e **Construir Metaprogramas**.

Construir Transformações M2C

O objetivo desta atividade é construir as transformações M2C reutilizáveis que serão aplicadas a modelos para realizar geração de código durante a EA.

Como é possível observar na Figura 3.2, partindo da sintaxe e semântica da linguagem de modelagem, o Engenheiro de Software escreve as transformações em uma linguagem de Transformações M2C com o objetivo de gerar código na linguagem alvo da implementação definida. Essa tarefa é apoiada por uma *Integrated Development Environment (IDE)*. Uma vez terminado esse processo de construção, as transformações M2C desenvolvidas são disponibilizadas para reúso em um Repositório de Artefatos.

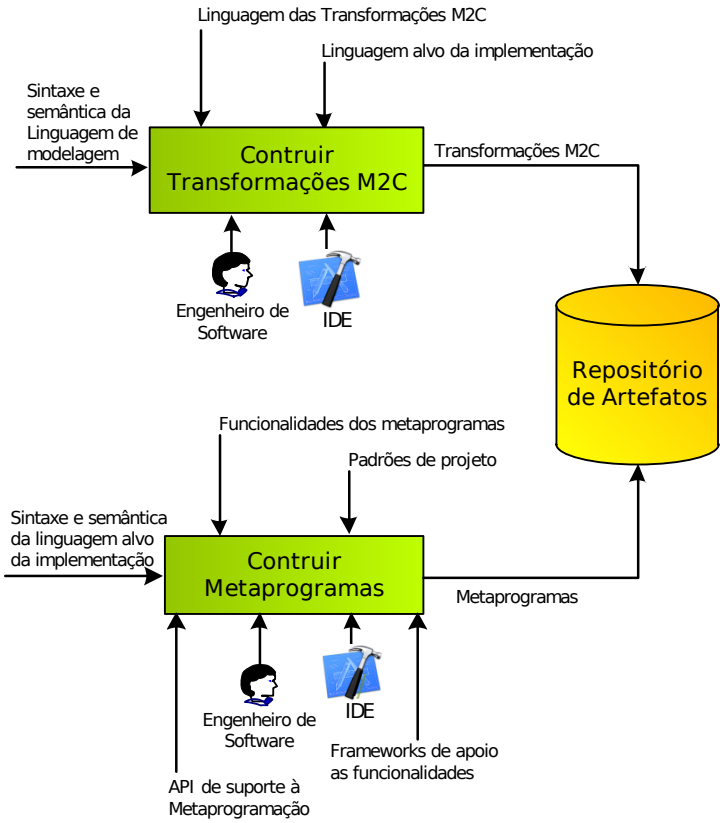


Figura 3.2: Engenharia de Domínio do processo proposto.

Conforme apontado na Seção 2.2.2, uma transformação M2C funciona como um gerador que recebe uma descrição do modelo de entrada e, a partir de um conjunto de regras de transformação, produz código fonte ou outros elementos textuais como saída (CZARNECKI; HELSEN, 2003).

No caso da utilização de linguagens de modelagens mais populares, como o caso da UML, diferentes sistemas e ferramentas CASE, tais como a MVCASE (LUCRÉDIO et al., 2003), dispõem de mecanismos para geração de código a partir de modelos. Dessa forma, um conjunto dessas transformações M2C existentes pode ser reutilizado. No entanto, outras transformações M2C podem ser construídas para transformar modelos contendo descrições específicas e que, eventualmente, não sejam contempladas pelas ferramentas disponíveis para acesso.

Exemplos de mecanismos que auxiliam a construção de transformações de modelos incluem o *Java Emitter Templates (JET)* (VOGEL, 2009), Acceleo (MUSSET et al., 2006), Xpand (EFFTINGE; KADURA, 2006) e MOFScript (OLDEVIK, 2011). Entretanto, uma transformação pode ser implementada de forma manual utilizando uma linguagem de programação com o suporte necessário. Adicionalmente, Czarnecki e

Helsen (2003, 2006), Sendall e Kozaczynski (2003), Mens e Gorp (2006) discutem conceitos sobre transformações de modelos e apresentam técnicas para suas construções.

A Figura 3.3(b) ilustra um exemplo de um trecho de transformação M2C chamada "UML para Java", que gera código de uma classe e seus atributos partindo da sintaxe e semântica de um modelo especificado usando o padrão *XML Metadata Interchange (XMI)* (OMG, 2011) da UML, que representa modelos no formato XML[1]. Esse modelo, conforme a Figura 3.3(a), é o modelo de entrada da transformação. Como se observa na Figura 3.3(c), a saída da transformação é um código na linguagem Java, adotada como linguagem alvo da implementação. Nesse exemplo, Java também foi a linguagem usada na escrita da transformação.

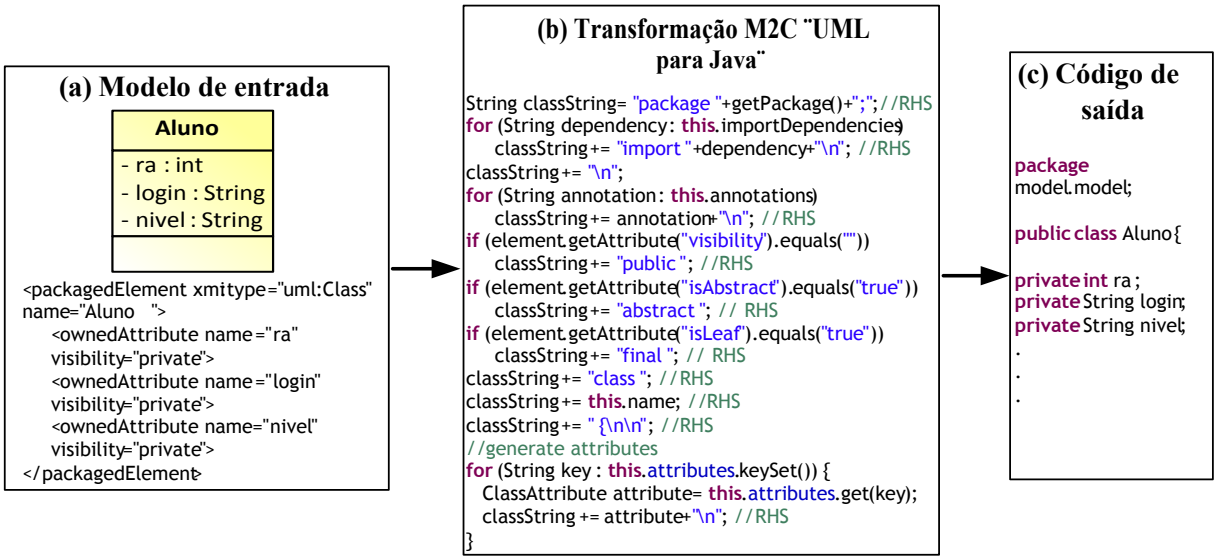


Figura 3.3: Exemplo de transformação que gera código Java a partir de um modelo de classes da UML.

Nesse exemplo, quando a transformação M2C, ilustrada na Figura 3.3(b), encontra em seu modelo XMI de entrada uma *tag* do tipo `packagedElement`, contendo o atributo `xmi:type` com o valor `uml:Class` e o atributo `name` com o valor Aluno, cria-se uma classe Java chamada Aluno. Caso existam *tags* `ownedAttribute` filhas da *tag* `packagedElement` da classe Aluno, então são gerados os códigos dos atributos dessa classe.

Nesse exemplo, a transformação fez uso da API *Document Object Model (DOM)*[2]

[1] O padrão XMI define uma estrutura formal de documento para especificar a relação entre dados e seus respectivos metadados, viabilizando a portabilidade de modelos da UML entre as ferramentas de modelagem.

[2] DOM - http://www.w3.org/DOM/

para analisar a descrição do modelo de entrada especificado na UML por meio do padrão XMI, definido pelo OMG[3].

Construir Metaprogramas

O objetivo dessa atividade é construir metaprogramas reutilizáveis que serão aplicados ao código gerado pelas transformações M2C para complementar a geração de código.

Como se observa na Figura 3.2, o Engenheiro de Software parte da sintaxe e semântica da linguagem alvo da implementação para escrever um metaprograma. Um metaprograma também implementa uma ou mais funcionalidades a serem geradas em uma linguagem alvo da implementação. Essa atividade é apoiada por uma IDE, *frameworks* que suportam a execução das funcionalidades do metaprograma e uma API de metaprogramação. Essa API de metaprogramação é necessária para processar e estender a implementação de um conjunto de classes em uma linguagem alvo da implementação, possibilitando extrair informações do código, como por exemplo, nome de classes, atributos e seus respectivos tipos, métodos, relacionamentos com outras classes e outros possíveis elementos estruturais. Depois de construídos, os metaprogramas são disponibilizados para reúso em um Repositório de Artefatos.

De modo informal, no processo proposto, o objetivo ao construir um metaprograma é que o mesmo seja capaz de processar um programa de origem, gerado por transformações M2C, e incrementar o programa com a geração de código de uma ou mais funcionalidades desejadas. As funcionalidades dos metaprogramas variam conforme as arquiteturas adotadas para implementação e tornam os metaprogramas mais robustos, não se limitando a geração de esqueletos e protótipos de classes. Por exemplo, é possível construir metaprogramas que geram códigos para dispositivos móveis e Web, serviços e persistência de dados.

É importante destacar que, no processo proposto, a construção de metaprogramas tem enfoque na utilização de padrões de projeto, de modo que o código gerado implemente padrões sempre que possível. Essa preocupação é justificada por dois motivos. Primeiramente, a utilização de padrões de projeto contribui para obter uma solução eficaz para um determinado problema e possibilita maior grau de reutilização em comparação com soluções que não utilizam padrões na implementação (GAMMA

[3]OMG - http://www.omg.org/

et al., 1995). Em segundo lugar, muitas vezes, o entendimento do código gerado nem sempre é uma tarefa fácil, sobretudo para os profissionais com menos experiência (VOELTER, 2003). Dessa forma, a utilização de padrões contribui para tornar o código gerado mais fácil de ser entendido.

Por exemplo, é possível construir um metaprograma reflexivo que, com base no código das classes geradas por transformações M2C, realiza a geração de código das funcionalidades CRUD com persistência em banco de dados. A Figura 3.4 mostra um fragmento de um metaprograma reflexivo chamado “M-CRUD”, que processa código Java e gera código de uma aplicação Web com as funcionalidades CRUD a partir de um conjunto de classes em Java.

```
1  Class classes[];
2  classes = CRUD.getClasses("model");
3  for (Class c : classes) {
4    //gera um bean gerenciado para cada classe
5    CRUD.generateManagedBean(c);
6    //gera um conversor para cada classe
7    CRUD.generateConverter(c);
8    //gera uma página Web de cadastro para cada classe
9    CRUD.generateViewAdd(c);
10   //gera uma página Web de edição para cada classe
11   CRUD.generateViewEdit(c);
12   //gera uma página Web de listagem para cada classe
13   CRUD.generateViewList(c);
14 }
15 CRUD.generateFacesConfig(classes);
16 CRUD.generateViewIndex(classes);
```

Figura 3.4: Fragmento de um metaprograma reflexivo que gera o código de uma aplicação com as funcionalidades CRUD a partir de um conjunto de classes em Java.

O exemplo de metaprograma da Figura 3.4 busca por todas as classes que estão no pacote “model” e para cada classe encontrada são geradas as classes das camadas de Controle e Visão, conforme o padrão MVC. No controle, têm-se os componentes denominados “gerenciados”, componentes de conversão de objetos e, na visão, páginas web para o CRUD. Essa geração baseia-se nas classes, com seus atributos, métodos e relacionamentos obtidos por meio do uso de técnicas de reflexão. O Quadro 3.1 mostra alguns exemplos de mapeamentos de tipos de atributos de uma classe e sua representação na camada Visão e a Listagem 3.1 exibe um *template* de um metaprograma para geração de uma página web de cadastro a partir dos atributos de uma classe em Java.

Quadro 3.1: Mapeamentos de atributos Java na camada Visão.

Tipo do Atributo	Visualização Gerada
String, int, float, double	
Date	
Collection	
boolean	

Listagem 3.1: *Template* **de um metaprograma para geração de uma página web de cadastro.**

```
public static void generateViewAdd(Class c) throws IOException
{
  Arquivo viewAdd = new Arquivo(CRUD.projectPath+"\\WebContent\\"
                        +c.getSimpleName()+"\\new"+c.getSimpleName()+".jsp");
  viewAdd.append("<%@page contentType=\"text/html\" pageEncoding=\"UTF-8\"%>\n"+
    "<%@taglib prefix=\"f\" uri=\"http://java.sun.com/jsf/core\"%>\n"+
    "<%@taglib prefix=\"h\" uri=\"http://java.sun.com/jsf/html\"%>\n\n"+
    "<f:view>\n"+
    "  <html>\n"+
    "    <head>\n"+
    "      <link rel=\"stylesheet\" type=\"text/css\"  href=\"../css/styles.css\"/>\n"+
    "     <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\"/>\n"+
    "     <title>New "+c.getSimpleName()+"</title>\n"+
    "   </head>\n"+
    "   <body>\n"+
    "     <div class=\"line-separator\"></div>\n"+
    "     <div style=\"text-align:center\">\n"+
    "     <h1 class=\"title\">New "+c.getSimpleName()+"</h1>\n"+
    "     </div>\n"+
    "     <div class=\"line-separator\"></div>\n"+
    "     <br>\n"+
    "     <h:form>\n"+
    "       <h:panelGrid columns=\"2\" styleClass=\"table\">\n");

  List<Field> fields = new ArrayList<Field>();
  //Considera os atributos da classe pai (se houver)
  for (Field f : c.getSuperclass().getDeclaredFields())
    fields.add(f);

  //Considera os atributos da classe que está sendo processada
  for (Field f : c.getDeclaredFields())
    fields.add(f);

  for (Field f : fields)
  {
    String fieldClass = f.getType().getSimpleName();
    String tipo = f.getType().getSimpleName();

    if (tipo.indexOf("PK") != -1)
    {
```

```
      //Se o atributo analisado é PK composta, realiza geração dos atributos da classe da
      //PK composta
      for (Field pkClass : f.getType().getDeclaredFields())
      {
        String beanPath = c.getSimpleName()+"ManagedBean."
                          +StringUtils.firstLetterLowerCase(c.getSimpleName())+"."
                          +StringUtils.firstLetterLowerCase(f.getName())+"."
                          +StringUtils.firstLetterLowerCase(pkClass.getName());
        viewAdd.append(evaluateFieldType(pkClass,beanPath));
      }
    }
    else
    {
      //Se o atributo não é PK composta, o atributo é gerado diretamente
      String beanPath = c.getSimpleName()+"ManagedBean."
                        +StringUtils.firstLetterLowerCase(c.getSimpleName().toLowerCase())
                        +"."+StringUtils.firstLetterLowerCase(f.getName());
      String fieldCode = evaluateFieldType(f,beanPath);
      if (fieldCode != null)
        viewAdd.append(fieldCode);
    }
  }
  viewAdd.append("        </h:panelGrid>\n"+
            "       <br>\n"+
            "       <h:messages errorClass=\"error\"></h:messages>\n"+
            "       <div style=\"text-align:center\">\n"+
            "         <h:commandButton value=\"Save\" action=\"#{"+c.getSimpleName()
                +"ManagedBean.new"+c.getSimpleName()+"}\" styleClass=\"save\"/>\n"+
            "         <h:commandButton value=\"Cancel\" immediate=\"true\" action=\"list"
                +c.getSimpleName()+"\" styleClass=\"cancel\"/>\n"+
            "       </div>\n"+
            "     </h:form>\n"+
            "   </body>\n"+
            " </html>\n"+
            "</f:view>\n");

  viewAdd.gravar();
}
```

Nesse caso, o metaprograma “M-CRUD” utilizou a API Reflection da linguagem Java como API de metaprogramação e o código gerado integra o uso de *frameworks* para apoio às funcionalidades do metaprograma e ao reúso de software. Foram utilizados os *frameworks* JSF, para implementação do padrão MVC e Hibernate[4], para realizar a persistência de objetos em banco de dados relacional. O código gerado pelo metaprograma “M-CRUD” também faz uso dos padrões de projeto *Singleton*, *Facade*, *Abstract Factory* e *Observer* (GAMMA et al., 1995), tornando o código mais fácil de ser reutilizado, auxiliando na documentação e na manutenção.

[4]Hibernate - http://hibernate.org/

Da mesma forma, é possível construir outros metaprogramas com a finalidade de gerar parte do código de uma aplicação. Nesse caso, outras tecnologias, *frameworks* e padrões podem ser utilizados na construção de metaprogramas.

3.2.2 Engenharia de Aplicação

Da mesma forma que nas abordagens de LPS, a EA tem como objetivo principal estabelecer um guia para a construção de aplicações com base na reutilização de software. Nessa etapa, os componentes de software reutilizáveis (transformações M2C e metaprogramas) construídos durante a ED são utilizados para realizar geração de código das aplicações desenvolvidas.

A Figura 3.5 exibe um diagrama SADT referente à etapa de EA no processo proposto, composta de três atividades: **Analisar**, **Projetar** e **Implementar**. Essa etapa tem como base as disciplinas do ciclo de vida clássico do software, adicionando conceitos e técnicas relacionadas ao MDD e metaprogramação para realizar a geração de código durante o desenvolvimento.

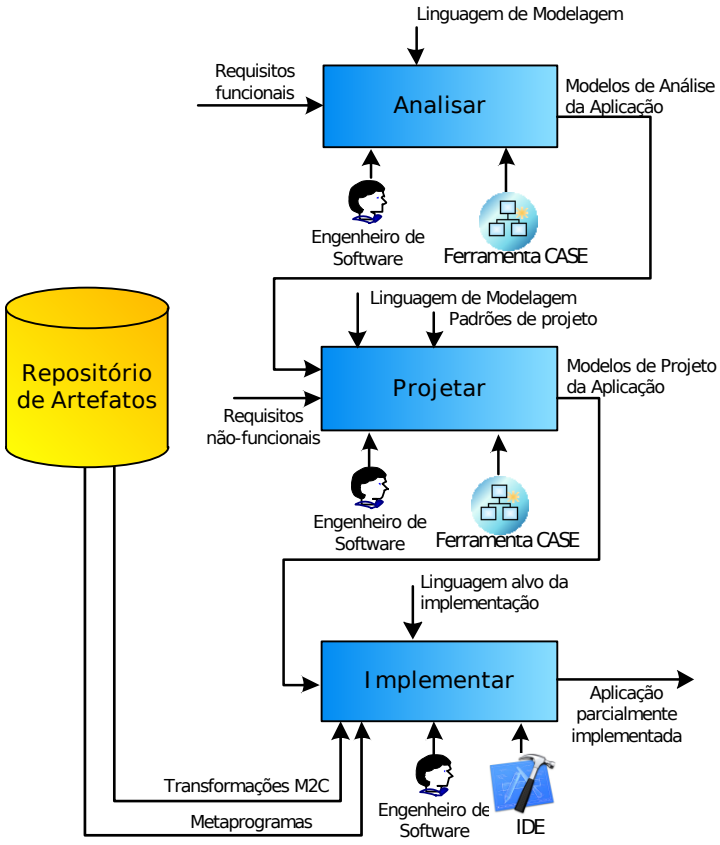


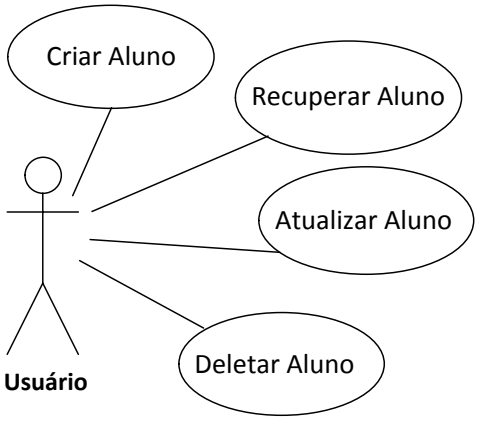
Figura 3.5: Engenharia de Aplicação do processo proposto.

Para melhor ilustrar essa etapa, o processo proposto foi instanciado para utilizar a UML como linguagem de modelagem e Java como linguagem alvo da implementação. Durante a explicação de cada atividade da EA, são apresentados exemplos aplicados no desenvolvimento de uma aplicação Web que implementa as funcionalidades CRUD de suas classes de entidade.

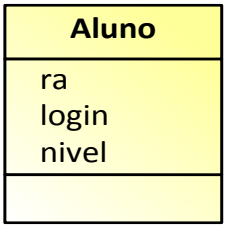
Analisar

Nessa atividade, a aplicação é inicialmente especificada a partir de seus requisitos funcionais. Conforme ilustrado na Figura 3.5, essa atividade é realizada pelo Engenheiro de Software auxiliado por uma ferramenta *Computer-Aided Software Engineering (CASE)* e orientada pela sintaxe e semântica da linguagem de modelagem utilizada, produzindo como saída os modelos de análise da aplicação.

Por exemplo, no caso da utilização da UML como linguagem de modelagem, é possível construir diagramas de caso de uso, classes e outros julgados necessários para a modelagem dos requisitos. A Figura 3.6 mostra alguns exemplo de modelos de análise produzidos na execução da atividade “Analisar” da aplicação Web com as funcionalidades CRUD.



(a) Diagrama de Caso de Uso de Aluno



(b) Classe de Aluno

Figura 3.6: Modelos de análise de uma aplicação Web com funcionalidades CRUD.

Projetar

Nessa atividade, o objetivo principal é definir os modelos de projeto da aplicação, obtidos por meio da extensão e refinamento dos modelos de análise da aplicação, realizado pelo Engenheiro de Software com base nos requisitos não-funcionais da

aplicação. Essa atividade também é auxiliada por uma ferramenta CASE, sendo orientada pela sintaxe e semântica da linguagem de modelagem e pela utilização de padrões de projeto.

Novamente, tomando a UML como linguagem de modelagem adotada, por exemplo, um diagrama de classes especificado durante a atividade “Analisar” pode ser refinado e incrementado com estereótipos, tipos dos atributos, aplicação de padrões de projeto, definição de métodos, uso de metamodelos de domínios específicos e outras decisões de projeto com a finalidade de agregar maior significado ao diagrama.

A Figura 3.7 mostra o diagrama de classes obtido por meio do refinamento do modelo de análise da aplicação Web com as funcionalidades CRUD. Na construção desse modelo, um perfil da UML foi utilizado para definir estereótipos com o objetivo de agregar maior significado aos modelos especificados. No Quadro 3.2, são apresentados exemplos de estereótipos definidos no perfil UML e seus respectivos significados.

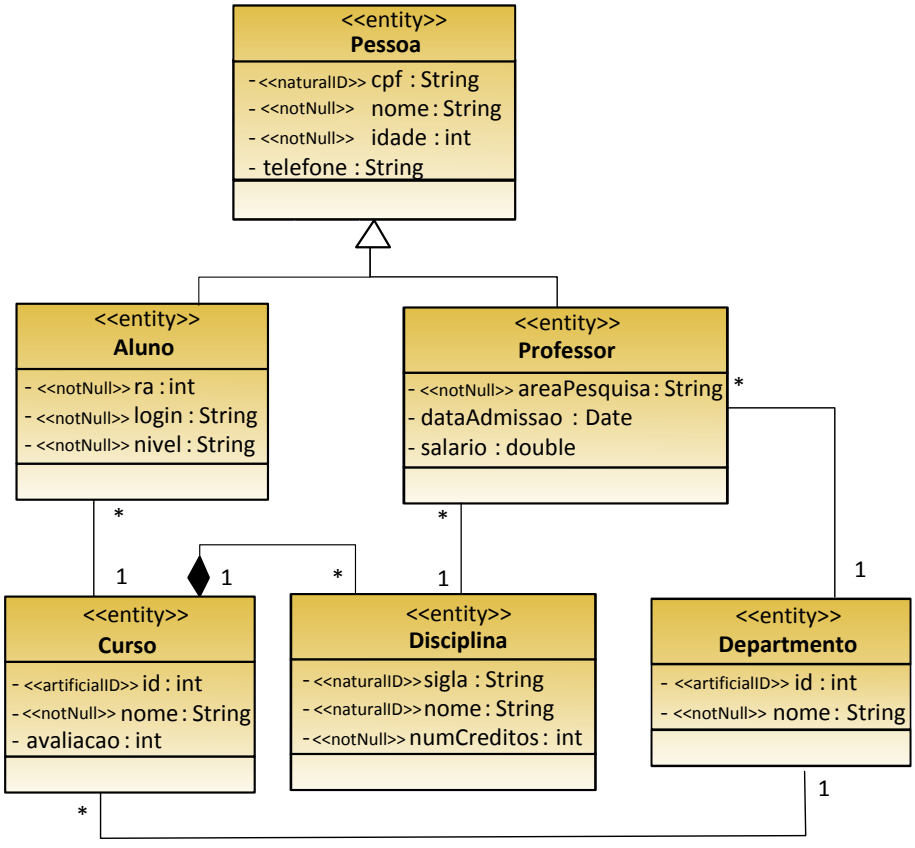


Figura 3.7: Modelo de projeto de uma aplicação Web com funcionalidades CRUD.

Quadro 3.2: Exemplos de estereótipos definidos em um perfil da UML.

	Estereótipo	Significado
Aplicável em classes	<<entity>>	Estereótipo aplicável a uma classe, indicando que será persistida em banco de dados.
Aplicável em atributos	<<notNull>>	O campo na tabela do banco de dados referente ao atributo não poderá ser nulo.
	<<unique>>	O campo na tabela do banco de dados referente ao atributo deverá ser único.
	<<naturalID>>	O campo na tabela do banco de dados referente ao atributo fará parte da chave primária natural.
	<<artificialID>>	O campo na tabela do banco de dados referente ao atributo será uma chave primária artificial.

Implementar

Uma vez finalizada a construção dos modelos de projeto da aplicação, seguindo o processo proposto, inicia-se de fato a implementação da aplicação por meio da geração de código a partir dos modelos construídos até o momento. Nessa atividade, ocorre o reúso dos artefatos (transformações M2C e metaprogramas) que foram previamente criados durante a ED.

Como se observa na Figura 3.8, essa atividade é subdividida em outras duas atividades: **Aplicar Transformações M2C** e **Aplicar Metaprogramas**.

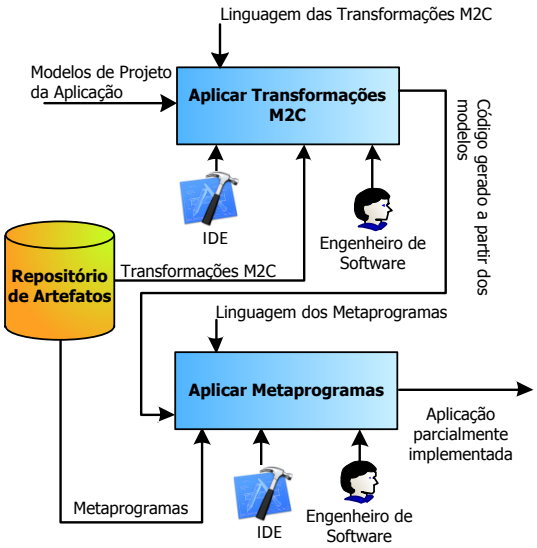


Figura 3.8: Subdivisões da atividade Implementar.

Na atividade Aplicar Transformações M2C, o Engenheiro de Software, auxiliado por uma IDE, seleciona e executa transformações M2C disponíveis no Repositório de Artefatos, com a finalidade de transformar os modelos de projeto da aplicação em código em uma linguagem de programação alvo da implementação.

Na atividade Aplicar Metaprogramas, com o auxílio de uma IDE, o Engenheiro de Software seleciona e executa metaprogramas disponíveis no Repositório de Artefatos. Os metaprogramas executados analisam o código gerado pelas transformações

e outra etapa de geração de código é realizada com o objetivo de complementar a geração de código feita pelas transformações M2C, gerando código de funcionalidades específicas, código referente à implementação de padrões de projeto e outros códigos definidos nos metaprogramas.

Para o exemplo do modelo de projeto da aplicação Web com as funcionalidades CRUD, ilustrado na Figura 3.7, foi selecionada a transformação M2C “UML para Java” (Figura 3.3) construída na ED. Essa transformação M2C gera o código Java de todas as classes com seus atributos, métodos, relacionamentos e anotações. Nesse caso, para executar essa transformação, o modelo de projeto foi exportado no formato seguindo o padrão XMI, conforme mostra a Figura 3.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
                           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
                           xmlns:profile="http:///schemas/profile/_9qaIUI0aEeCvOp9lI4h9FQ/10"
                           xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
                           xsi:schemaLocation=
                           "http:///schemas/profile/_9qaIUI0aEeCvOp9lI4h9FQ/10
                           myProfile.profile.uml#_9qaIUY0aEeCvOp9lI4h9FQ">
  <uml:Model xmi:id="_XOdmIJzZEeCGi4RLqB6Chg" name="model">
    <packagedElement xmi:type="uml:Package" xmi:id="_Sc8JIJzfEeCGi4RLqB6Chg" name="modelo">
      <packagedElement xmi:type="uml:Class" xmi:id="_iYJooJzfEeCGi4RLqB6Chg" name="Pessoa">
        <ownedAttribute xmi:id="_z_7XoJzfEeCGi4RLqB6Chg" name="cpf" visibility="private">
          <type xmi:type="uml:PrimitiveType" href="String"/>
          <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_mJ5mgJzgEeCGi4RLqB6Chg"
           value="1"/>
          <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_mJ4_cJzgEeCGi4RLqB6Chg" value=
          "1"/>
        </ownedAttribute>
        <ownedAttribute xmi:id="_EP0GoJzgEeCGi4RLqB6Chg" name="rg" visibility="private">
          <type xmi:type="uml:PrimitiveType" href="String"/>
          <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_5lICkJ0AEeCGi4RLqB6Chg"
           value="1"/>
          <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_5lHbgJ0AEeCGi4RLqB6Chg" value=
          "1"/>
        </ownedAttribute>
        <ownedAttribute xmi:id="_Fiz_kJzgEeCGi4RLqB6Chg" name="nome" visibility="private">
          <type xmi:type="uml:PrimitiveType" href="String"/>
        </ownedAttribute>
        <ownedAttribute xmi:id="_F9IkEJzgEeCGi4RLqB6Chg" name="idade" visibility="private">
          <type xmi:type="uml:PrimitiveType" href="int"/>
        </ownedAttribute>
        <ownedAttribute xmi:id="_Hvf5kJzgEeCGi4RLqB6Chg" name="telefone" visibility="private"
          <type xmi:type="uml:PrimitiveType" href="String"/>
        </ownedAttribute>
      </packagedElement>
      <packagedElement xmi:type="uml:Class" xmi:id="_s2AuoJzfEeCGi4RLqB6Chg" name="Aluno">
        <generalization xmi:id="_2HRpMJzgEeCGi4RLqB6Chg" general="_iYJooJzfEeCGi4RLqB6Chg"/>
        <ownedAttribute xmi:id="_6eMmIJzgEeCGi4RLqB6Chg" name="ra" visibility="private">
          .
          .
          .
  </uml:Model>
</xmi:XMI>
```

Figura 3.9: Modelo de projeto da aplicação CRUD no formato do padrão XMI.

A partir desse modelo em XMI, a transformação “UML para Java” é executada para realizar a geração de código na linguagem Java por meio das regras de transformação definidas durante sua construção (Figura 3.10). A Figura 3.11 mostra um trecho do código da classe Aluno que herda da classe Pessoa e associa-se com a classe Curso, gerado pela transformação M2C “UML para Java”.

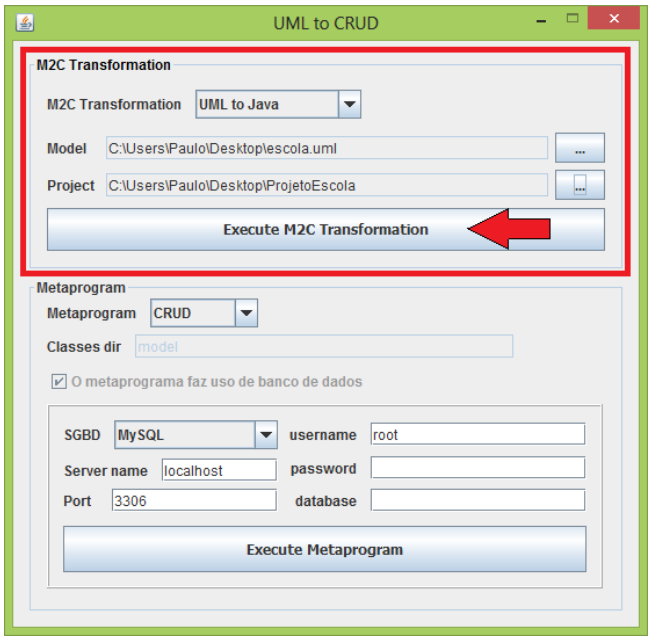


Figura 3.10: Execução da Transformação M2C “UML para Java”.

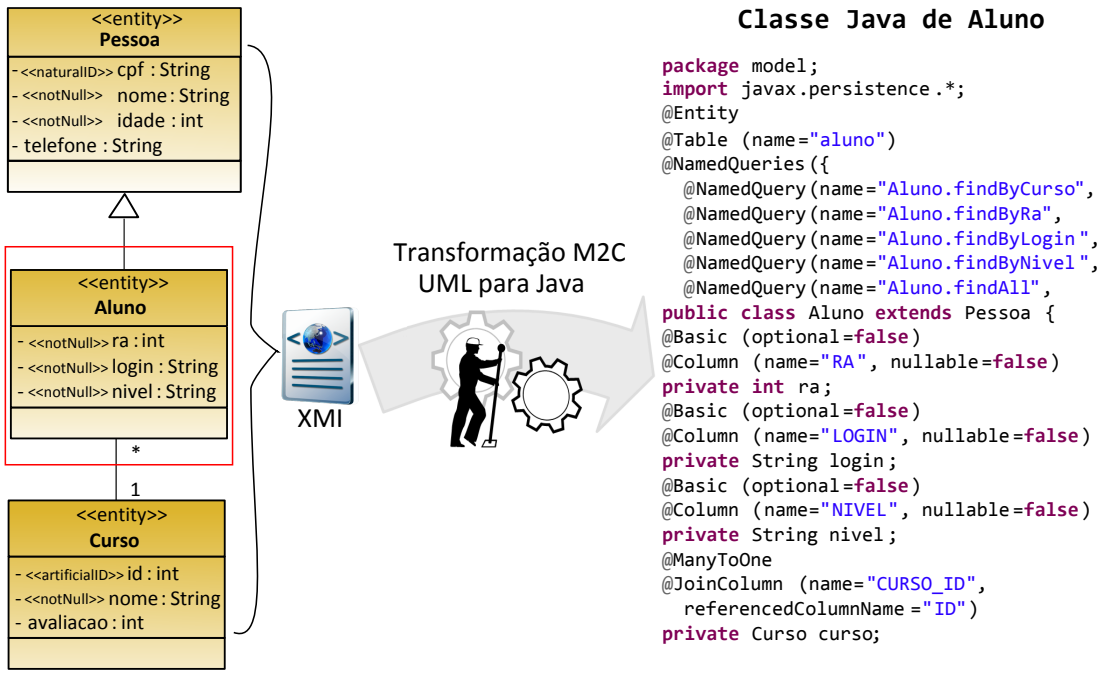


Figura 3.11: Geração de código Java a partir de um modelo de classes da UML.

Ainda nesse exemplo, uma vez terminada a geração de código feita pela transformação M2C, foi aplicado o metaprograma “M-CRUD” (Figura 3.12), construído na ED. Esse metaprograma processa o código das classes geradas pela Transformação M2C “UML para Java” e gera código de páginas Web, classes de acesso a banco de dados, classes auxiliares e código para a implementação de padrões de projeto, como mostra a Figura 3.13.

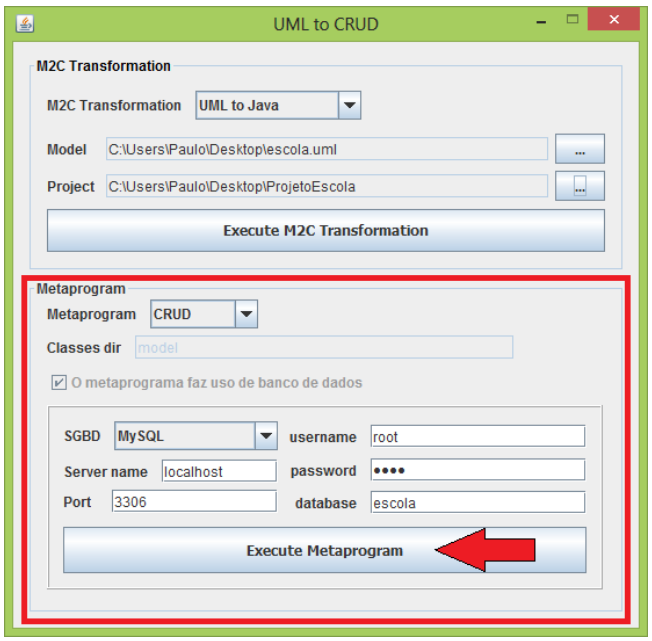
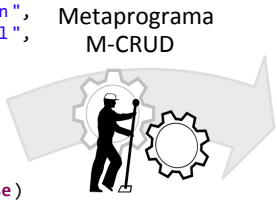


Figura 3.12: Execução do Metaprograma para geração de CRUD.

Classe Java de Aluno

```
package model;
import javax.persistence.*;
@Entity
@Table (name="aluno")
@NamedQueries({
  @NamedQuery(name="Aluno.findByCurso",
  @NamedQuery(name="Aluno.findByRa",
  @NamedQuery(name="Aluno.findByLogin",
  @NamedQuery(name="Aluno.findByNivel",
  @NamedQuery(name="Aluno.findAll",
public class Aluno extends Pessoa {
@Basic (optional=false)
@Column (name="RA", nullable=false)
private int ra;
@Basic (optional=false)
@Column (name="LOGIN", nullable=false)
private String login;
@Basic (optional=false)
@Column (name="NIVEL", nullable=false)
private String nivel;
@ManyToOne
@JoinColumn (name="CURSO_ID",
  referencedColumnName="ID")
private Curso curso;
```



Novo Aluno

cpf	[illegible]
nome	Paulo Eduado Papotti
idade	23
telefone	(016) 1234-5678
ra	298425
login	paulo.papotti
nivel	Graduação
curso	Ciência da Computação

Salvar Cancelar

Figura 3.13: Geração de uma página Web para cadastro de Alunos a partir de uma classe em Java.

Dado que uma das preocupações da geração de código no processo proposto está relacionada ao uso de padrões de projeto, no exemplo apresentado anteriormente, o metaprograma para geração de CRUD implementou os padrões MVC, *Facade* e *AbstractFactory*, conforme mostra a Figura 3.14. Nesse caso, os padrões presentes no código gerado contribuem para o desenvolvimento de soluções bem projetadas, o que implica em uma maior facilidade na compreensão do sistema e nas futuras manutenções.

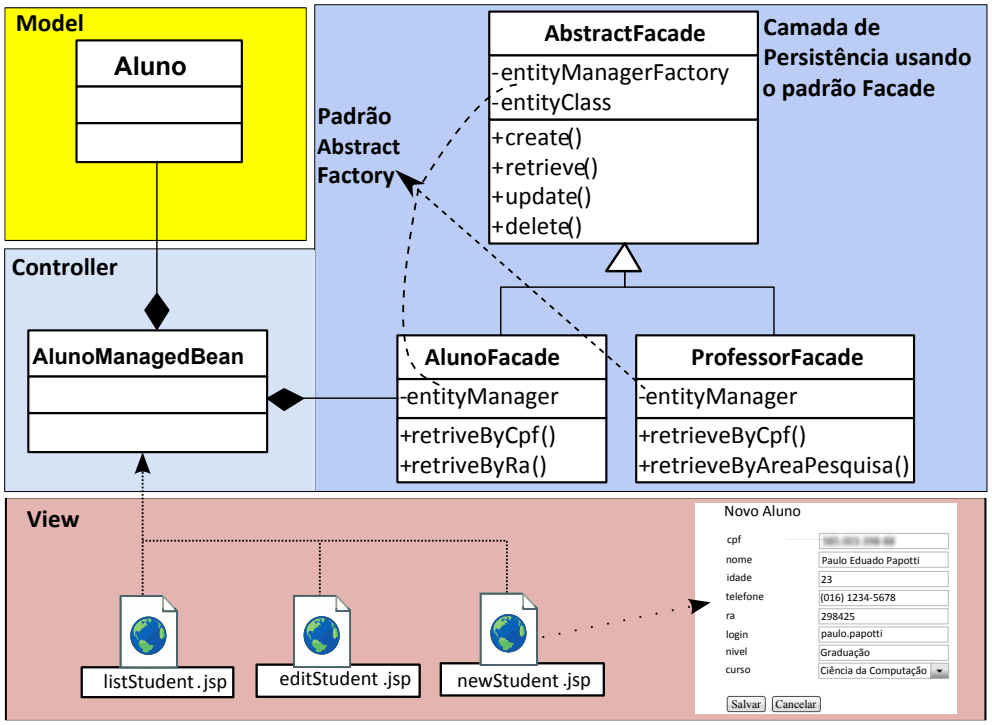


Figura 3.14: Representação do uso de padrões de projeto na aplicação gerada.

Ao final do processo, o resultado é uma aplicação funcional parcialmente implementada. Dessa forma, o código e os modelos gerados ao longo do processo proposto podem ser incrementados de acordo com requisitos mais específicos a serem atendidos pela aplicação, como por exemplo, utilização de folhas de estilos e implementação de algoritmos ou funcionalidades não providas pela geração de código.

3.3 Adaptação do Processo para Utilização na Reengenharia de Sistemas Legados

Um dos motivos do processo proposto ter sido projetado como sendo de propósito geral é proporcionar sua adaptação para utilização em diferentes contextos. De acordo com as necessidades de cada aplicação, o processo proposto pode ser adaptado ou modificado com o objetivo de explorar o uso de geração de código de diferentes formas. Um dos contextos alvo para a aplicação do processo proposto foi o de reengenharia de software com o objetivo de apoiar a migração de sistemas legados para utilização do paradigma MDD.

O processo proposto foi modificado com o objetivo de apoiar a reengenharia de sistemas legados a partir de seus bancos de dados e código fonte, visando a adoção do MDD. Nesse sentido, diversas etapas de reconstrução do software, sobretudo na modelagem e reimplementação, podem ser parcialmente automatizadas (DUVALL; MATYAS; GLOVER, 2007), com o objetivo de reduzir a ocorrência de erros e obter um ganho de tempo e esforço na reengenharia de um sistema.

Nesse caso, o processo de reengenharia é realizado de modo semelhante ao apresentado durante a definição do processo proposto. As etapas de Engenharia de Domínio e Engenharia de Aplicação foram mantidas, mas algumas modificações e complementos foram introduzidos para a realização do processo de reengenharia de software.

3.3.1 Engenharia de Domínio

De modo geral, na ED do processo proposto não houve muitas modificações com relação ao processo original. A única modificação realizada foi definir uma atividade para realizar a construção de metaprogramas destinados à geração de modelos a partir de um código especificado em uma linguagem de programação. Essa atividade é necessária para dar apoio à etapa de Engenharia Reversa durante o processo de reengenharia.

Por outro lado, na EA, houve diversas modificações para adaptar as atividades do processo proposto para o contexto de reengenharia de software. Dessa forma, as atividades originais deram lugar às atividades de Engenharia Reversa e Engenharia Avante.

3.3.2 Engenharia de Aplicação

Como mostra o diagrama SADT da Figura 3.15, a EA do processo de reengenharia é dividida em duas principais atividades: Engenharia Reversa (ER) e Engenharia Avante (EAv). A EA se inicia com a ER onde, caso a aplicação utilize um banco de dados, um *framework* de persistência ORM e um IDE são utilizados para realizar a geração de código das classes de entidades persistentes no banco de dados da aplicação. Com base nesse código gerado e no código legado da aplicação, metaprogramas analisam a estrutura das classes e geram um modelo OO correspondente. Esse modelo é descrito em UML, por meio do padrão XMI, e visualizado em uma ferramenta CASE.

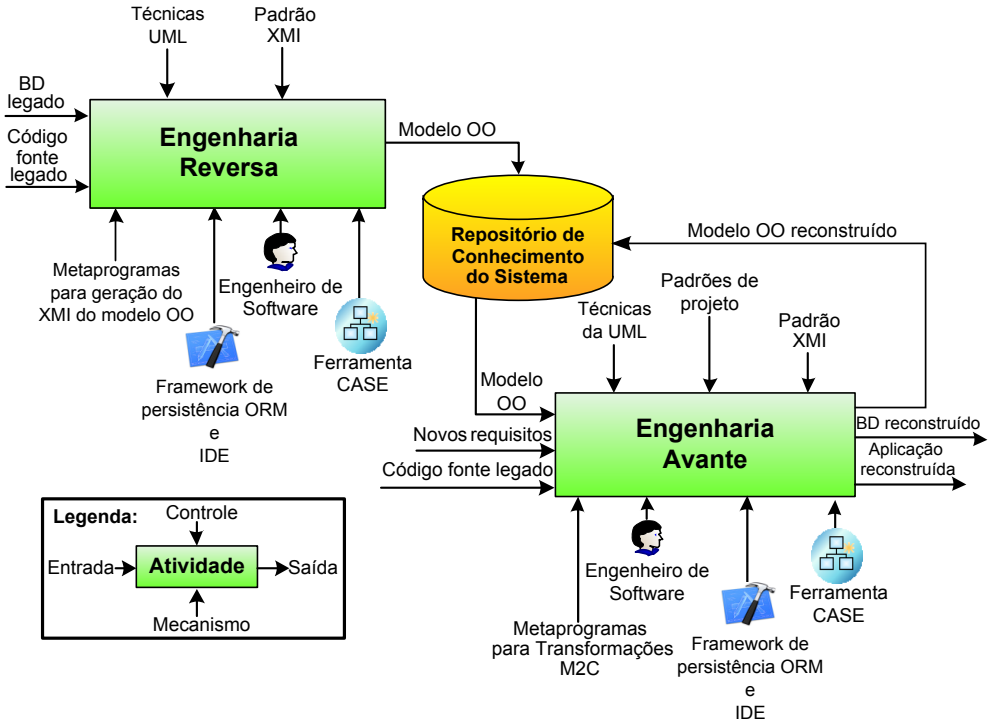


Figura 3.15: Visão geral da EA do processo de reengenharia proposto.

Na EAv, conforme a Figura 3.15, a aplicação é reconstruída seguindo os princípios do MDD. O modelo OO obtido na ER é refinado e outros modelos podem ser especificados usando a UML, com base nos novos requisitos da aplicação e no código fonte legado. Uma vez finalizada a modelagem, transformações M2C e metaprogramas são executados para realizar a geração de código de estruturas e funcionalidades da aplicação, como o código dos atributos, protótipos dos métodos e interfaces CRUD, reduzindo os esforços necessários na implementação da nova aplicação.

Engenharia Reversa

A Engenharia Reversa (ER) é o processo de análise de uma aplicação legada para identificar seus componentes e inter-relacionamentos, criando representações da mesma em outra forma ou em um nível mais alto de abstração (CHIKOFSKY; CROSS et al., 1990). Nessa etapa, o objetivo é obter um modelo orientado a objetos, descrito por meio da UML, das classes de uma aplicação legada.

Essa etapa inicia na atividade **Gerar Código das Classes de Entidade**. Essa atividade é realizada apenas se a aplicação legada fizer uso de um banco de dados. Nesse caso, o modelo físico das entidades do banco de dados da aplicação legada é mapeado no código fonte correspondente das classes de entidades, utilizando o paradigma orientado a objetos. Nessa atividade, o Engenheiro de Software reutiliza um *framework* de persistência ORM e uma IDE para acessar a estrutura física do banco de dados e realizar a geração de código das classes que mapeiam as entidades persistentes em uma linguagem de programação alvo. A Figura 3.16 ilustra a execução dessa atividade, em que o código de uma classe de entidade, implementada na linguagem Java, é gerado a partir de uma tabela de banco de dados, por meio do uso de um *framework* de persistência ORM.

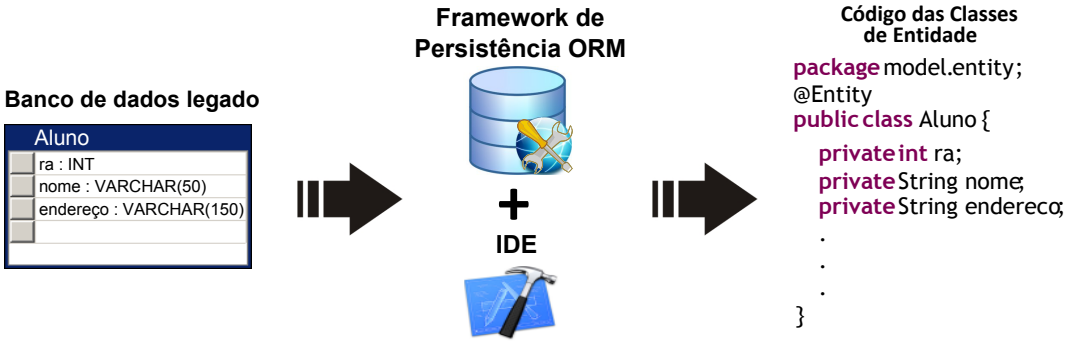


Figura 3.16: Geração de uma classe de entidade a partir do banco de dados.

Na atividade **Executar Metaprogramas para Geração do Modelo OO**, o Engenheiro de Software seleciona e executa metaprogramas, previamente construídos, destinados à geração de modelos a partir do código de classes implementadas. Conforme mostra a Figura 3.17, a partir da representação do código da aplicação legada e das classes de entidades geradas pelo *framework* de persistência ORM, os metaprogramas geram *tags* em um arquivo no padrão XMI, que representam a informação da estrutura dos elementos de cada classe analisada. Ao final da atividade, tem-se um modelo OO descrito na UML, contendo as classes de entidade e da aplicação legada.

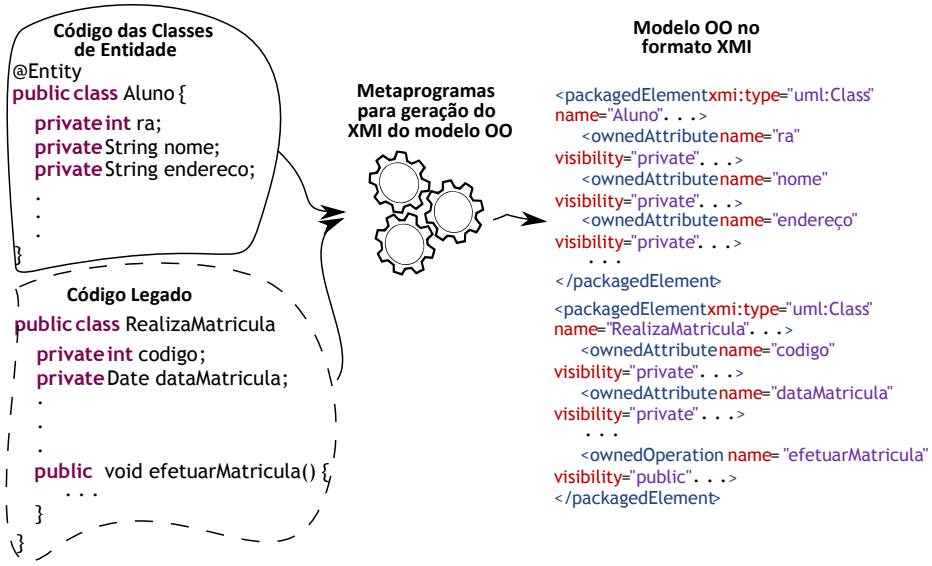


Figura 3.17: Fragmento de um modelo OO, no padrão XMI, gerado a partir de classes em Java.

Finalizada a geração do XMI do modelo OO das classes, na atividade **Importar XMI**, uma ferramenta CASE compatível com o padrão XMI é utilizada pelo Engenheiro de Software para obter a visualização gráfica do modelo OO em UML. Como mostra a Figura 3.18, o arquivo do modelo OO no padrão XMI é importado e interpretado pela ferramenta CASE, que reproduz sua representação gráfica correspondente na UML. Diferentes ferramentas CASE disponíveis no mercado, tanto proprietárias quanto gratuitas, possuem mecanismos para apoiar esse processo de importação.

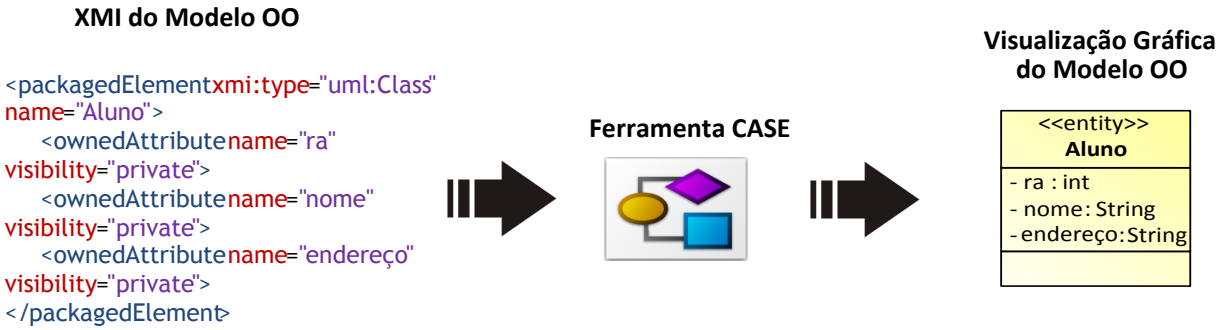


Figura 3.18: Importação de um modelo OO, no padrão XMI, e sua representação gráfica gerada em uma ferramenta CASE.

Após a obtenção da representação gráfica do modelo OO das classes, a etapa de Engenharia Reversa está concluída. Dessa forma, o modelo OO obtido pode ser refinado para atender os novos requisitos da aplicação na etapa de Engenharia Avante.

Engenharia Avante

A Engenharia Avante (EAv) é o processo de levar abstrações de alto nível até a implementação física de um sistema (CHIKOFSKY; CROSS et al., 1990). Nessa etapa, o objetivo é realizar o refinamento do modelo OO e a geração de código parcial da aplicação, seguindo os princípios do MDD.

Essa etapa inicia com a atividade **Refinar Modelo**, que compreende as disciplinas de Análise e Projeto do processo de desenvolvimento de software. Essa atividade é essencial para o desenvolvimento da nova aplicação, uma vez que o modelo OO gerado na ER necessita ser refinado de acordo com novos requisitos e especificações não contempladas pela geração de código.

Para realizar o refinamento do modelo OO, o Engenheiro de Software, parte da visualização gráfica do modelo OO gerado na ER e, apoiado por uma ferramenta CASE, analisa o modelo e especifica as alterações necessárias, tendo como base o código fonte da aplicação legada. Por exemplo, novas classes persistentes e não persistentes podem ser adicionadas ao modelo de classes OO, assim como novos atributos e relacionamentos. A Figura 3.19 ilustra essa atividade, em que novas classes são criadas para representar componentes de sessão no padrão Fachada e serviços Web.

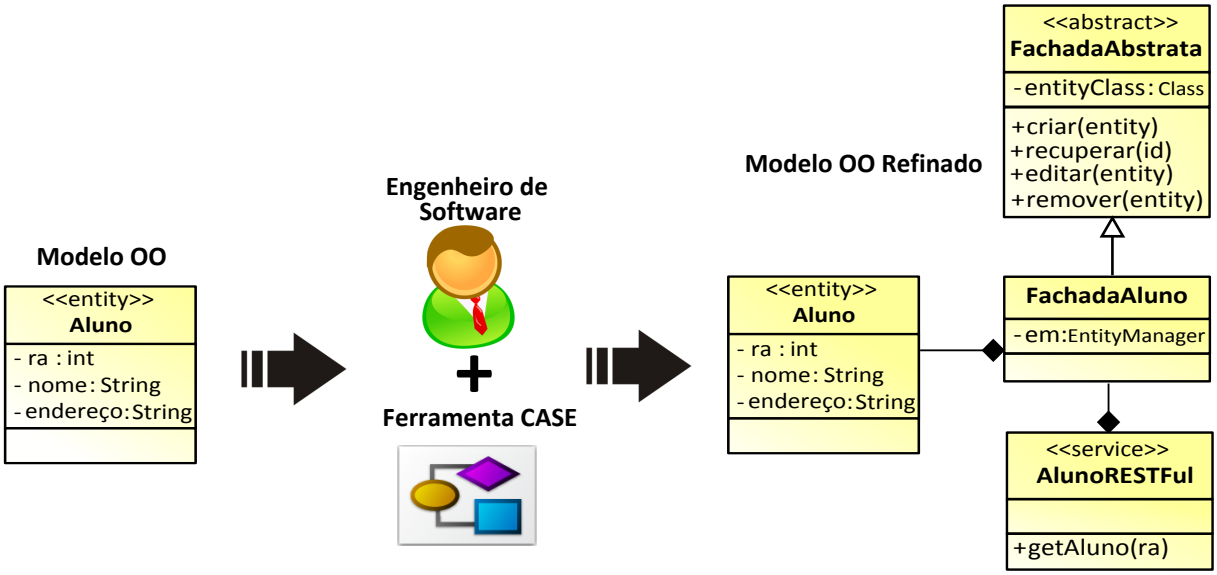


Figura 3.19: Refinamento de um modelo de classes.

É importante ressaltar que, na atividade Refinar Modelo, novos modelos utilizando a UML também podem ser especificados com a finalidade de tornar a modelagem da aplicação mais robusta e significativa. Por exemplo, diagramas de caso de uso, dia-

gramas de sequência e outros diagramas da UML podem ser utilizados para mapear as regras de negócio da aplicação.

Uma vez concluída a modelagem da aplicação, o processo prossegue com a atividade **Executar Transformações M2C/Metaprogramas**. Nessa atividade, geradores são executados para transformar os modelos e realizar a geração parcial do código da aplicação. O Engenheiro de Software, auxiliado por uma *Integrated Development Environment (IDE)* e uma ferramenta CASE, seleciona e executa transformações M2C, tendo como resultado o código gerado a partir dos modelos. Essas transformações acessam o XMI dos modelos de entrada e geram o código fonte de saída, em uma linguagem de programação alvo da implementação, com base nas regras de transformação definidas em cada transformação. Adicionalmente, metaprogramas, existentes no repositório de artefatos, são executados para complementar a geração de código realizada por transformações M2C, tal como definido no processo proposto.

Com o objetivo de tornar o entendimento do código gerado mais fácil por parte dos desenvolvedores, a geração de código é realizada com base em padrões de projeto. Dessa forma, o código gerado apresenta soluções mais fáceis de serem mantidas e modificadas. Por exemplo, a Figura 3.20 ilustra a geração de código realizada por transformações M2C e metaprogramas para geração de código de classes de entidades, classes não-persistentes e classes de serviços Web usando a linguagem Java. Nesse exemplo, o código gerado faz uso dos padrões MVC e Fachada (GAMMA et al., 1995), tendo como base o uso de perfis da UML, contendo estereótipos como `«entity»`, `«service»` e outros. Os estereótipos aplicados aos modelos orientam o processo de geração de código, dando maior significado aos modelos e resolvendo questões como a incompatibilidade de tipos entre as classes do modelo OO e o banco de dados.

Após a geração de código da aplicação, a atividade **Gerar Banco de Dados Reconstruído** é realizada. Essa atividade somente é realizada se a aplicação fizer uso de um banco de dados. Nessa atividade, o *framework* de persistência ORM é reutilizado para realizar a geração do modelo físico do banco de dados a partir do código das classes de entidades persistentes. Dessa forma, as modificações especificadas no modelo OO podem ser refletidas na estrutura das tabelas existentes no banco.

Ao final, tem-se a aplicação reconstruída parcialmente implementada, o banco de dados reconstruído e a modelagem da aplicação consistente com seu código. A aplicação gerada pode ser complementada de acordo com as especificações mais

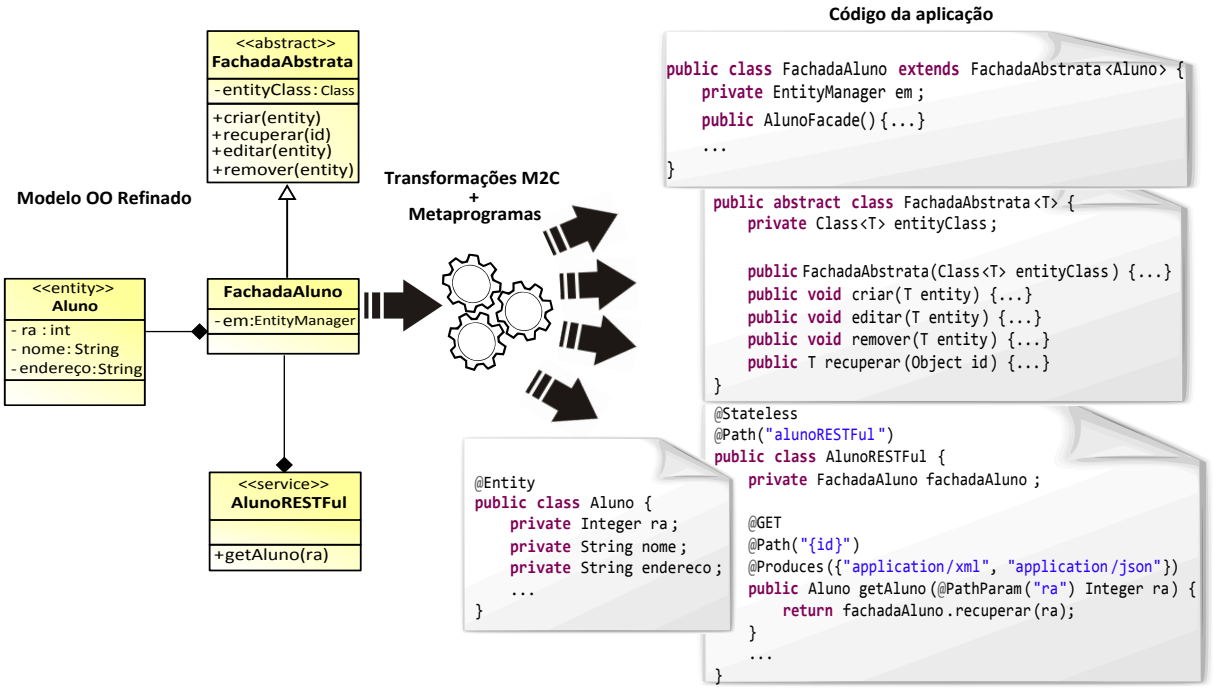


Figura 3.20: Geração de código utilizando os padrões MVC e Fachada, realizada a partir de um modelo de classes por meio de transformações M2C e metaprogramas.

detalhadas das regras de negócio da aplicação. A aplicação reconstruída, conforme o processo proposto, beneficia-se das vantagens de reutilizar o conhecimento embutido no banco de dados e da utilização do MDD, produzindo e mantendo a modelagem consistente com o código gerado. A finalização da implementação, assim como as futuras manutenções da aplicação são realizadas seguindo os princípios do MDD, com base nos modelos e artefatos construídos durante o processo.

3.4 Considerações Finais

Este capítulo apresentou um processo de software para realizar geração de código na construção de aplicações, seguindo os princípios do MDD e utilizando técnicas de metaprogramação. O processo proposto se concentra na construção de artefatos reutilizáveis de geração de código (transformações M2C e metaprogramas) e no desenvolvimento de aplicações com base em modelos definidos durante a modelagem.

Com base nos conceitos de LPS, o processo proposto é realizado em duas etapas: Engenharia de Domínio (ED) e Engenharia de Aplicação (EA). A ED engloba as atividades referentes à construção de artefatos reutilizáveis que apoiam o desenvolvimento de aplicações por meio de geração de código. Esses artefatos incluem

transformações M2C (para geração de código referente aos elementos especificados nos modelos construídos) e metaprogramas (para complementar o código gerado por transformações M2C e realizar a geração de funcionalidades utilizando padrões de projeto). Por sua vez, a EA define diretrizes para o desenvolvimento de aplicações com reúso dos artefatos construídos na ED.

Com o objetivo de apoiar a reengenharia de sistemas legados, o processo proposto foi adaptado para apoiar a geração de código na reconstrução de sistemas legados. A EA do processo de reengenharia foi dividida nas etapas de Engenharia Reversa (ER) e Engenharia Avante (EAv). Na ER, um modelo OO é obtido a partir de classes da aplicação legada e classes de entidade geradas a partir do banco de dados da aplicação legada. Na EAv, esse modelo OO é refinado e os artefatos de geração de código reutilizáveis automatizam a escrita de código da aplicação reconstruída.

Capítulo 4

EXPERIMENTAÇÃO DA PROPOSTA

4.1 Considerações Iniciais

Neste Capítulo, são reportados os resultados de um estudo quantitativo, por meio da metodologia experimental (WOHLIN et al., 2000). Esse estudo avaliou o impacto da utilização de geração de código dirigida a modelos quanto ao tempo gasto por equipes no desenvolvimento de sistemas com e sem o uso de geração de código. Durante o experimento, em um primeiro momento, as equipes desenvolveram uma aplicação Web seguindo as disciplinas do ciclo de vida clássico (PRESSMAN, 2005), sem o uso de geração de código a partir de modelos. Em um outro momento, as equipes realizaram o desenvolvimento seguindo o processo proposto, com a utilização de geradores a partir de modelos definidos. A análise dos resultados teve como base os dados coletados referentes ao tempo gasto pelas equipes participantes. A partir dos resultados, uma redução expressiva no tempo de desenvolvimento foi evidenciada com o uso de mecanismos de geração de código, além da diminuição da ocorrência de dificuldades enfrentadas pelas equipes. A organização do experimento seguiu uma estrutura semelhante a apresentada por Cirilo et al. (2011).

Este Capítulo está organizado da seguinte forma: a Seção 4.2 introduz alguns conceitos sobre os princípios da Experimentação, que é completamente detalhada na Seção 4.3. Por fim, o capítulo apresenta algumas considerações finais na Seção 4.4.

4.2 Princípios da Experimentação

O crescimento do número de trabalhos da comunidade científica com uma validação empírica significativa contribui para acelerar o processo de formação da Engenharia de Software como ciência (CONRADI et al., 2001). O processo de experimentação fornece um modo sistemático, disciplinado, computável e controlado para a avaliação da atividade humana (TRAVASSOS; GUROV; AMARAL, 2002).

Na Engenharia de Software, a experimentação possui grande importância, dado que possibilita realizar o teste de hipóteses sobre uma determinado objeto de estudo (e.g. processo, método, ferramenta, recurso, modelo, teoria) e, dessa forma, observar os efeitos resultantes da sua adoção na prática. A partir de medições realizadas, os dados são analisados e as hipóteses formuladas podem ser validadas ou refutadas (WOHLIN et al., 2000; TRAVASSOS; GUROV; AMARAL, 2002). Na Figura 4.1 está ilustrada a estrutura de um experimento.

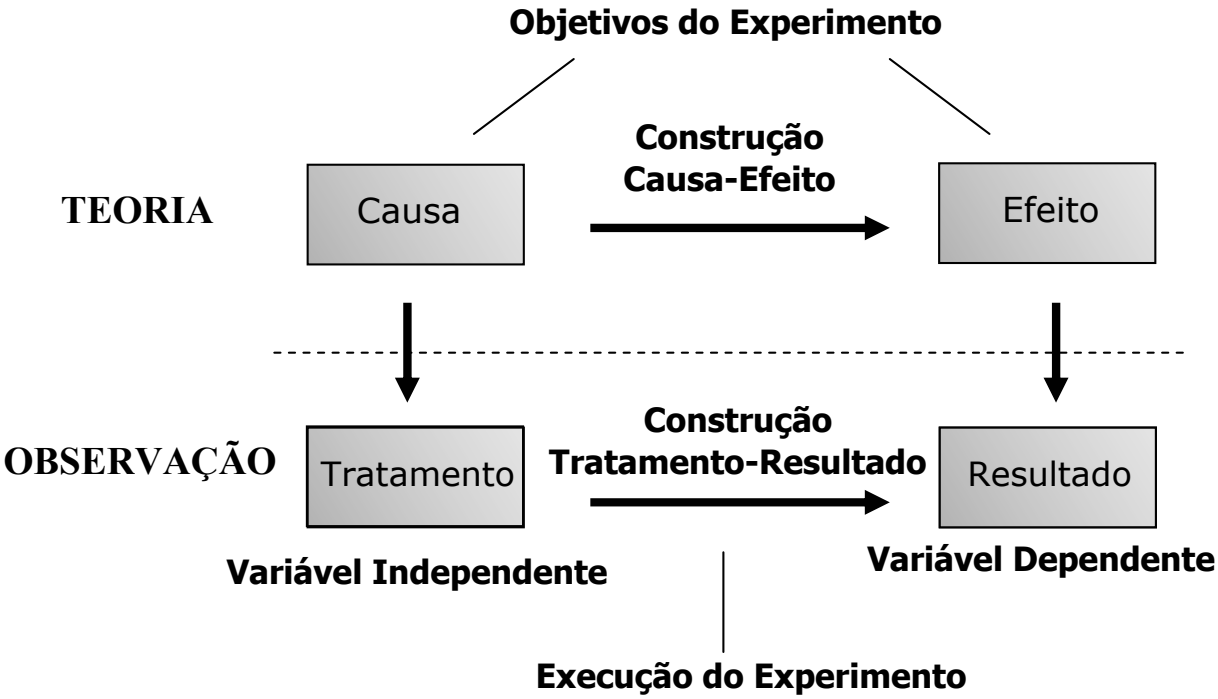


Figura 4.1: Estrutura geral de um experimento (WOHLIN et al., 2000).

A hipótese nula (denotada por H0) é a hipótese principal do experimento e indica que não há um relacionamento com significância estatística entre a causa e o efeito que se propõe investigar. Geralmente, em um experimento, o objetivo é refutar essa hipótese nula em favor de uma ou mais hipóteses alternativas (H1, H2, H3, ..., Hn).

Na realização de um experimento, existem as variáveis independentes e as va-

riáveis dependentes. As variáveis independentes referem-se aos elementos que são perturbados ou mantidos durante o experimento com o objetivo de identificar as causas que afetam o resultado do experimento. Essas variáveis também são chamadas de fatores. Já as variáveis dependentes são os elementos que são testados ao longo da execução. O valor atribuído a um determinado fator dá-se o nome de tratamento. Dessa forma, os tratamentos devem ser aplicados sobre os objetos de estudo do experimento e por um conjunto de participantes.

4.3 Experimentação Realizada

A experimentação do processo proposto seguiu as fases experimentais, como proposto por (WOHLIN et al., 2000) e foi conduzida no segundo semestre de 2011. O experimento consistiu de uma análise comparativa do tempo gasto na implementação das funcionalidades CRUD de sistemas Web a partir de modelos (diagramas de classes da UML) das classes de entidade da aplicação, utilizando tanto o processo proposto quanto o processo com base no ciclo de vida clássico para realizar a implementação.

Na realização do experimento, foi utilizada uma parte do sistema PrograduWeb[1], da Universidade Federal de São Carlos - UFSCar. O PrograduWeb é um grande sistema acadêmico em funcionamento desde meados de 1998, que controla todas as informações sobre os alunos, professores, matrícula, emissão de diplomas e certificados e outras informações a respeitos dos cursos de graduação da UFSCar.

A tarefa dos participantes do experimento foi desenvolver uma aplicação que realizasse as funcionalidades CRUD a partir das classes de entidade de um diagrama de classes da UML referentes a um banco de dados relacional, seguindo as disciplinas convencionais de Análise, Projeto, Implementação e Testes da Engenharia de Software.

4.3.1 Definição

O experimento realizado foi definido com o seguinte objetivo:

- **Analisar** a utilização do processo proposto na construção de aplicações Web;

[1] PrograduWeb - https://prograduweb.ufscar.br/prograduweb

- **Com propósito** de avaliar o processo proposto quanto ao apoio à construção de aplicações, por meio de geração de código;
- **Com respeito à** eficiência em termo de tempo despendido;
- **Do ponto de vista** de desenvolvedores de software;
- **No contexto** de estudantes de graduação em Ciência e Engenharia da Computação.

4.3.2 Planejamento

O planejamento do experimento foi realizado por meio das seguintes etapas:

Seleção do Contexto

O experimento foi conduzido em ambiente universitário, no Laboratório de Ensino do Departamento de Computação da UFSCar, no âmbito da disciplina de Tópicos em Computação 2011. O experimento envolveu a participação de estudantes do terceiro e quarto anos dos cursos de Ciência da Computação e Engenharia da Computação da UFSCar.

Formulação das Hipóteses

Visando determinar o efeito da utilização do processo de desenvolvimento no resultado obtido, foram elaboradas três hipóteses para o experimento. Para a formulação das hipóteses, foram consideradas as seguintes métricas:

τ - Tempo total gasto pela equipe para o desenvolvimento de uma aplicação Web com as funcionalidades CRUD.

$\mu\tau$ - Tempo médio despendido pelas equipes para o desenvolvimento de uma aplicação Web com as funcionalidades CRUD.

As hipóteses formuladas para o experimento são:

Hipótese Nula (H0): Em geral, não há diferença entre equipes utilizando o processo proposto e equipes utilizando o processo baseado no ciclo de vida clássico para

a construção de aplicações Web, com respeito à eficiência (ε) da equipe.

$$H0 : \varepsilon_{Processo} = \varepsilon_{Clássico} \Rightarrow \mu\tau_{Processo} = \mu\tau_{Clássico}$$

Hipótese Alternativa (H1): Equipes utilizando o processo proposto para a construção de aplicações Web são, em geral, mais eficientes do que equipes utilizando o ciclo de vida clássico.

$$H1 : \varepsilon_{Processo} > \varepsilon_{Clássico} \Rightarrow \mu\tau_{Processo} < \mu\tau_{Clássico}$$

Hipótese alternativa (H2): Equipes utilizando o ciclo de vida clássico para a construção de aplicações Web são, em geral, mais eficientes do que equipes utilizando o processo proposto.

$$H2 : \varepsilon_{Processo} < \varepsilon_{Clássico} \Rightarrow \mu\tau_{Processo} > \mu\tau_{Clássico}$$

Seleção das Variáveis

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

- **Escolha das variáveis independentes:** As variáveis independentes que foram selecionadas na realização do experimento incluem a *aplicação* desenvolvida; o *processo de desenvolvimento* utilizado na construção da aplicação; o *ambiente*; e as *tecnologias* utilizadas. Dado que o intuito do experimento é investigar o impacto do uso dos processos na eficiência da equipe, a variável "*processo de desenvolvimento*" foi escolhida como fator que receberia tratamentos distintos. As demais variáveis independentes permaneceram constantes durante a execução do experimento:
  - **Aplicação:** CRUD de parte do ProgradWeb.
  - **Ambiente de desenvolvimento:** IDE Eclipse e banco de dados MySQL.

  - **Tecnologias de Desenvolvimento:** Java, Hibernate, Java Persistence API (JPA), Java Server Faces (JSF) and XHTML.

- **Escolha das variáveis dependentes:** A *eficiência* da equipe foi a variável dependente selecionada para a execução do experimento.

Seleção dos participantes

A seleção dos participantes do experimento foi realizada segundo a **técnica de amostragem não-probabilística por conveniência** (WOHLIN et al., 2000), de modo que os participantes selecionados eram os mais próximos e mais convenientes para realizar o experimento. De forma voluntária, 29 alunos do 3º e 4º anos de graduação dos cursos de Bacharelado em Ciência da Computação e Engenharia da Computação da UFSCar, matriculados na disciplina Tópicos em Computação 2011, participaram do experimento.

Projeto do experimento

O experimento realizado seguiu o princípio geral de projeto de agrupamento dos participantes **em blocos** (WOHLIN et al., 2000) homogêneos, de modo que o fator *nível de experiência dos participantes* não impactasse diretamente nos resultados dos tratamentos do fator *processo de desenvolvimento*, visando aumentar a precisão do experimento.

Os participantes foram divididos em 9 grupos (blocos) homogêneos, sendo 7 grupos compostos de 3 participantes e 2 grupos compostos de 4 participantes. Essa divisão dos grupos foi realizada de modo que o nível de experiência médio do grupo fosse o mais próximo possível.

Para determinar o nível de experiência de cada participante, foi utilizado um *Formulário de Caracterização dos Participantes* (Apêndice A), de modo que os participantes responderam questões de múltipla escolha a respeito do próprio conhecimento sobre assuntos abordados no experimento (e.g. conhecimentos sobre Java, Hibernate, JSF, UML) e, dessa forma, foi possível quantificar o nível de experiência de cada participante relacionado às tecnologias utilizadas.

O gráfico da Figura 4.2 exibe os níveis de experiência individuais de cada participante (números no topo de cada barra) e o nível médio de experiência dos grupos

(rótulos sobre cada grupo de barras adjacentes), conforme as informações apresentadas pelos alunos em seus respectivos formulários de caracterização. Esses níveis foram obtidos pelo cálculo da média do grau de conhecimento (escalas de 4 e 5 pontos) em cada questão respondida pelos participantes nos formulários. Além disso, a alocação dos participantes nos grupos foi realizada de maneira desbalanceada para refletir equipes com número variado de membros.

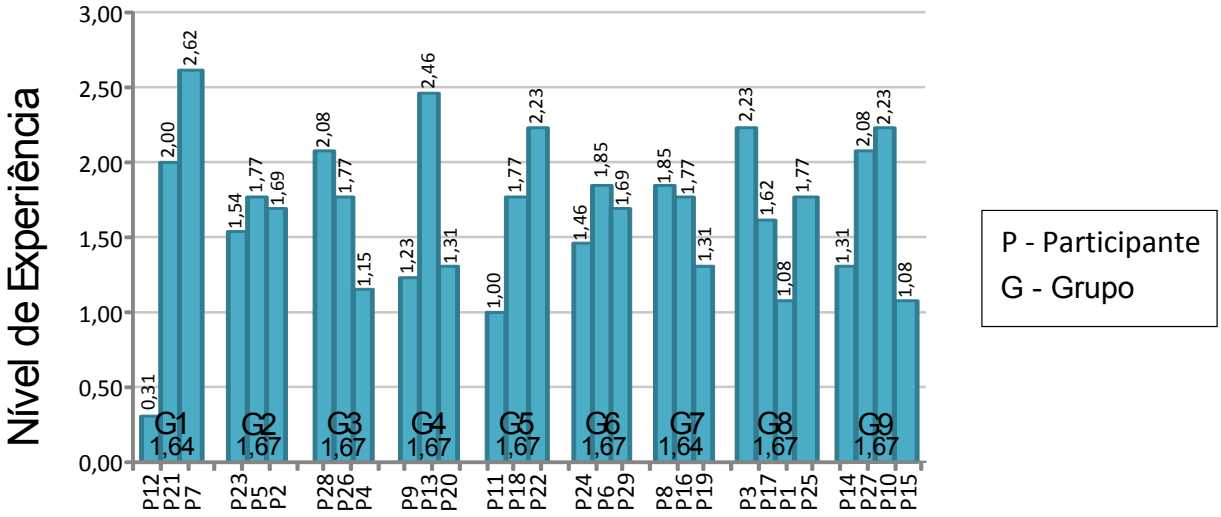


Figura 4.2: Distribuição dos participantes no experimento.

O experimento seguiu um tipo de projeto de **comparação pareada** (WOHLIN et al., 2000) de **um fator** (processo de desenvolvimento) com **dois tratamentos** (processo proposto e ciclo de vida clássico). Nesse tipo de projeto, cada indivíduo (ou grupo de indivíduos) utiliza ambos os tratamentos no mesmo objeto de estudo. Nesse caso, a ordem que os participantes aplicam os tratamentos é definida de forma aleatória. Dessa forma, na experimentação realizada, a aplicação foi desenvolvida utilizando ambos os processos por todos os grupos participantes. A Figura 4.3 mostra a ordem sorteada para a aplicação dos tratamentos pelos grupos. O número 1 indica qual tratamento foi aplicado primeiro pelo grupos e o número 2 indica o tratamento foi aplicado em uma segunda etapa.

Instrumentação

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

Grupo	Processo Proposto	Ciclo de Vida Clássico
G1	1	2
G2	1	2
G3	2	1
G4	1	2
G5	2	1
G6	2	1
G7	2	1
G8	1	2
G9	1	2

Figura 4.3: Atribuição dos tratamentos pelos grupos no experimento.

4.3.3 Execução

Uma vez que o experimento foi definido e planejado, essa fase descreve a execução do experimento propriamente dita, envolvendo as etapas de preparação, execução e validação dos dados coletados.

Preparação

Nessa etapa, os alunos participantes do experimento foram conscientizados a respeitos dos objetivos da pesquisa e sobre a intenção do experimento. Durante o preparo do experimento, todos os participantes foram informados e aceitaram os termos a respeito da confidencialidade dos dados fornecidos (sendo utilizados apenas para fins acadêmicos) e sobre a liberdade de desistência, por meio do *Formulário de Consentimento* (Apêndice B).

Nessa etapa, os seguintes objetos foram produzidos para uso no experimento:

- **Formulário de Caracterização do Participante:** questionário contendo perguntas para cada participante avaliar o próprio conhecimento sobre as tecnologias e conceitos usados no experimento (Apêndice A).
- **Formulário de Consentimento**: documento lido e assinado pelos participantes do experimento, informando os objetivos do estudo, o caráter confidencial e a liberdade de desistência (Apêndice B).
- **Descrição da Tarefa (Ciclo de Vida Clássico)**: documento utilizado para con-

textualizar e descrever a tarefa a ser realizada pelos participantes do experimento, utilizando o Ciclo de Vida Clássico (Apêndice C).

- **Material de apoio (Processo Ciclo de Vida Clássico)**: roteiro de tarefas a serem realizadas pelos participantes no experimento, seguindo o Ciclo de Vida Clássico (Apêndice E).
- **Formulário de Coleta de Dados (Ciclo de Vida Clássico)**: formulário preenchido pelos participantes, indicando o horário de início e o horário de fim de cada tarefa realizada no experimento, utilizando o Ciclo de Vida Clássico (Apêndice G).
- **Formulário de Avaliação (Ciclo de Vida Clássico)**: documento de avaliação da opinião dos participantes com perguntas a respeito da utilização do Ciclo de Vida Clássico (Apêndice I).
- **Descrição da Tarefa (Processo Proposto)**: documento utilizado para contextualizar e descrever a tarefa a ser realizada pelos participantes do experimento, utilizando o processo proposto (Apêndice D).
- **Material de apoio (Processo Proposto)**: roteiro de tarefas a serem realizadas pelos participantes no experimento, seguindo o processo proposto (Apêndice F).
- **Formulário de Coleta de Dados (Processo Proposto)**: formulário preenchido pelos participantes, indicando o horário de início e o horário de fim de cada tarefa realizada no experimento, utilizando o processo proposto (Apêndice H).
- **Formulário de Avaliação (Processo Proposto)**: documento usado para avaliação da opinião dos participantes com perguntas a respeito da utilização do processo proposto (Apêndice J).

Com o objetivo de evitar a interferência do tempo gasto no aprendizado do processo proposto e do processo baseado no ciclo de vida clássico, um treinamento com duração de duas semanas foi planejado e oferecido a todos os participantes como forma de aquecimento (*warm-up*), de modo que todos estivessem aptos a desenvolver as aplicações propostas no experimento.

A plataforma adotada para o desenvolvimento de ambas as aplicações teve Java como linguagem de implementação, o *framework* JSF para apoiar a implementação do padrão MVC, o *framework* Hibernate e JPA para auxiliar a persistência em banco

de dados MySQL[2] e o IDE Eclipse[3] como ambiente de desenvolvimento. Essa plataforma, bem como todo os recursos de software e hardware necessários para o experimento foram previamente configurados nos computadores que o experimento foi realizado.

Execução

No dia da execução do experimento, as ferramentas e os materiais necessários foram armazenados em um pacote que foi disponibilizado para os grupos. Este pacote conteve os seguintes elementos:

- O IDE Eclipse configurado com o *plug-in* da ferramenta CASE Papyrus UML[4].
- Arquivos de bibliotecas (Hibernate, JSF e Conector MySQL).
- Pasta contendo o servidor Apache Tomcat para execução da aplicação desenvolvida.
- Diagrama de classes da aplicação desenvolvida.
- Todos os documentos descritos na etapa de preparação do experimento.

Além disso, quando utilizaram o processo proposto, os grupos receberam um gerador previamente desenvolvido para o experimento. Nesse gerador, transformações M2C para gerar código Java a partir de diagramas de classe da UML e metaprogramas para geração de funcionalidades CRUD foram reutilizados pelos participantes durante o processo proposto.

Durante o experimento, os grupos desenvolveram as atividades e registraram o horário de inicio e fim de cada uma das tarefas realizadas na implementação da aplicação no *Formulário de Coleta de Dados*. O Quadro 4.1 apresenta uma compilação dos dados preenchidos pelos participantes, mostrando os tempos gastos pelos grupos nas atividades de implementação das aplicações e o Quadro 4.2 mostra os significados dos acrônimos das atividades executadas.

Conforme visto no Quadro 4.1, os dados foram divididos em dois blocos. No bloco superior estão os dados referentes ao processo baseado no ciclo de vida clássico e

[2]http://www.mysql.com/
[3]http://www.eclipse.org/
[4]Papyrus UML - http://www.eclipse.org/papyrus/

Quadro 4.1: Tempos gastos pelos grupos na realização do experimento.

		TProjeto	TClasses Entidade	TPersistência	TBeans Gerenciados	TConversores	TPáginas Web	Total
Ciclo de Vida Clássico	G1	0 h 14 min	0 h 32 min	0 h 7 min	0 h 25 min	0 h 6 min	0 h 44 min	2 h 8 min
	G2	0 h 6 min	0 h 38 min	0 h 3 min	0 h 13 min	0 h 4 min	0 h 59 min	2 h 3 min
	G3	0 h 7 min	0 h 59 min	0 h 3 min	0 h 8 min	0 h 3 min	0 h 37 min	1 h 57 min
	G4	0 h 8 min	1 h 8 min	0 h 8 min	0 h 14 min	0 h 6 min	0 h 15 min	1 h 59 min
	G5	0 h 7 min	0 h 49 min	0 h 8 min	0 h 15 min	0 h 6 min	0 h 40 min	2 h 5 min
	G6	0 h 8 min	0 h 41 min	0 h 13 min	0 h 25 min	0 h 7 min	0 h 33 min	2 h 7 min
	G7	0 h 13 min	0 h 33 min	0 h 7 min	0 h 32 min	0 h 8 min	0 h 22 min	1 h 55 min
	G8	0 h 5 min	0 h 30 min	0 h 10 min	0 h 10 min	0 h 2 min	1 h 2 min	1 h 59 min
	G9	0 h 8 min	1 h 7 min	0 h 9 min	0 h 8 min	0 h 6 min	0 h 29 min	2 h 7 min
	Média	**0 h 8 min**	**0 h 46 min**	**0 h 7 min**	**0 h 16 min**	**0 h 5 min**	**0 h 37 min**	**2 h 2 min**
Processo Proposto	G1	0 h 7 min	0 h 4 min	0 h 4 min				0 h 15 min
	G2	0 h 8 min	0 h 3 min	0 h 1 min				0 h 12 min
	G3	0 h 1 min	0 h 7 min	0 h 1 min				0 h 9 min
	G4	0 h 3 min	0 h 4 min	0 h 1 min				0 h 8 min
	G5	0 h 2 min	0 h 1 min	0 h 1 min				0 h 4 min
	G6	0 h 8 min	0 h 2 min	0 h 3 min				0 h 13 min
	G7	0 h 10 min	0 h 2 min	0 h 1 min				0 h 13 min
	G8	0 h 6 min	0 h 2 min	0 h 1 min				0 h 9 min
	G9	0 h 9 min	0 h 6 min	0 h 3 min				0 h 18 min
	Média	**0 h 6 min**	**0 h 3 min**	**0 h 1 min**				**0 h 11 min**

Quadro 4.2: Significado dos acrônimos das atividades do experimento.

Acrônimo	Descrição
Tprojeto	Tempo gasto na criação do projeto Web no IDE Eclipse
TClassesEntidade	Tempo gasto na implementação das classes de entidade usando JPA
Tpersistência	Tempo gasto na implementação dos beans de sessão para acesso do banco de dados
TBeansGerenciados	Tempo gasto na implementação dos beans gerencidos
Tconversores	Tempo gasto na implementação de conversores
TPáginasWeb	Tempo gasto na implementação de páginas Web usando JSF

no bloco inferior estão os dados referentes ao processo proposto. A coluna rotulada com “Total”, representa o tempo total de implementação das aplicações. As linhas rotuladas com “Média”, mostram os tempos médios gasto na implementação de cada atividade e o tempo médio total de implementação das aplicações.

Validação dos Dados

De modo geral, constatou-se que os grupos desempenharam satisfatoriamente as tarefas que lhes foram propostas e os dados coletados apresentaram-se dentro

dos limites esperados. Dessa forma, os tratamentos foram executados corretamente e dentro de acordo com o planejamento realizado. Consequentemente, os dados obtidos foram considerados válidos para realizar a avaliação proposta.

4.3.4 Análise e Interpretação dos Resultados

Analisando os dados coletados, nota-se que, quando seguiram o processo baseado no ciclo de vida clássico para realizar a implementação da aplicação, os grupos gastaram, em média, 2 horas e 2 minutos para finalizar as atividades propostas, enquanto que seguindo o processo proposto foram gastos, em média, apenas 11 minutos (redução de 90,98%). Portanto, conforme mostra o Quadro 4.1, observa-se claramente que os grupos, utilizando o processo proposto, completaram a implementação da aplicação do experimento de forma mais rápida comparando-se ao ciclo de vida clássico. Essa diferença se justifica pela reutilização de mecanismos de transformações M2C e metaprogramas que realizaram a geração de grande parte do código da aplicação.

É importante ressaltar que com o aumento do número de classes e atributos em cada classe presente nos modelos, a quantidade de tempo e esforços necessários para finalizar as atividades de implementação, seguindo o processo baseado no ciclo de vida clássico, tende a ser maior proporcionalmente. Assim, embora muitas vezes a geração de código não seja total, no caso de operações de CRUD e afins, a utilização de uma abordagem baseada em geração de código a partir de modelos pode proporcionar um ganho de produtividade significativo que justifica o esforço e o tempo gasto na construção dos mecanismos de geração (geradores), que a princípio é realizada uma única vez. Portanto, qualquer outra aplicação que utilize essa mesma plataforma e tecnologias poderá se beneficiar dessa geração de código.

Estatística Descritiva

A estatística descritiva lida com a apresentação e processamento numérico de um conjunto de dados (WOHLIN et al., 2000). Depois que o experimento é realizado, os dados coletados devem ser analisados, agrupados e apresentados de forma gráfica para para descrever resultados sob diferentes perspectivas.

A forma utilizada para apresentar a dispersão dos tempos gastos pelos grupos participantes do experimento foi através do uso de gráficos de caixa (*boxplot*). A Fi-

gura 4.4 apresenta o gráfico *boxplot* referente à dispersão dos tempos totais gastos pelos grupos durante o experimento. Conforme observado na Figura 4.4, evidencia-se que os grupos que seguiram o processo proposto (com base em geração de código a partir de modelos) foram claramente mais rápidos do que os grupos que seguiram o ciclo de vida clássico.

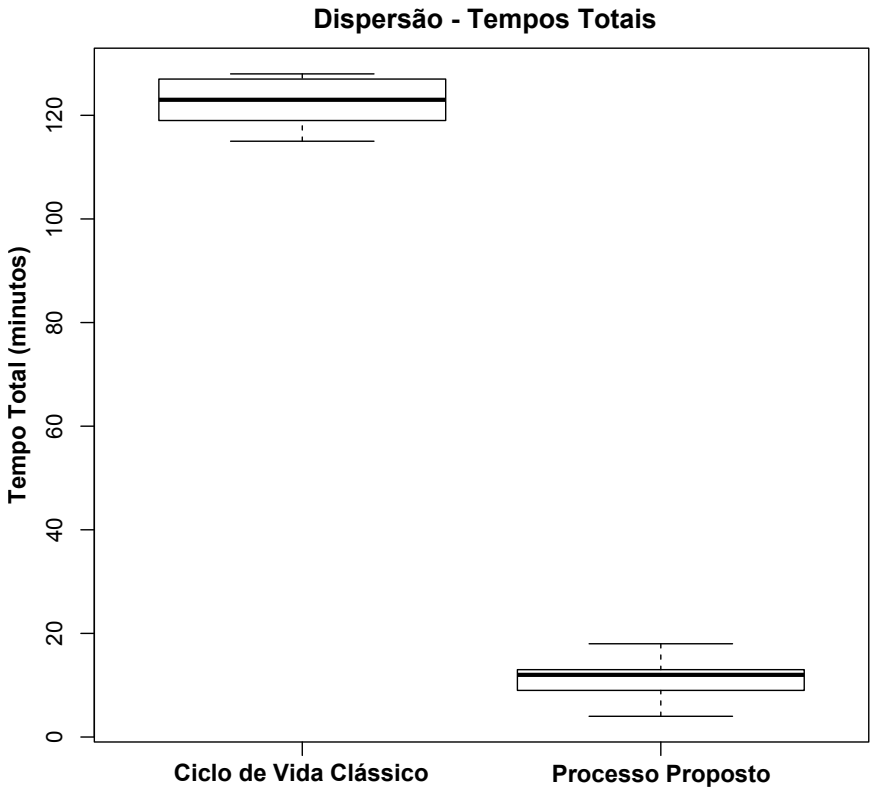


Figura 4.4: Dispersão dos tempos totais dos grupos participantes do experimento.

Por meio do teste não paramétrico *Shapiro-Wilk* (SHAPIRO; WILK, 1965) foi realizada a verificação da normalidade do conjunto de dados. O resultado do teste confirmou que os conjuntos de amostras utilizados (para ambos processos) consistiam de conjuntos normalmente distribuídos, conforme exibido na Figura 4.5.

Redução do Conjunto de Dados

Antes de realizar a aplicação de métodos estatísticos, é necessário verificar a qualidade dos dados de entrada (WOHLIN et al., 2000). A representação e a validade dos dados impactam diretamente nas conclusões obtidas a partir dos resultados, de modo que se a análise estatísticas não for feita sobre dados corretos, conclusões erradas podem ser obtidas. Falhas no conjunto de dados de entradas podem ser ocasionadas devido a erros sistemáticos ou à presença de *outliers*, que são dados

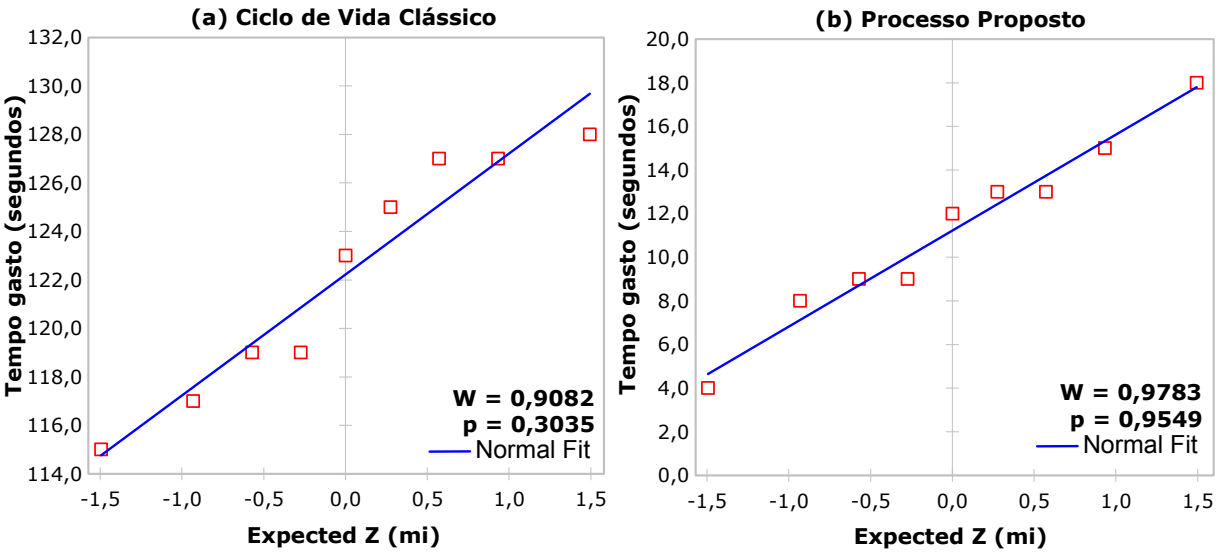


Figura 4.5: Normalidades dos conjuntos de amostras.

muito maiores ou muito menores dos que são esperados em comparação com os demais dados do conjunto.

A partir da análise cuidadosa dos formulários preenchidos pelos participantes do experimento, notou-se que não foram detectados erros sistemáticos (e.g erros de transcrição por parte dos participantes) ou a presença de qualquer *outlier* no conjunto de dados de entrada. Sendo assim, não houve necessidade de realizar qualquer redução do conjunto de dados e, portanto, os dados foram considerados válidos para a aplicação de métodos estatísticos de análise dos dados.

Teste das Hipóteses

De modo geral, analisando os dados obtidos pelos grupos participantes do experimento, notou-se que os grupos que utilizaram o processo proposto foram mais eficientes que os grupos que seguiram o processo com base no ciclo de vida clássico. Visando comprovar e quantificar o ganho obtido com o processo proposto, a partir dos dados coletados, aplicou-se o *t-test pareado* (MONTGOMERY, 2008) (Equação 4.1).

$$t_0 = \frac{\bar{d}}{S_d/(\sqrt{n})} \tag{4.1}$$

em que:

n = Número de amostras pareadas.

$\bar{d}$ = Diferença das médias de cada conjunto de amostras.

S_d = Desvio padrão das diferenças das amostras (Equação 4.2).

$$S_d = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \bar{d})^2}{n-1}} \tag{4.2}$$

em que:

d_i = Diferença entre cada par de amostra.

Com base nas amostras de $\tau_{Clássico} = \{128, 123, 117, 119, 125, 127, 115, 119, 127\}$ e $\tau_{Processo} = \{15, 12, 9, 8, 4, 13, 13, 9, 18\}$ (Quadro 4.1), tem-se que $n = 9$ e $d = \{113, 111, 108, 111, 121, 114, 102, 110, 109\}$. Os valores médios μ de cada conjunto de amostras são $\mu\tau_{Clássico} = 122,222$ e $\mu\tau_{Processo} = 11,222$. Pode-se, então, obter $\bar{d} = 122,222 - 11,222 = 111$. Assim, $S_d = 5,099$ e $t_0 = 65,30667$.

O número de graus de liberdade é $gl = n - 1 = 8$. Tomou-se $\alpha = 6,8 \cdot 10^{-12}$. Na tabela padrão de distribuição de probabilidade estatística *t* de *Student*, verifica-se que $t_{3,4 \cdot 10^{-12},8} = 65,2154$. Desde que $|t_0| > t_{3,4 \cdot 10^{-12},8}$ **foi possível rejeitar a hipótese nula H_0 com $6,8 \cdot 10^{-12}$% de significância**.

Para realizar os cálculos apresentados neste trabalho foi utilizado o software R[5], um ambiente de software para geração de gráficos e análises estatísticas. Além disso, foi utilizado o software RKWard[6], usado como interface gráfica para o R.

Conclusões do Experimento

Uma vez realizado o teste das hipóteses e tendo sido constatado ser possível rejeitar a hipótese nula (H_0), é possível tirar algumas conclusões sobre o experimento no que diz respeito da influência das variáveis independentes sobre a variável dependente, considerando a validade do experimento e tendo sido devidamente tratadas as ameaças à validade (Seção 4.3.5).

A partir da rejeição da hipótese nula, é possível afirmar que as diferenças nos tempos de desenvolvimento, evidenciadas nas amostras dos grupos quando seguiram o

[5] R - http://www.r-project.org/

[6] RKWard - http://rkward.sourceforge.net/

ciclo de vida clássico e quando utilizaram o processo proposto, possuem significância estatística. Isso indica que a diferença apresentada no tempo de desenvolvimento gasto pelas equipes foi ocasionada devido à variação do *processo de desenvolvimento* utilizado e não por acaso, ou em decorrência de erros na coleta das amostras.

Como se observa no Quadro 4.1, os grupos de participantes quando seguiram o processo proposto apresentaram uma média do tempo total menor que a média dos grupos que utilizaram o ciclo de vida clássico ($\mu\tau_{Processo} < \mu\tau_{Clássico}$). Dessa forma, existe evidência de que a hipótese H_1 possa ser validada em detrimento da hipótese H_2, de modo que há indícios para afirmar que equipes que utilizam o processo proposto, geralmente, gastam menos tempo no desenvolvimento de software. Este resultado se encontra dentro das expectativas do experimento, em que se esperava que os geradores de código automatizariam grande partes das tarefas de desenvolvimento e essa expectativa foi constatada na prática.

Além disso, é importante constatar que as conclusões sobre os resultados obtidos nesse experimento se restringem ao escopo de aplicações CRUD implementadas por desenvolvedores de software no ambiente universitário, uma vez que o experimento foi realizado *in-vitro* e sob condições controladas. Nos dados coletados, apenas o tempo a respeito das atividades de implementação foi considerado, de modo que o tempo gasto em outras etapas de desenvolvimento de software (por exemplo, modelagem e teste) não foi coberto pelo experimento proposto. Para expandir o resultado obtido para um contexto mais amplo, torna-se necessária a realização de novos experimentos com um número maior de participantes e a execução em ambientes *in-vivo*, inclusive realizando comparação do processo proposto com outras abordagens para o de desenvolvimento de software, além do ciclo de vida clássico, abordado no estudo.

A reaplicação do experimento em um ambiente industrial é um importante passo para incrementar a validação do processo proposto, uma vez que irá colaborar para estender a amplitude dos resultados a novos contextos, onde o processo proposto pode ser comparados com outras abordagens de desenvolvimento (por exemplo, linhas de produtos de software e métodos ágeis) também considerando fatores ausentes em um ambiente acadêmico. Além disso, outros efeitos relacionados com o desenvolvimento de software, tais como a eficácia em relação da ocorrência de falhas durante o desenvolvimento, também pode ser estudada. Neste caso, as avaliações empíricas com usuários e testes de inspeção devem ser planejados a fim de coletar métricas relevantes para avaliar o grau em que as aplicações desenvolvidas atingem as metas

de eficácia, eficiência e satisfação subjetiva do ponto de vista dos usuários finais.

O pacote, contendo ferramentas e materiais utilizados ao longo do experimento, encontra-se disponível em *www.dc.ufscar.br/∼paulo.papotti/EXPERIMENTO.zip* e pode ser aproveitado por pesquisadores e profissionais para auxiliar na realização de novos experimentos que estejam relacionados com o estudo proposto. Dessa forma, espera-se que novos resultados possam ser alcançados e, dessa forma, contribuir ainda mais com o estado da arte na área de geração de código.

Feedback dos Participantes

Com o objetivo de coletar a opinião dos participantes sobre o as abordagens consideradas no experimento, após a execução, os alunos receberam dois formulários de avaliação, sendo um formulário com perguntas contendo espaço para os participantes reportassem sua percepção relacionada ao ciclo de vida clássico (Apêndice I) e outro formulário de avaliação com perguntas a respeito do processo proposto (Apêndice J).

Após os participantes responderem ambos os questionários, as respostas foram analisadas e pôde-se notar alguns resultados interessantes.

Na avaliação sobre o processo do ciclo de vida clássico, os participantes foram perguntados se utilizaram mecanismos ou ferramentas para realizar algum tipo de geração de código e, em caso afirmativo, se o mecanismo ou ferramenta atendeu às necessidades deles. Como mostrado na Figura 4.6, a maioria dos participantes (55%) indicou que fez uso de algum mecanismo de geração de código, atendendo suas necessidades de forma parcial. Uma parcela menor dos participantes (31%) admitiram ter realizado geração de código e ter atendido totalmente as suas necessidades. 10% utilizaram algum mecanismo, mas não foram atendidos em suas necessidades e, por fim, 4% não utilizaram quaisquer mecanismos ou ferramentas com o objetivo de realizar geração de código. Esse resultado era esperado, uma vez que os mecanismos utilizados pelos participantes foram geradores de métodos de acesso (*getters* e *setters*) às propriedades das classes, construtores e similares. Tais geradores, embora úteis, são bastante simples e limitados.

Em seguida, os participantes foram perguntados se enfrentaram algum tipo de dificuldade durante o desenvolvimento da tarefa proposta. Os resultados, como mostrado na Figura 4.7, revelaram que 55% dos participantes afirmaram ter tido dificuldades, 32% julgaram ter tido dificuldades de forma parcial e apenas 14% não apresentaram

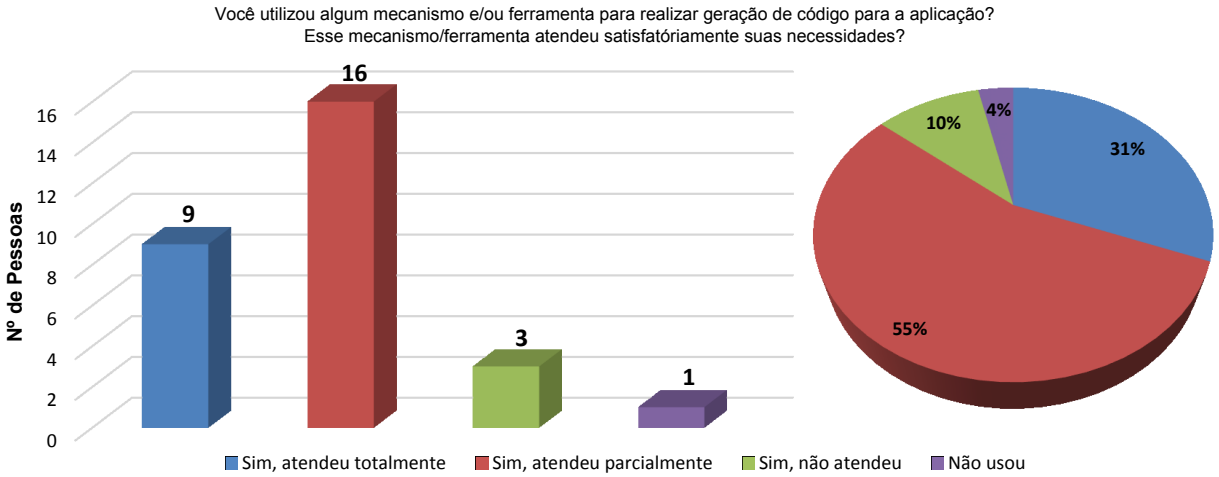


Figura 4.6: Estatística do uso de mecanismos ou ferramentas de geração de código durante o processo clássico.

dificuldades. A partir desse cenário, notou-se que a grande maioria encontrou dificuldades ao longo do desenvolvimento da tarefa proposta seguindo o processo com base no ciclo de vida clássico. Dentre as dificuldades apontadas pelos participantes quando seguiram o ciclo de vida clássico, destacam-se: (1) muito esforço gasto na codificação; (2) resolução de problemas relacionados à linguagem; e (3) erros cometidos por falta de atenção do programador.

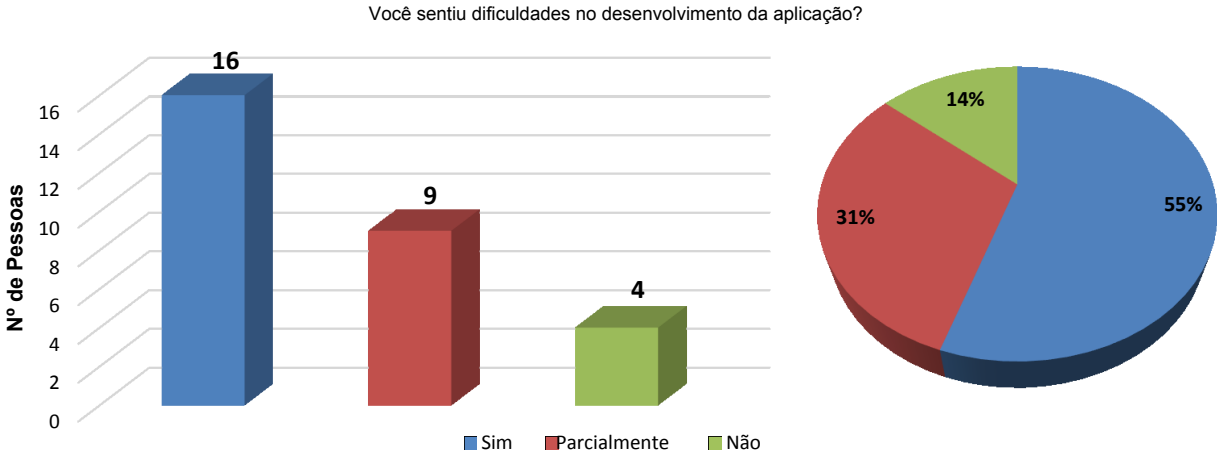


Figura 4.7: Estatística sobre as dificuldades enfrentadas durante o processo clássico.

Outra questão presente no questionário envolveu se o participante considera que um processo, com base em modelos e geração de código poderia contribuir para facilitar o desenvolvimento de software. Nesse caso, quase que de forma unânime (97%), os participantes afirmaram que um processo com tais características poderia tornar mais fácil algumas tarefas durante o desenvolvimento. Por outro lado, apenas 3% das pessoas respondeu que poderia ajudar de forma parcial e nenhum participante

afirmou que não ajudaria. Os dados estatísticos, referentes a essa questão podem ser visualizados na Figura 4.8.

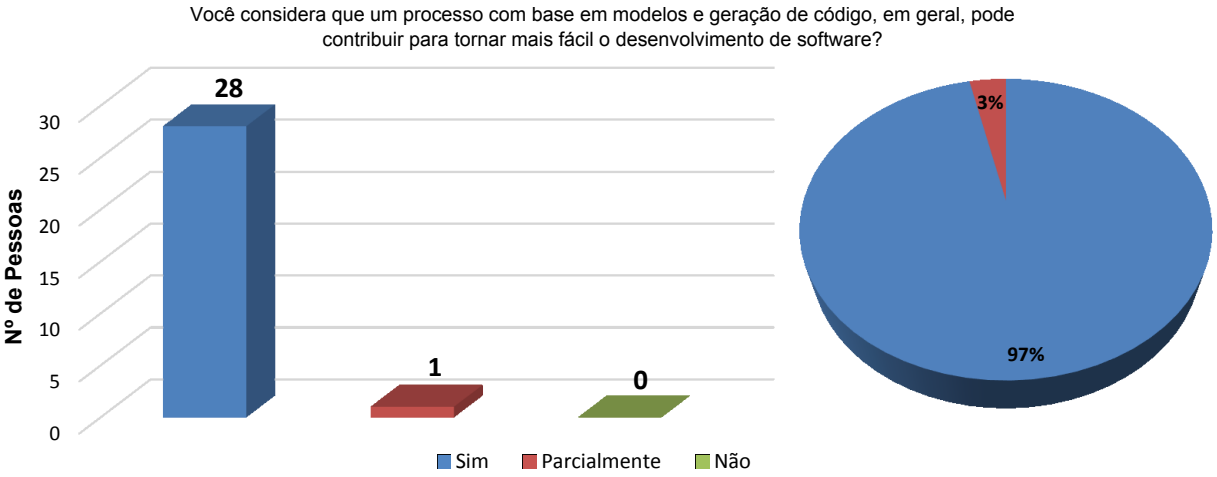


Figura 4.8: Estatística sobre a possível utilização de um processo com base em modelos e geracão de código.

Ainda relacionado ao questionário de avaliação do ciclo de vida clássico, outra questão envolveu mensurar se, na opinião dos participantes, era viável realizar a construção de aplicações sem a geração de funcionalidades tais como CRUD, considerando o esforço necessário. Nesse caso, como mostra a Figura 4.9, verificou-se que 21% dos entrevistados afirmaram que é viável construir aplicações sem nenhum tipo de geração de funcionalidades. Por outro lado, 34% julgaram que não é viável, considerando o esforço, a não utilização de qualquer tipo de geração de funcionalidades. Por fim, a maioria representada por 45% considerou que é parcialmente viável utilizar a geração de funcionalidades. Dessa forma, é possível perceber que os participantes acreditam que apenas em alguns casos ou alguns tipos de funcionalidades são viáveis de serem geradas automaticamente.

Terminada a análise da opinião dos participantes a partir dos dados coletados sobre o processo com base no ciclo de vida clássico, realizou-se a mesma análise com os dados coletados a parir dos questionários respondidos sobre o processo proposto.

A primeira questão do questionário do processo proposto estava relacionada em saber se a utilização do processo auxiliou de alguma forma no desenvolvimento das tarefas realizadas. Nesse caso, todos os participantes afirmaram que o processo proposto ajudou na execução das tarefas, mostrando que houve benefícios com a utilização do processo.

Em seguida, os participantes foram perguntados se enfrentaram algum tipo de difi-

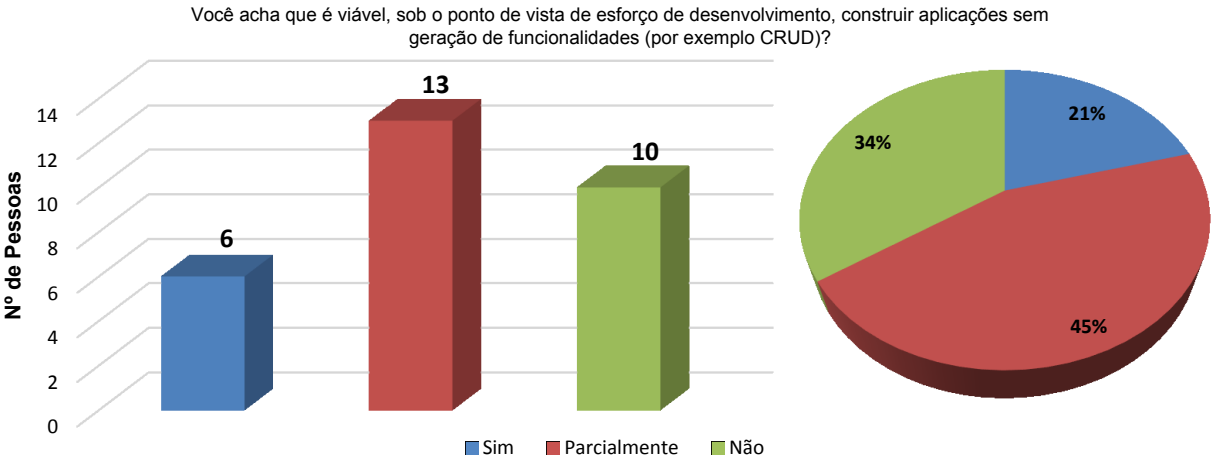


Figura 4.9: Estatística sobre a construção de aplicação sem a geração de funcionalidades.

culdade no decorrer do desenvolvimento da tarefa, seguindo o processo proposto. Os resultados, como mostra a Figura 4.10, mostraram que a grande maioria (73%) dos participantes realizaram todas as tarefas sem qualquer dificuldade. Adicionalmente, 10% enfrentaram dificuldades de forma parcial e 17% afirmaram ter enfrentado dificuldades de forma total.

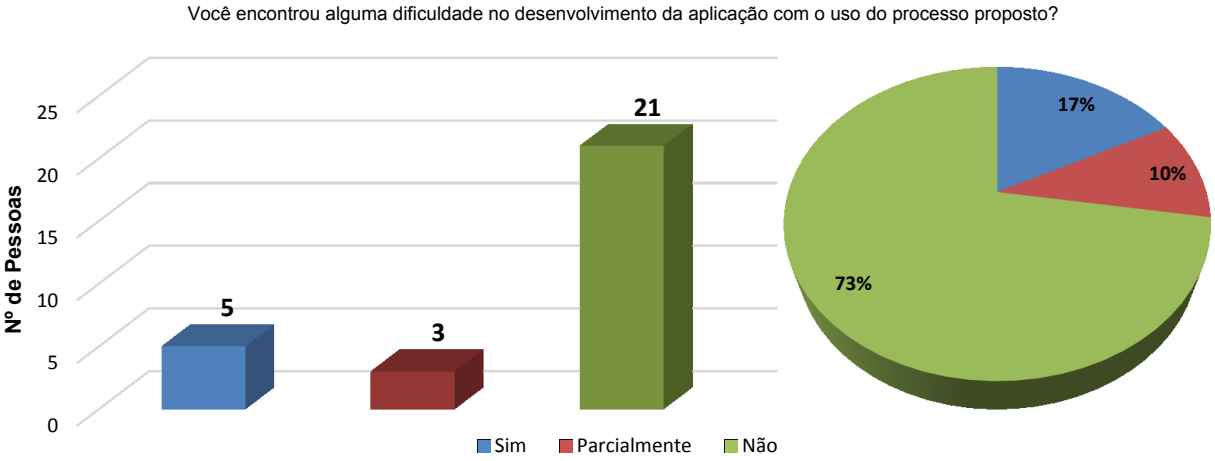


Figura 4.10: Estatística sobre a dificuldades enfrentadas durante o processo proposto.

Conforme visto na Figura 4.11, é possível verificar que houve diminuição da dificuldade apresentada pelos participantes quando utilizaram mecanismos de geração de código. Seguindo o ciclo de vida clássico, 86% dos participantes apresentaram algum tipo de dificuldade, sendo total ou parcial (Figura 4.7). Utilizando o processo proposto, esse índice caiu para 27%. Dessa forma, acredita-se que os mecanismos de geração de código foram fundamentais para tornar mais fácil a tarefa dos participantes, proporcionando um aumento de 59% na porcentagem de participantes que desenvolveram

a tarefa proposta sem qualquer dificuldade. Além disso, dentre as dificuldades enfrentadas pelos participantes quando utilizaram o processo proposto, destaca-se: (1) falta de prática com uso de geradores; e (2) a baixa integração entre o gerador e o IDE de desenvolvimento.

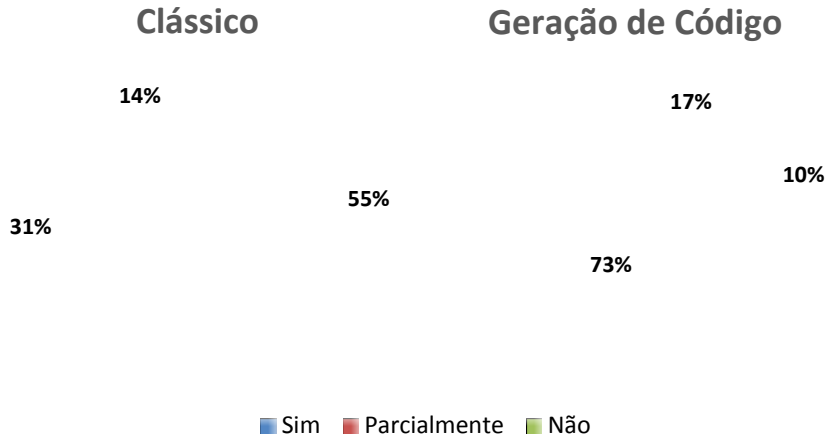


Figura 4.11: Comparativo sobre a dificuldade enfrentada pelos participantes seguindo tanto o ciclo de vida clássico quanto o processo proposto.

Outro ponto avaliado no questionário do processo proposto visou descobrir se os participantes consideravam que o uso dos geradores (transformações e metaprogramas) contribuíram para o desenvolvimento da aplicação do experimento. Confirmando as expectativas do experimento, a Figura 4.12 mostra que 97% dos participantes afirmaram positivamente, ratificando os geradores como tendo papel fundamental no auxílio para a realização das tarefas. 3% julgaram que a contribuição dos geradores foi de forma parcial e nenhum participante afirmou que os geradores não contribuíram de nenhuma forma. Esse resultado indica o reconhecimento dos participantes em relação à utilidade dos geradores de código durante o desenvolvimento das tarefas, proporcionando economia de tempo e esforço na implementação.

A possível utilização do processo proposto em outros tipos de sistemas não abordados no experimento (Arquitetura Mobile, Serviços, etc) também foi um ponto avaliado no questionário. Nesse caso, como ilustrado na Figura 4.13, 86% acreditam que o processo proposto é viável de ser utilizado para auxiliar o desenvolvimento de outros tipos de sistemas, 14% responderam que o processo pode ser parcialmente usado e nenhum participante afirmou que o processo não possa ser utilizado em outros casos. Esse resultado também possui grande relevância para a pesquisa, uma vez que toda abordagem com base em geração de código pode apresentar dificuldades para se adaptar às características de diferentes tipos de sistemas. Esse resultado, no entanto,

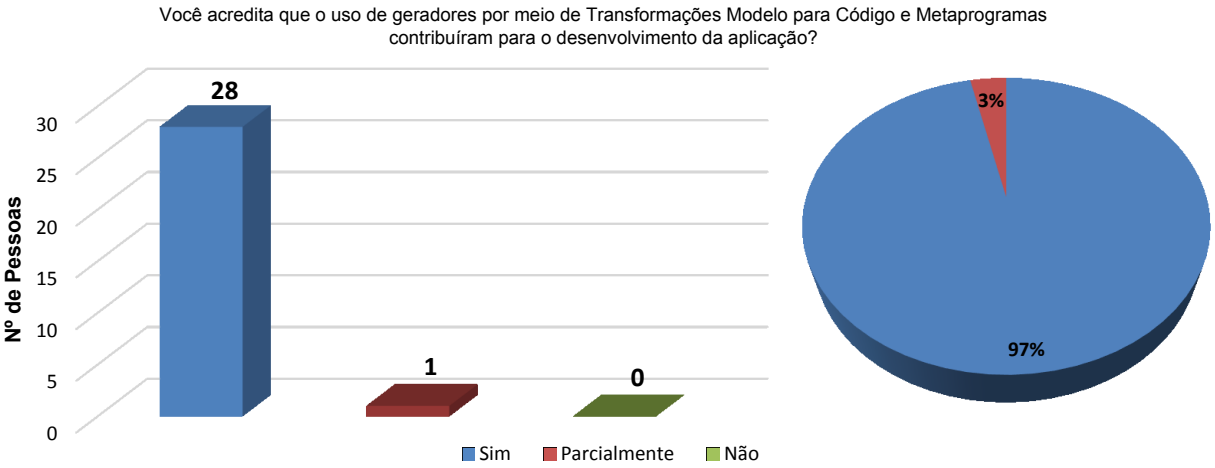


Figura 4.12: Estatística sobre a avaliação do uso de geradores durante o processo proposto.

mostra o otimismo dos usuários nesse sentido.

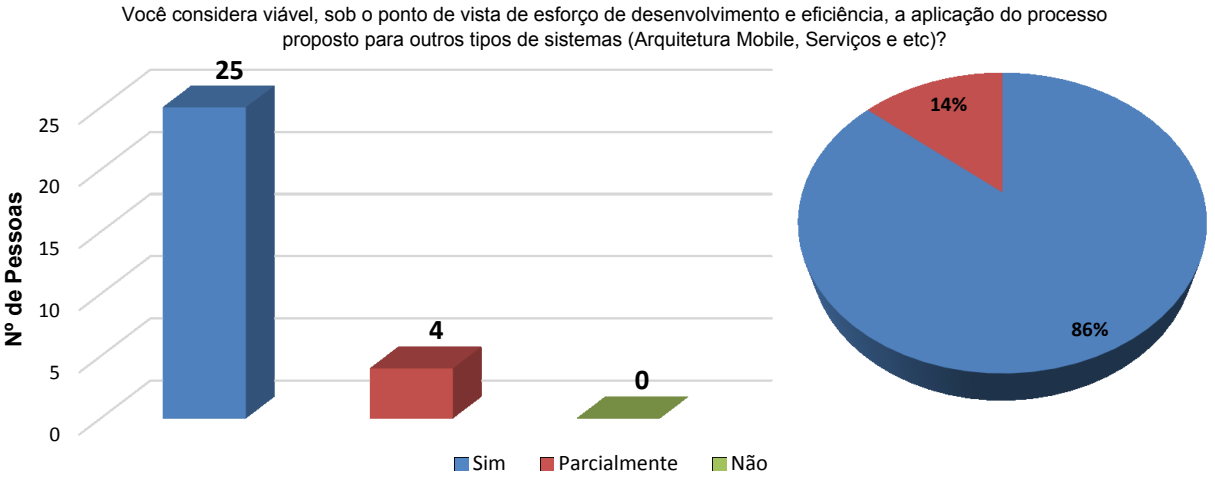


Figura 4.13: Estatística da avaliação do uso do processo proposto em outros tipos de sistemas.

Uma vez que todos os participantes afirmaram que o processo proposto auxiliou no desenvolvimento da tarefa proposta, um das questões do formulário de avaliação visava identificar quais as vantagens da sua utilização em comparação com ao desenvolvimento seguindo o ciclo de vida clássico. A Figura 4.14 exibe os resultados obtidos, de modo que 89,66% dos participantes destacaram a geração de parte do código da aplicação e o uso de padrão de software como sendo as maiores vantagens do processo proposto. Além disso, outras benefícios apontados incluem o maior foco na modelagem (68,97%), facilidade nas manutenções futuras (55,17%) e outras (6,90%).

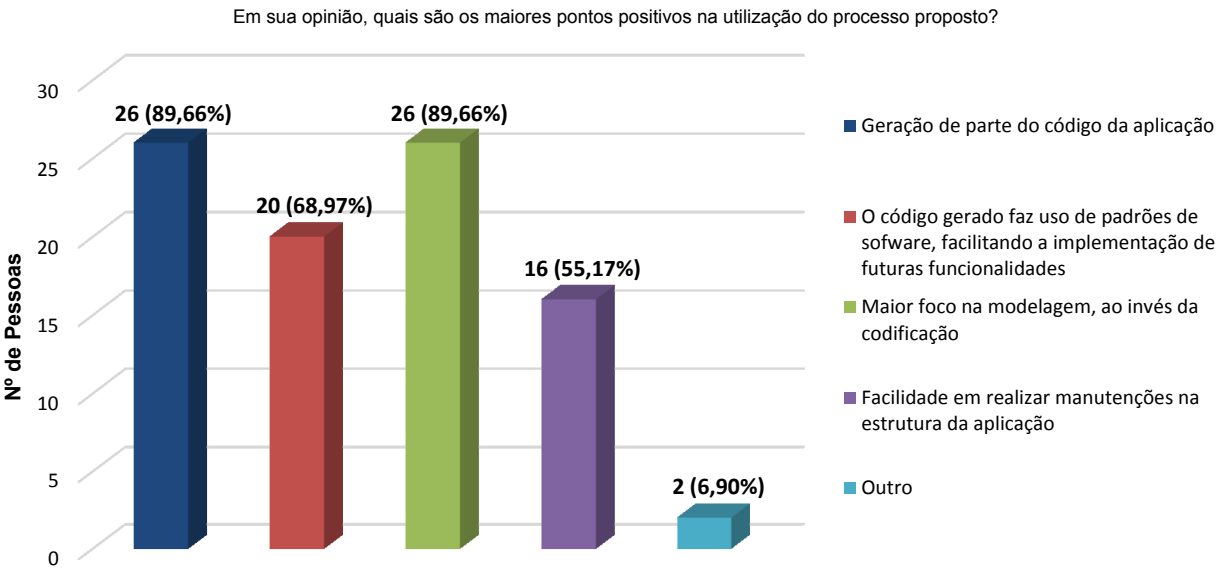


Figura 4.14: Estatísticas dos pontos positivos da utilização do processo proposto.

Por fim, outro objetivo do questionário foi mensurar o grau de satisfação dos participantes com a utilização do processo proposto comparado ao ciclo de vida clássico. Em uma escala de 0 a 10, em que 0 é nada satisfeito e 10 é totalmente satisfeito, os participantes avaliaram o grau de satisfação obtido com o processo proposto. Como mostra a Figura 4.15, o grau apresentado pelos participantes ficou entre os níveis 7 e 10, sendo que 3% tiveram grau de satisfação 7, 21% com grau 8, 38% com grau 9 e 38% com grau 10. Além disso, a média do grau de satisfação dos participantes manteve-se em torno de 9.1, demonstrando que o processo proposto teve a aceitação dos participantes de modo geral.

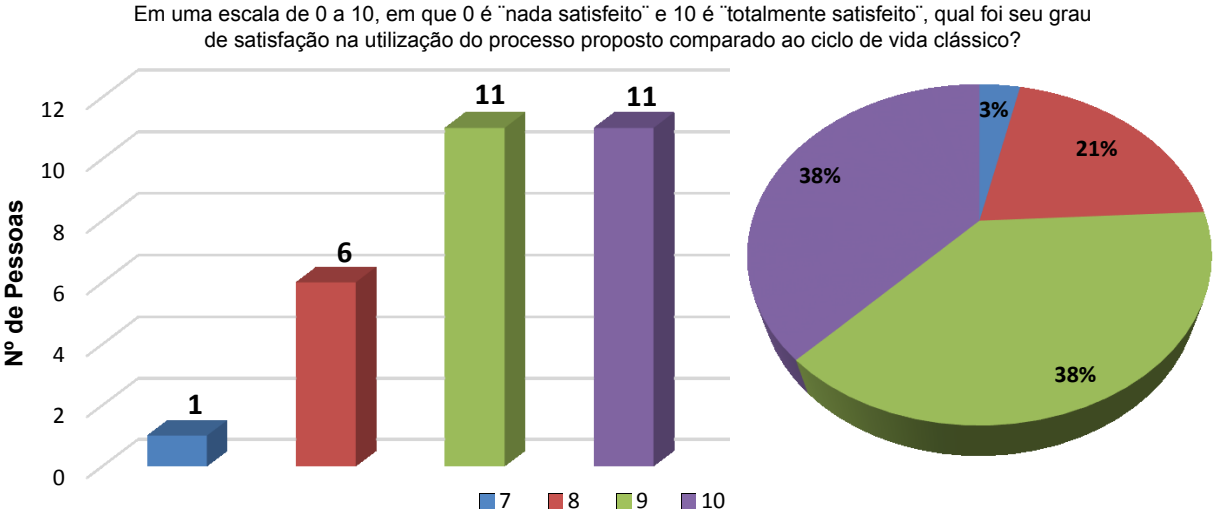


Figura 4.15: Estatística do grau de satisfação dos participantes com o processo proposto.

4.3.5 Ameaças à Validade

Uma questão muito importante em um experimento é determinar a validade dos resultados obtidos. Por diferentes razões, um estudo pode ter seus resultados invalidados dependendo da forma em que foi conduzido. Portanto, é fundamental considerar as condições para a validade desde as etapas inicias do experimento. Dentre os diferentes tipos de validade que podem colocar em risco os resultados de um experimento, têm-se: Validade de Conclusão, Validade Interna, Validade de Construção e Validade Externa (WOHLIN et al., 2000). Todas essas validades foram consideradas no experimento proposto.

Validade de Conclusão

Diferentes cuidados foram tomados visando assegurar a validade de conclusão do experimento. Usou-se um teste estatístico paramétrico (t-test pareado), sendo adequado para avaliar o fator "*processo de desenvolvimento*" com os tratamentos "*processo proposto*" e "*ciclo de vida clássico*".

Para realizar a aplicação do t-test pareado, foi comprovada previamente a normalidade dos dados por meio do teste de normalidade de *Shapiro-Wilk* (Figura 4.5) com nível de significância de pelo menos 5%. No entanto, para reforçar e assegurar a validade dos resultados, foi aplicado o teste não-paramétrico de *Wilcoxon*, sendo um teste alternativo ao t-test pareado que não requer que os dados coletados estejam normalmente distribuídos. Nesse caso, os resultados obtidos foram na mesma direção dos resultados apresentados com o t-test pareado, de modo a rejeitar a hipótese nula.

Além disso, visando aumentar a validade de conclusão, os dados coletados pelos participantes (hora e minutos) não foram dependentes de julgamento humano e, portanto, possuem maior confiabilidade, embora tenham sido coletados pelos próprios participantes.

Por fim, a heterogeneidade do nível de experiência dos participantes foi tratada por meio do agrupamento de participantes em blocos (grupos) de nível de experiência médio similar. Dessa forma, evitou-se valores demasiadamente discrepantes nos dados das amostras coletadas, ocasionados pela diferença de conhecimento entre os participantes.

Validade Interna

O experimento foi realizado por estudantes cursando graduação na área de Computação em estágio final (terceiro e quarto anos letivos), que geralmente estão habituados com desenvolvimento de software. Dessa forma, assumiu-se que esses estudantes são representativos para a população de desenvolvedores de software.

Além disso, um questionário de caracterização dos participantes do experimento foi utilizado para captar o perfil e o nível médio de experiência dos participantes com o intuito de dividi-los em grupos homogêneos de nível médio de experiência similares. Dessa forma, buscou-se evitar a influência de fatores que não são de interesse do estudo.

Validade de Construção

O objetivo do experimento proposto foi comparar dois processos de desenvolvimento de software diferentes (processo proposto e ciclo de vida clássico) e seu impacto no tempo gasto no desenvolvimento pela equipe. Dessa forma, dados de tempo dos grupos foram coletados durante o desenvolvimento de uma aplicação exemplo usando ambos processos, sendo esses os dados necessários para realizar tal comparação.

Visando evitar a interferência no comportamento dos participantes, as métricas e cálculos que seriam utilizados sobre os dados coletados não foram divulgados aos participantes, de modo que os mesmos ficassem concentrados em desenvolver o experimento proposto da forma mais espontânea possível, ao invés de buscar se empenhar mais ou menos do que de costume para obter resultados que favorecessem, ou prejudicassem o experimento.

Validade Externa

O experimento proposto foi conduzido em um laboratório informatizado, contendo os equipamentos necessários para a realização das tarefas pelos grupos participantes, incluindo ferramentas e tecnologias utilizadas no desenvolvimento de software em ambientes industriais, tais como a linguagem de programação Java e IDE Eclipse. Em relação às questões temporais, o experimento foi completamente executado em um período de aproximadamente 3 horas e, dessa forma, evitou-se que os resultados

pudessem ser afetados em decorrência do cansaço excessivo ou tédio por parte dos participantes do experimento.

4.4 Considerações Finais

Este capítulo apresentou os resultados de uma análise quantitativa a partir da realização de um experimento para avaliar a utilização de mecanismos de geração de código a partir de modelos no desenvolvimento de sistemas. A população de participantes do experimento foi composta por alunos de graduação dos últimos anos dos cursos de computação da UFSCar que desenvolveram uma aplicação com base no sistema ProgradWeb, seguindo tanto uma abordagem de desenvolvimento baseado no ciclo de vida clássico, quanto seguindo o processo proposto. Foram coletados dados sobre o tempo gasto no desenvolvimento das tarefas propostas em ambos os casos. Os resultados favoráveis, obtidos com a avaliação do trabalho, estão de acordo com as expectativas iniciais da concepção do processo proposto, incluindo a redução de esforços e aumento na produtividade no desenvolvimento de software.

Além disso, os resultados encontrados reafirmam os benefícios proporcionados por meio do uso das abordagens de desenvolvimento de software dirigido por modelos. Dentre outros, o aumento de produtividade e a redução de dificuldades durante o desenvolvimento foram os pontos mais explorados no experimento realizado. Embora o estudo tenha sido realizado em ambiente universitário, os resultados obtidos possuem validade e, respeitando as condições adotadas, podem ser utilizados em outros projetos de pesquisa que visem avançar, ainda mais, no que diz respeito ao estado da arte nas áreas de pesquisa relacionadas.

Capítulo 5

ESTUDO DE CASO - REENGENHARIA

5.1 Considerações Iniciais

Um estudo de caso é realizado com o objetivo de investigar um objeto ou fenômeno em um específico espaço de tempo (WOHLIN et al., 2000). Neste capítulo, são apresentados o detalhamento e os resultados de um estudo de caso realizado para testar a viabilidade da utilização do processo proposto no contexto apoio à reengenharia de sistemas legados, usando a adaptação do processo (Seção 3.3).

A partir das impressões e resultados obtidos, é possível ter uma noção melhor do potencial do processo proposto no suporte à reengenharia de sistemas considerando tempos e esforços que podem ser poupados na reconstrução de um sistema real.

Este Capítulo está organizado da seguinte forma: a Seção 5.1 apresenta uma introdução do estudo de caso realizado. Em seguida, a Seção 5.2 detalha a avaliação realizada e mostra os resultados obtidos. Por fim, neste capítulo são apresentadas algumas considerações finais na Seção 5.3.

5.2 Estudo de Caso

Com o objetivo de analisar a aplicabilidade do processo proposto no apoio à reengenharia de sistemas legados, uma avaliação foi desenvolvida por meio de sua aplicação na reengenharia de um sistema real no domínio acadêmico. Essa avaliação teve como base o sistema PrograduWeb[1] da Universidade Federal de São Carlos (UFSCar). O PrograduWeb, é um grande sistema Web que tem sido usado desde meados de 1998

[1] PrograduWeb - https://prograduweb.ufscar.br/prograduweb/

e sua utilização está relacionada à gestão acadêmica das informações a respeito dos cursos de graduação da UFSCar, incluindo dados sobre professores, alunos, cursos e emissões de documentos oficiais.

Ao longo dos anos, o PrograduWeb tem sofrido sucessivas manutenções visando melhorar o uso do sistema e adicionar novas funcionalidades para se adequar às mudanças que a universidade tem passado. No entanto, atualmente, o sistema necessita de grandes alterações estruturais, como o integração com outros sistemas da universidade. Além disso, após esses anos, sua manutenção tornou-se onerosa em virtude de algumas tecnologias utilizadas estarem obsoletas.

A arquitetura original do sistema, ilustrada na Figura 5.1, foi concebida em uma única camada (contendo interfaces, controles e regras de negócio) e sem o uso de padrões estruturais. Essa arquitetura foi desenvolvida utilizando a tecnologia *Java Servlets* (HUNTER; CRAWFORD, 2001) e páginas Web na linguagem *Java Server Pages (JSP)* (BERGSTEN, 2003). O sistema é executado em um servidor configurado com o Apache Tomcat[2] e realiza conexão, utilizando a API *Java Database Connectivity (JDBC)*, com um banco de dados PostgreSQL[3] contendo 155 tabelas.

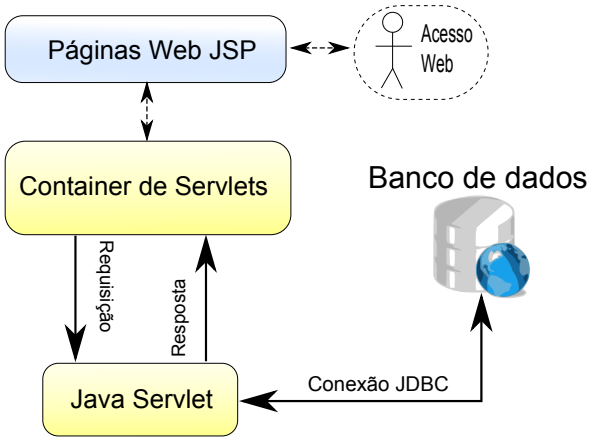


Figura 5.1: Arquitetura original do PrograduWeb.

A seguir, apresenta-se a execução das etapas da EA do processo proposto para apoiar a reengenharia do PrograduWeb: Engenharia Reversa (ER) e Engenharia Avante (EAv).

[2]Apache Tomcat - http://tomcat.apache.org/
[3]PostgreSQL - http://www.postgresql.org.br/

5.2.1 Execução da ER

Inicialmente, partiu-se do banco de dados do sistema para obter um modelo OO das classes de entidade. Um modelo de Entidade-Relacionamento (ER) obtido a partir das estrutura física do banco é apresentado parcialmente na Figura 5.2.

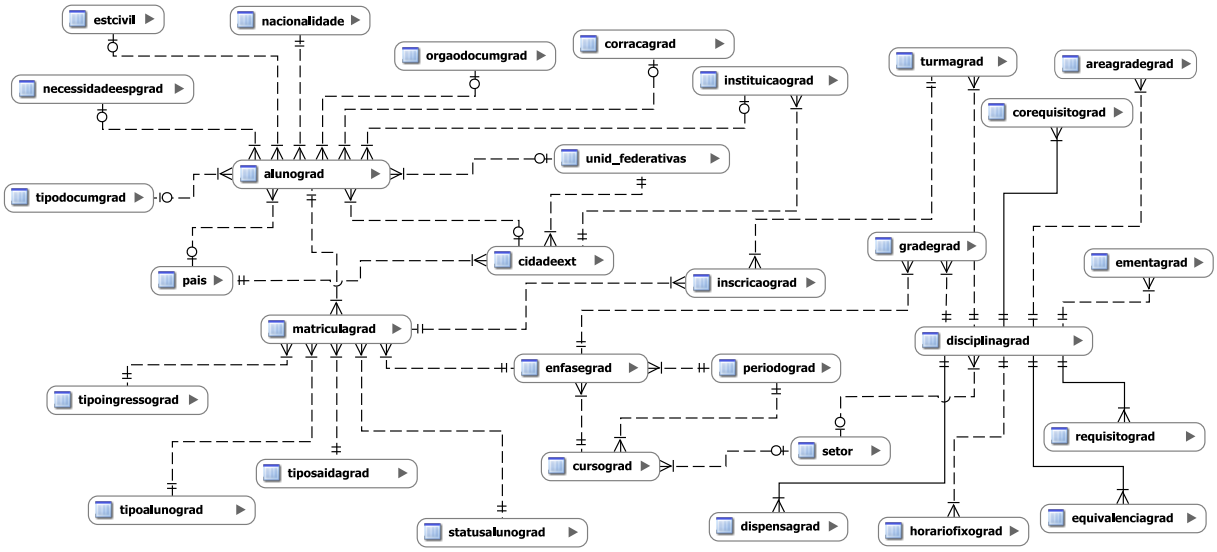


Figura 5.2: Modelo ER parcial do banco de dados do Progradweb.

A partir do banco de dados, foi realizada a geração do código das classes de entidade na linguagem Java, por meio do *framework* de mapeamento objeto-relacional Hibernate[4] e auxílio da IDE NetBeans[5]. Dessa forma, foi gerada uma classe em Java para cada tabela existente no banco de dados, contendo código, anotado conforme a API JPA, de atributos, relacionamentos e métodos de acesso. A Figura 5.3 exemplifica essa geração de código.

Em seguida, um metaprograma foi reutilizado e executado para obter um modelo OO de classes descrito segundo o padrão XMI. Esse metaprograma, mostrado parcialmente na Figura 5.4, possui regras de transformação que utilizam mecanismos de reflexão da linguagem Java para obter as definições das estruturas das classes, processá-las e gerar *tags* em um documento de acordo com o padrão XMI. Nesse metaprograma, a API DOM foi utilizada para realizar a manipulação das estruturas no formato XML, utilizadas no modelo XMI gerado.

Após a execução do metaprograma, o XMI do modelo OO de classes foi gerado e importado utilizando a ferramenta CASE Papyrus UML, obtendo-se a representação

[4]Hibernate - http://www.hibernate.org/
[5]http://www.netbeans.org/

Figura 5.3: Exemplo de geração de código das classes de entidade em Java a partir de um banco de dados.

```
1  import org.w3c.dom.Document;
2  import org.w3c.dom.Element;
3  import java.lang.reflect.Field;
4  import java.lang.reflect.Method;
5  import java.lang.reflect.Modifier;
6  import javax.xml.parsers.DocumentBuilder;
7  import javax.xml.parsers.DocumentBuilderFactory;
8  ...
9
10 class XmiMetaprogram {
11
12 private DocumentBuilderFactory docBuilderFac;
13 private DocumentBuilder docBuilder;
14 private Document doc;
15 ...
16
17 public generateXMI() {
18   ...
19   for (Class c : classes) {
20      Element classNode = doc.createElement("packagedElement");
21      classNode.setAttribute("xmi:type", "uml:Class");
22      classNode.setAttribute("name", c.getSimpleName());
23
24      for (Field f : c.getDeclaredFields()) {
25         Element attributeNode = doc.createElement("ownedAttribute");
26         attributeNode.setAttribute("name", f.getName());
27         if (Modifier.isPublic(f.getModifiers())) {
28            attributeNode.setAttribute("visibility", "public");
29         }
30         else if (Modifier.isPrivate(f.getModifiers())) {
31            attributeNode.setAttribute("visibility", "private");
32         }
33         else if (Modifier.isProtected(f.getModifiers())) {
34            attributeNode.setAttribute("visibility", "protected");
35         }
36         ...
37         classNode.appendChild(attributeNode);
38      }
39
40      for (Method m : c.getDeclaredMethods()) {
41         Element operationNode = doc.createElement("ownedOperation");
42         operationNode.setAttribute("name", m.getName());
43         if (Modifier.isPublic(m.getModifiers())) {
44            operationNode.setAttribute("visibility", "public");
45         }
46         else if (Modifier.isPrivate(m.getModifiers())) {
47            operationNode.setAttribute("visibility", "private");
48         }
49         else if (Modifier.isProtected(m.getModifiers())) {
50            operationNode.setAttribute("visibility", "protected");
51         }
52         ...
53         classNode.appendChild(operationNode);
54      }
55      ...
56   }
57 }
58 }
```

Figura 5.4: Código parcial do metaprograma utilizado para a geração do modelo XMI.

gráfica do modelo OO. A Figura 5.5 mostra parte do modelo OO obtido nessa etapa.

Uma vez obtido o modelo OO, a execução da etapa de Engenharia Reversa está concluída. Em seguida, a execução da etapa de Engenharia Avante é detalhada.

5.2.2 Execução da EAv

Prosseguindo com as etapas do processo proposto, o modelo OO obtido na ER foi refinado por um Engenheiro de Software para atender novos requisitos relacionados com a arquitetura alvo do PogradWeb. Basicamente, o modelo OO sofreu diferentes alterações para se adequar a uma arquitetura de software conforme o padrão MVC e estereótipos foram adicionados às classes do modelo OO para agregar maior signifi-

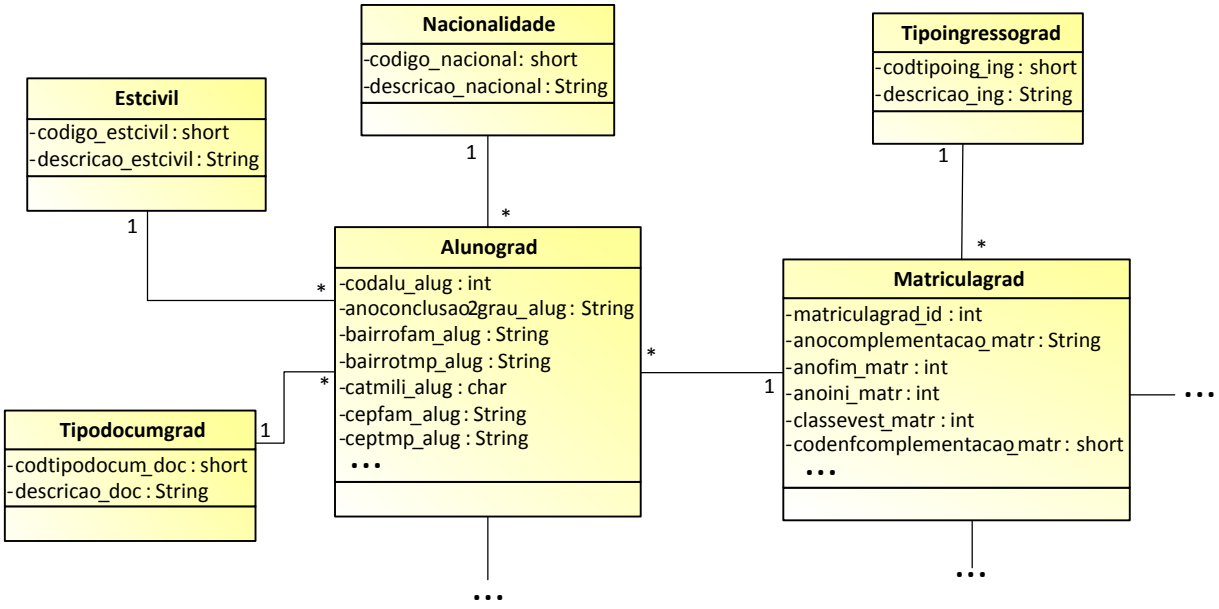


Figura 5.5: Modelo OO parcial do Pogradweb obtido na ER.

cado ao modelo e auxiliar na geração de código a ser realizada a partir do modelo.

Em seguida, transformações M2C e metaprogramas foram reutilizados e executados para realizar a geração de grande parte do código fonte referente à arquitetura do sistema alvo na linguagem Java. Por exemplo, foram gerados os códigos das classes (persistentes e não-persistentes) presentes no modelo OO, das interfaces gráficas destinadas à realização das operações CRUD (*Create, Retrieve, Update e Delete*), componentes gerenciados do *framework* Java Server Faces (JSF) e dos conversores de tipos utilizados pelo JSF. Adicionalmente, também foi gerado código dos componentes de sessão no padrão Fachada e de serviços Web RESTFul para prover interoperabilidade entre outros sistemas da universidade.

Dado que, no código gerado, estão presentes as classes de entidade existentes no modelo OO refinado contendo as devidas anotações segundo a API JPA, o *framework* Hibernate foi utilizado para refletir as alterações que, eventualmente, existiram entre o banco de dados original do PogradWeb e as classes de entidades geradas a partir do modelo OO refinado. Dessa forma, o banco de dados foi atualizado conforme as alterações que foram realizadas nas classes de entidade presentes no modelo OO.

Finalizada a geração de código, obteve-se a implementação de uma nova arquitetura do PogradWeb. A Figura 5.6 mostra uma comparação da arquitetura do PogradWeb antes e depois do processo de reengenharia.

Ao final do processo, obteve-se a implementação parcial da nova arquitetura pre-

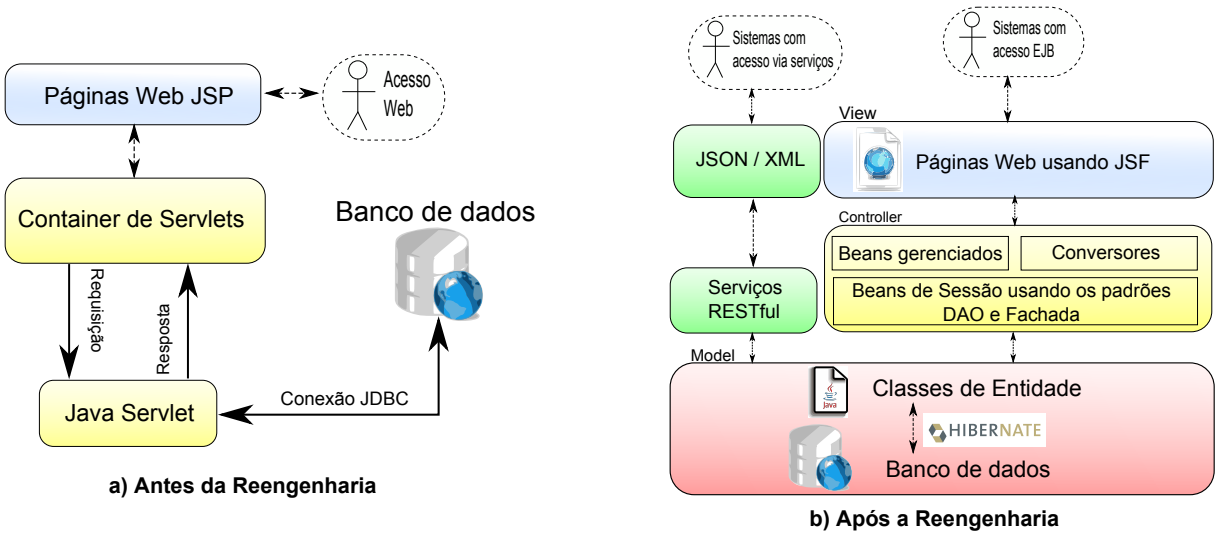


Figura 5.6: Arquitetura do PrograWeb antes e após o processo de reengenharia.

tendida para sistema PrograWeb alvo. Além disso, como artefatos de saída do processo de apoio à reengenharia, obteve-se o banco de dados reconstruído e o modelo OO do sistema. Uma vez que a geração foi concluída, a implementação resultante foi implantada em um servidor de aplicações *JBoss Application Server*[6] para a realização de testes e prosseguimento da reengenharia. Dessa forma, essa primeira versão do sistema serviu de base para sua evolução para outras versões cada vez mais refinadas.

5.2.3 Avaliação dos resultados

Com o objetivo de ter uma ideia melhor do ganho proporcionado por meio da utilização do processo proposto para apoiar a reengenharia do PrograWeb nesse estudo de caso, esta seção apresenta uma cálculo de estimativa do tempo (medido em meses) e esforços (medido em pessoas-mês) que seriam gastos, caso os artefatos gerados durante o processo proposto fossem desenvolvidos seguindo o ciclo de vida clássico do software desde o início. Dessa forma, é possível conhecer um valor estimado de tempo e esforços que, supostamente, puderam ser poupados durante o processo, na etapa de reimplementação do sistema. Esse cálculo teve como base a quantidade linhas de código fonte (*Source Line of Code - SLOC*) (sem considerar linhas em branco e comentários) dos artefatos gerados.

Contabilizando os artefatos gerados que são considerados nesse cálculo, no total,

[6]JBoss AS - http://www.jboss.org/jbossas/

foram geradas 86.947 SLOC's, distribuídas em 1.127 arquivos e tendo como base o banco de dados do PrograduWeb com 155 tabelas.

Para realizar o cálculo da estimativa de tempo e esforço, utilizou-se um modelo empírico, com base na Equação do Software, proposto por Putnam e Myers (1992), como mostra a Equação (5.1). Dessa forma, foram estimados o tempo e esforço que seriam gastos no desenvolvimento da arquitetura do sistema seguindo o processo com base no ciclo de vida clássico do software (PRESSMAN, 2005).

$$E = [LOC * B^{0,333}/P]^3 * (1/t^4) \tag{5.1}$$

em que:

E = esforço em pessoas-mês ou pessoas-ano
t = duração do projeto em meses ou ano
LOC = linhas de código fonte da aplicação
B = fator de aptidões especiais
P = parâmetro de produtividade

Do total de 86.947 SLOC's de implementação geradas durante as etapas durante o processo proposto, apenas 62.761 SLOC's foram consideradas no cálculo, considerando que 24.186 SLOC's são relativas ao código das classes de entidade, que no ciclo de vida clássico também poderiam ser geradas por uma IDE e um *framework* de mapeamento objeto-relacional, como o Hibernate e a IDE NetBeans.

A partir da Equação (5.1), outras equações são derivadas em (PUTNAM; MYERS, 1992), como o cálculo estimado do tempo de desenvolvimento mínimo (em meses), como mostra a Equação (5.2).

$$t_{min} = 8,14 * (LOC/P)^{0,43} \quad para\ t_{min} > 6\ meses \tag{5.2}$$

Substituindo os valores de LOC=62.721 e P=28.000[7], na Equação (5.2), tem-se que o tempo mínimo para o desenvolvimento da arquitetura da aplicação é, aproximadamente, 11,5 meses.

[7] P = 28.000 é um parâmetro típico para aplicações de sistemas comerciais (PRESSMAN, 2005), sendo a categoria considerado para o sistema PrograduWeb.

Para poder estimar o esforço para o estudo de caso, outra equação proposta em (PUTNAM; MYERS, 1992) foi utilizada para calcular uma estimativa de esforço em pessoas-mês, tendo como base o tempo estimado, conforme mostra a Equação (5.3).

$$E = 180 * Bt^3 \ \ para \ \ E > 20 \ pessoas{-}mês \tag{5.3}$$

Substituindo os valores de B=0,39[8] e t=0,096 ano (equivalente a 11,5 meses) na Equação (5.3), tem-se que o ganho de esforço no desenvolvimento da aplicação é de, aproximadamente, 62 pessoas-mês.

Pelos cálculos realizados, considerando o código possível de ser gerado, estima-se que a utilização do processo proposto proporcionou um ganho aproximado de 62 pessoas-mês no esforço requerido e 11,5 meses no tempo de desenvolvimento, uma vez que o tempo gasto na geração dos artefatos de código a partir da modelagem é considerado desprezível. Nesse caso, é possível observar um indicativo de que o processo contribui para atingir um ganho significativo no tempo e esforços despendidos na implementação da arquitetura do sistema. Além disso, o processo também propiciou maior qualidade para que a finalização da aplicação possa ser realizada, devido à recuperação do modelo OO da aplicação e geração de código com base em padrões de software. Outro ponto a ressaltar é que o modelo OO reconstruído, o novo banco de dados e o código do sistema gerados contribuem para orientar o Engenheiro de Software na continuação do desenvolvimento do sistema e na realização de futuras manutenções.

Apesar do ganho atingido, um possível questionamento a ser realizado está relacionado com o tempo gasto para a construção de metaprogramas e transformações M2C. Para a realização do estudo de caso, foram gastos 2,2 meses para a construção desses artefatos. Mesmo considerando esse tempo no cálculo realizado, tem-se um ganho no tempo de desenvolvimento de 9,3 meses e 33 pessoas-mês no esforço. No entanto, os artefatos de geração de código desenvolvidos podem ser reutilizados em qualquer sistema que utilize a linguagem UML na modelagem e a linguagem Java na implementação. Nesse caso, o tempo gasto nessa construção não necessitaria ser considerado, caso fossem reutilizados para realizar a geração de código para outros sistemas.

[8] B aumenta quando “cresce a necessidade de integração, teste, garantia de qualidade, documentação e aptidões gerencias"(PUTNAM; MYERS, 1992). Valores típicos para programas (KLOC = 5 a 15), B = 0,16. Para programas maiores, com cerca de 70 KLOC ou mais, B = 0,39.

5.3 Considerações Finais

Este capítulo apresentou os resultados de um estudo de caso realizado com o objetivo de avaliar a utilização do processo proposto com a finalidade de apoiar a reengenharia de sistemas legados por meio do uso de mecanismos de geração de código a partir de modelos.

Um sistema real de médio/grande porte da UFSCar serviu de base para o estudo realizado, que consistiu na realização da reengenharia da arquitetura do sistema para utilização de novas tecnologias e padrões de software.

No processo de reengenharia, seguiu-se as etapas de Engenharia Reversa e Engenharia Avante da EA, conforme o processo proposto. Um modelo de classes orientado a objetos foi recuperado a partir do banco de dados do sistema e serviu como base para a geração de código para a implementação de uma nova arquitetura seguindo o padrão MVC.

Por meio de uma comparação realizada com um modelo de estimativa empírico, foi possível ter um indicativo de que o processo, de fato, pode contribuir para fornecer subsídios com o objetivo de reduzir a quantidade de tempo e esforços despendidos durante o processo de reengenharia de um sistema. Esse resultado demonstra um potencial do processo proposto para ter seu uso em projetos reais e reforça os benefícios dos conceitos de metaprogramação e MDD que foram aplicados em sua concepção.

Capítulo 6

TRABALHOS CORRELATOS

6.1 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.

A organização deste capítulo está realizada da seguinte forma: na Seção 6.2 é realizada uma discussão dos conceitos utilizados durante o desenvolvimento deste trabalho de pesquisa relacionando com outros trabalhos da literatura. Na Seção 6.3 é apresentado um breve resumo de alguns trabalhos relacionados que contribuíram com ideias para o desenvolvimento do processo proposto, mostrando uma análise comparativa na Seção 6.4. Por fim, na Seção 6.5 são apresentadas as considerações finais do capítulo.

6.2 Contextualização

Dentre as diversas soluções pesquisadas na literatura, nota-se uma grande variedade de trabalhos que utilizam os conceitos abordados para diferentes propósitos. Dessa forma, torna-se necessário discutir as decisões tomadas no trabalho desenvolvido em relação aos demais trabalhos que vem sendo pesquisados na área.

Um dos pontos importantes está relacionado ao uso de *frameworks* ORM, que é uma prática comum nos dias atuais (CERNY; SONG, 2010), dado que grande parte das aplicações são candidatas a utilizarem o mapeamento ORM, embora existam

questões que inviabilizam seu uso em certos casos, como alguns tipos de sistemas que necessidade extrema de desempenho. Sabe-se que existem diferentes formas de realizar o mapeamento objeto-relacional de um modelo OO para fazer a persistência em um banco de dados relacional (KELLER, 1997; BAUER; KING, 2006). Isso acaba ocasionando problemas relacionados à diferença de granularidade no mapeamento. Dessa forma, por exemplo, duas ou mais classes no modelo OO podem ser persistidas em uma única tabela no banco de dados ou vice-versa. No processo proposto, para efeito de geração, a recuperação do modelo OO a partir do banco de dados resulta na geração de uma classe de entidade para cada tabela no banco de dados. No entanto, o modelo gerado pode ser facilmente adaptado pelo Engenheiro de Software, que pode tomar as melhores decisões de projeto da arquitetura em um nível maior de abstração, diretamente no modelo. O uso de estereótipos da UML durante o refinamento do modelo OO é outro fator que auxilia os metaprogramas e transformação M2C a resolverem esse tipo de questão.

Outra questão relevante a ser abordada está relacionada com a linguagem de modelagem usada na construção dos modelos em um processo de reengenharia dirigida a modelos. Algumas abordagens (WANG; SHEN; CHEN, 2009; CERNY; SONG, 2010) utilizam a UML para descrever os modelos manipulados durante o processo, porém outros trabalhos (IZQUIERDO; MOLINA, 2012; PEÑA; CORREAL; HERNANDEZ, 2010) preferem adotar uma DSL própria. As DSL's estão restritas a determinados domínios e exigem que os profissionais gastem tempo para seu entendimento, embora facilitem a geração de código de artefatos mais específicos do sistema. Neste trabalho, houve um enfoque na UML pelo fato de ser uma linguagem amplamente conhecida e difundida entre os profissionais da área de Engenharia de Software e, sendo de propósito geral, pode ser adotada na maior parte dos projetos de sistemas computacionais. Além disso, a utilização de perfis na UML maximiza seu poder de geração de código para diferentes tipos de aplicações e pode contribuir para adequar os modelos construídos para uso em domínios mais específicos.

Levando em consideração trabalhos relacionados à reengenharia de software, grande parte das abordagens que têm sido propostas procuram realizar a reengenharia usando tecnologias específicas como, por exemplo, *Web Services* (UMAR; ZORDAN, 2009; FUHR et al., 2011; CASTILLO; GARCÍA-RODRÍGUEZ; CABALLERO, 2009) e migração de tecnologias antigas para tecnologias mais recentes (SNEED, 2010). Por outro lado, no desenvolvimento do processo proposto, optou-se por definir um guia de propósito geral para apoiar a reengenharia sistemas legados. Além disso,

o uso da metaprogramação por meio da reflexão é uma alternativa com menor grau de complexidade em sua utilização em comparação com outras abordagens, como a representação do código fonte como grafos (CORREIA et al., 2007; HUNOLD et al., 2009).

A realização de experimentos e estudos de caso práticos também foi um ponto de destaque em uma série de trabalhos presentes na literatura. Por exemplo, Heijstek e Chaudron (2009) apresentam um estudo a respeito do impacto da adoção do MDD em larga escala por meio da investigação de diferentes características de um projeto industrial de grande porte no qual o MDD foi aplicado em sua forma pura. O estudo focou na análise da complexidade e tamanho dos modelos produzidos no projeto considerando métricas relacionadas à qualidade e esforços de modelagem.

Na mesma linha, Hutchinson, Rouncefield e Whittle (2011) descrevem três estudos de casos referentes à implantação da MDE na indústria. O estudo destaca a importância de fatores sociais, técnicos e organizacionais como requisitos para o sucesso na adoção das práticas do desenvolvimento de software dirigido a modelos. Os autores utilizaram questionários, observação e entrevistas para realizar a coleta de dados de profissionais da indústria.

Mohagheghi et al. (2009) e Quintero et al. (2012) apresentam os desafios e barreiras com a adoção do MDD na prática a partir da experiência dos autores em diferentes projetos industriais e acadêmicos. Ricca et al. (2012) realizam uma análise quantitativa com resultados preliminares da adoção de uma ferramenta destinada à manutenção de software no contexto do MDD. Assim como esses trabalhos, na avaliação do processo proposto (Capítulos 4 e 5), buscou-se realizar estudos práticos para obtenção de melhores resultados.

6.3 Trabalhos Encontrados na Literatura

Em seguida, são apresentados alguns dos trabalhos que inspiraram e estão relacionados de diferentes formas com o trabalho proposto. Uma breve descrição dos trabalhos é apresentada, comparando cada trabalho com a pesquisa realizada.

6.3.1 UWE4JSF

UWE4JSF: A Model-Driven Generation Approach for Web Applications (KROISS; KOCH; KNAPP, 2009) é uma abordagem para apoiar a geração de aplicações Web, utilizando a plataforma JSF, com escopo na *UML-based Web Engineering (UWE)*[1].

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[2].

Conforme mostra a Figura 6.1, 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 mapeamentos específicos de elementos.

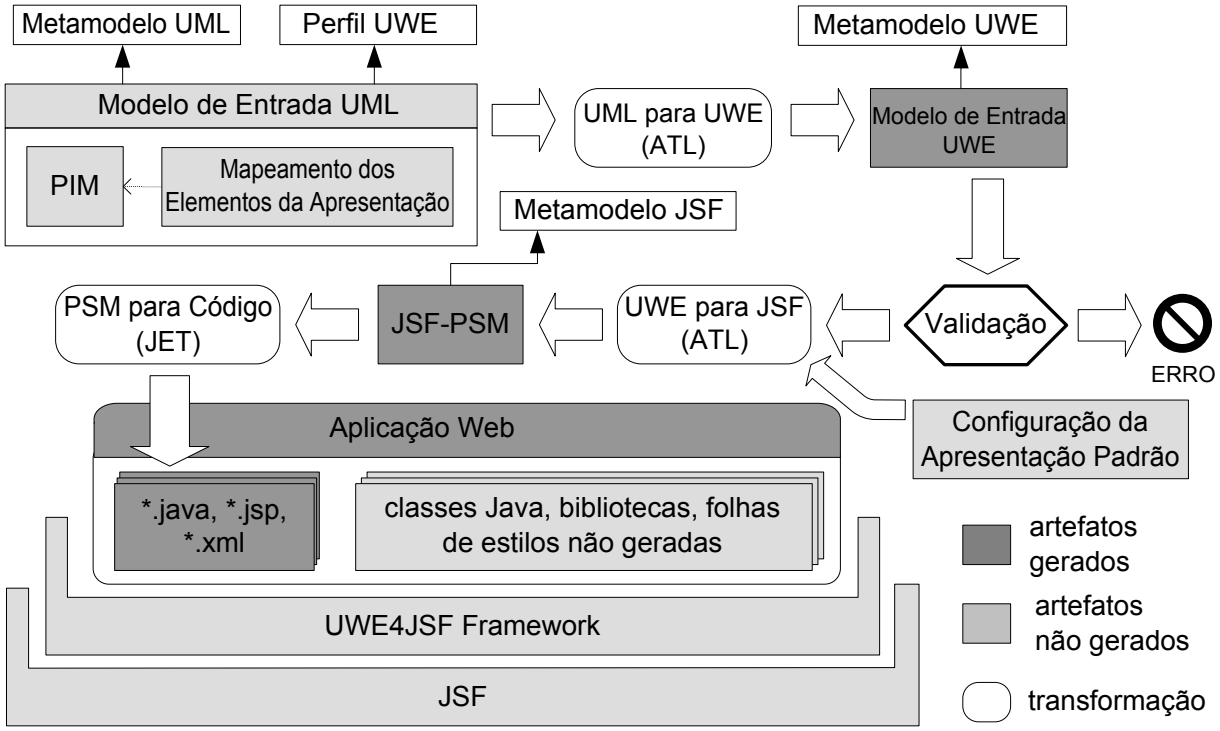


Figura 6.1: Processo de Geração UWE4JSF. Adaptado de (KROISS; KOCH; KNAPP, 2009).

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

[1] UWE - http://uwe.pst.ifi.lmu.de/index.html
[2] Perfil UWE - http://uwe.pst.ifi.lmu.de/publicationsMetamodelAndProfile.html

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, mas destinada especificamente à geração de código para a plataforma JSF a partir de modelos UML. Por outro lado, o processo proposto é propósito geral para a geração de código a partir de modelos e faz uso de mecanismos de metaprogramação e padrões de projeto na geração de código.

6.3.2 VULCAN

VULCAN: A Tool for Automatically Generating Code from Design Patterns (FREDERICK; BOND; TILLEY, 2008) apresenta uma ferramenta que visa facilitar a criação de código de alta qualidade por meio da instanciação automática de padrões de projeto.

Para tornar seu uso mais fácil e prático para os desenvolvedores, essa ferramenta está na forma de *plug-in* para o ambiente de programação Eclipse. Além disso, alguns dos padrões de projeto mais comuns já estão incluídos na ferramenta. No entanto, novos padrões podem ser definidos pelo usuário.

Um diagrama da estrutura interna dos objetos da ferramenta VULCAN é mostrado na Figura 6.2. Esses objetos determinam o fluxo de controle da ferramenta, a leitura da definição dos padrões e a geração de código referente a cada padrão.

A leitura dos padrões é feita por meio de arquivos XML que descrevem cada padrão. Cada arquivo de padrão contém *tags* específicas incluindo o nome do padrão, a categoria, descrição e um *template* do código a ser gerado.

Com o auxílio de um assistente, o usuário tem a opção de criar um novo projeto e selecionar os padrões que deseja gerar. Dessa forma, a ferramenta automaticamente cria uma estrutura de pacotes do projeto para a plataforma Java e os arquivos com os referentes aos padrões selecionados, preenchendo-os com o *template* do código dos respectivos padrões.

Dessa forma, a ferramenta VULCAN visa realizar a geração automática de código

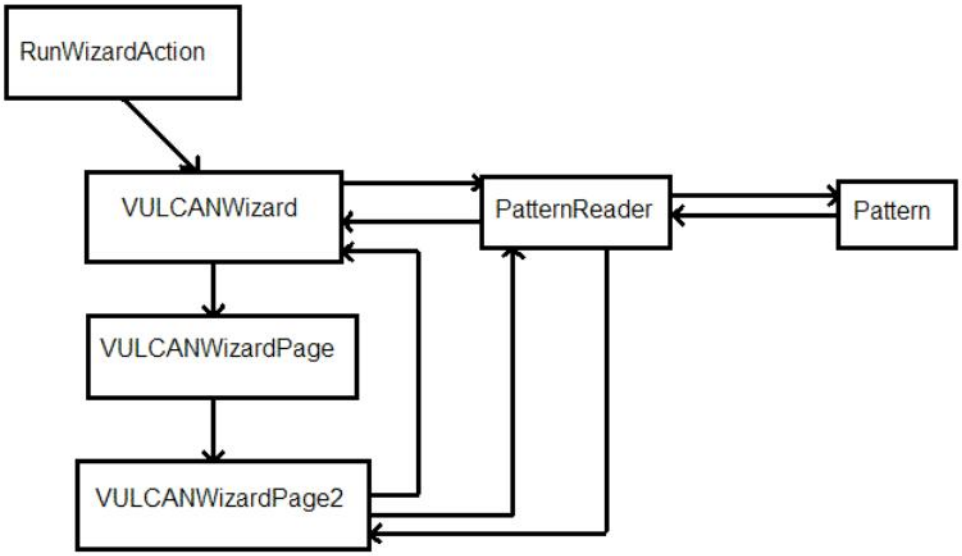


Figura 6.2: Estrutura interna do VULCAN (FREDERICK; BOND; TILLEY, 2008).

de padrões de projeto por meio do uso de *templates*, descritos por meio de arquivos XML previamente desenvolvidos. Em contrapartida, o processo proposto realiza a geração de código da implementação de padrões de projeto por meio de metaprogramas e a partir de modelos. Dessa forma, os modelos tornam o uso do processo proposto mais intuitivo e o código gerado dos padrões não necessitam ser constantemente adaptados, como ocorre com o uso de *templates*.

6.3.3 The Design of an Automated Unit Test Code Generation System

The Design of an Automated Unit Test Code Generation System (FIX, 2009) apresenta um *framework* para geração de código de testes unitários automatizados, visando facilitar a manutenção de software em um ambiente .NET.

Como mostra a Figura 6.3, o processo de geração dos testes inicia com a criação de um metaprograma que utiliza técnicas de reflexão, da linguagem C#, para extrair informações de métodos do sistema que será testado (representado por um arquivo com extensão DLL ou EXE). Essa informação inclui a assinatura dos métodos (nome do método, parâmetros de entrada e saída e tipo de retorno), que é salva em um arquivo de metadados externo no formato XML.

Feito isso, uma transformação, previamente construída utilizando a linguagem *eX-*

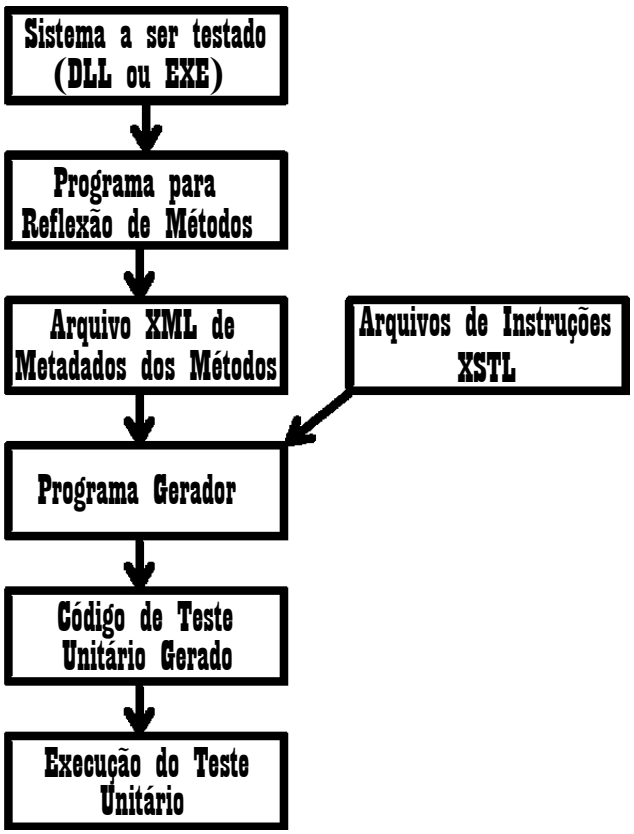


Figura 6.3: Processo de utilização do *framework*. Adaptado de (FIX, 2009).

tensible Stylesheet Language for Transformation (XSLT)[3], é aplicada sobre os metadados no formato XML para a geração dos casos de testes. Esses casos de testes são programas executáveis que fazem várias chamadas dos métodos a serem testados para diferentes parâmetros, a fim de validar suas funcionalidades. Terminada a geração dos casos de testes, o usuário pode executar os casos de testes na IDE Visual Studio ou na ferramenta NUnit.

Comparando esse trabalho com o processo proposto, é possível observar que o *framework* em questão possui enfoque na geração de código de testes unitários a partir de arquivos (DLL ou EXE) de programas desenvolvidos usando a plataforma .NET. Embora tanto o *framework* quanto o processo proposto faça uso de metaprogramas reflexivos para analisar a estrutura de programas, o processo proposto não está associado a uma plataforma específica e se baseia em modelos para realizar a geração de código.

[3]XSLT - http://www.w3.org/TR/xslt

6.3.4 Generating Java code from UML Class and Sequence Diagrams

Generating Java code from UML Class and Sequence Diagrams (PARADA; SIEGERT; BRISOLARA, 2011) apresenta uma abordagem para geração de código estrutural e comportamental, a partir de modelos da UML. Nessa abordagem, as aplicações são modeladas utilizando um diagrama de classe para definir a estrutura da aplicação e diversos diagramas de sequência para representar o comportamento.

A partir do diagrama de classes, os códigos das classes, atributos e assinaturas de métodos são gerados na linguagem Java.

Já os diagramas de sequência, fornecem a informação de invocação de métodos, incluindo argumentos e retornos. Laços de repetição e estruturas condicionais também são capturadas a partir dos diagramas de sequência, de modo que o código Java correspondente (*for/while* ou *switch/if-else*) é gerado.

A proposta é validada por meio de uma ferramenta chamada *GenCode*, que a partir de um modelo UML representado no formato XMI transforma-o em código Java.

Para fins de comparação com o processo proposto, nessa abordagem, é proposta a geração de código estrutural e comportamental em Java a partir dos diagramas de classes e sequência da UML. Por outro lado, o processo proposto é de propósito geral, não restringido as linguagens de modelagem e de implementação utilizadas.

6.3.5 Modernization of Legacy Web Applications into Rich Internet Applications

Esse trabalho propõe um *framework* para realizar a modernização sistemática e semiautomática de aplicações Web legadas (*WA - Web Aplications*) em aplicações de internet ricas (*RIA - Rich Internet Application*) de acordo com os princípios definidos pelo *OMG Architecture Driven Modernization (ADM)*.

Primeiramente, é realizada uma extração estática e dinâmica de informações a partir do código fonte (dados, lógica e apresentação) da aplicação. Dessa forma, o *layout* de interface, a descrição das páginas Web e os mapas de navegação e operação são recuperados em uma primeira etapa. Em seguida, o conhecimento obtido é representado e refinado utilizando um modelo independente de tecnologia a partir das especificações geradas na primeira etapa. Esse modelo de conhecimento é des-

crito conforme o *Knowledge Discovery Metamodel (KDM)* por meio da execução de transformações M2M.

Opcionalmente, é realizada uma projeção do sistema conceitual utilizando modelos específicos para RIA e um refinamento dos modelos Web por meio da aplicação de padrões RIA. Por fim, a aplicação RIA alvo é gerada seguindo os princípios do MDD por meio do reuso de técnicas e ferramentas previamente definidas como, por exemplo, transformações M2C usando *OMG MOF Model to Text*, *JET*, *XPand* e outras. A Figura 6.4 mostra uma visão geral do processo de modernização proposto.

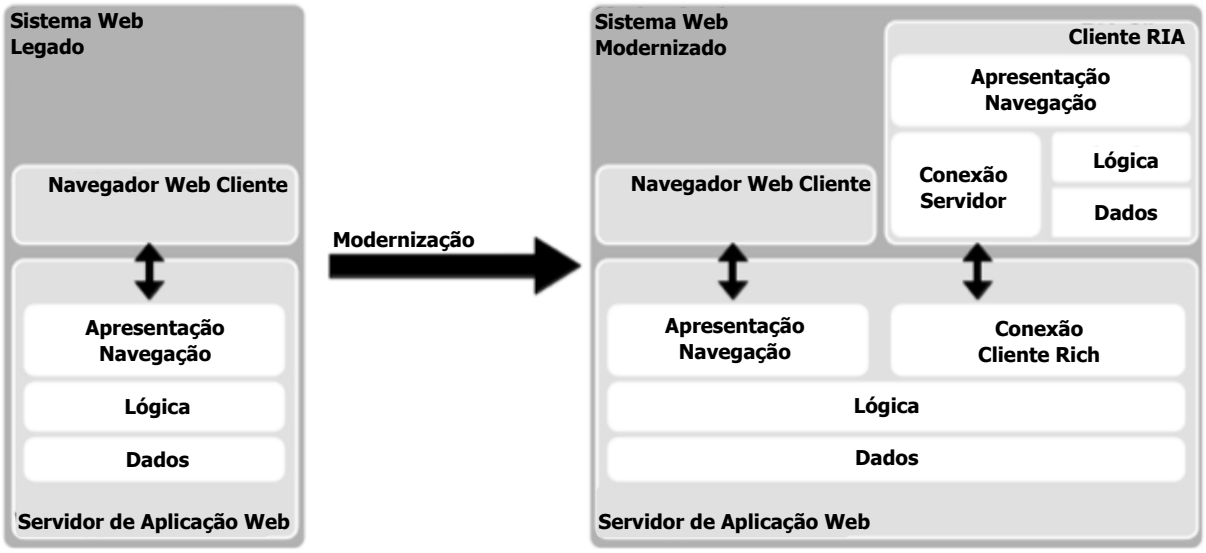


Figura 6.4: Processo de modernização. Adaptado de (RODRÍGUEZ-ECHEVERRÍA et al., 2012).

Em síntese, no trabalho em questão, é proposto uma abordagem para modernização de aplicações Web legadas em aplicações de interfaces ricas. São aplicados princípios do MDD e o uso de especificações OMG *Architecture Driven Modernization (ADM)* para a geração de interfaces ricas a partir do código fonte das camadas de apresentação e navegação de aplicações Web legadas. O processo proposto difere desse trabalho por possuir um propósito mais geral, não se destinando apenas a um único tipo de aplicações, tais como aplicações Web. Além disso, o processo proposto oferece a possibilidade de usar o banco de dados no processo de reengenharia.

6.3.6 Model-Driven Reengineering of Database

Model-Driven Reengineering of Database (WANG; SHEN; CHEN, 2009) propõe uma abordagem com base na MDA para apoiar o processo de engenharia reversa de

banco de dados (*Database Reverse Engineering - DBRE*) relacionais. Como mostra a Figura 6.5, a abordagem é composta por seguidas transformações de modelos e dividida nas etapas de extração e contextualização.

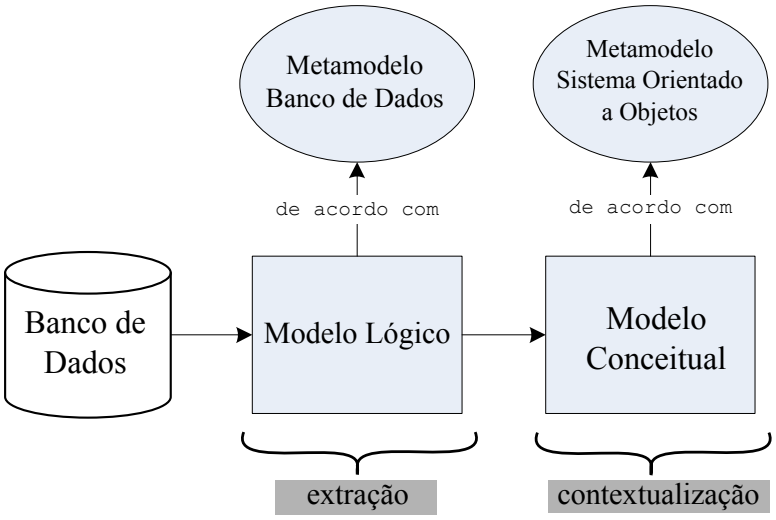


Figura 6.5: Abordagem para reengenharia de banco de dados. Adaptado de (WANG; SHEN; CHEN, 2009).

Na etapa de extração, um modelo específico de plataforma (PSM) é obtido a partir da estrutura do banco de dados (modelo lógico). Já na etapa de contextualização um modelo independente de plataforma (modelo conceitual) é gerado a partir do PSM. Como resultado do processo, têm-se modelos de entidade-relacionamento e diagramas de classes.

De modo semelhante ao trabalho em questão, o processo proposto oferece suporte à reengenharia de sistemas legados, levando em consideração a estrutura de banco de dados existente. Assim, por meio do uso de uma processo dirigido a modelos, é possível obter um modelo com maior nível de abstração (e.g diagrama de classes) que auxilia no processo de reengenharia. Entretanto, o processo proposto difere-se desse trabalho por utilizar *frameworks* de mapeamento objeto-relacional e técnicas de metaprogramação na obtenção de um modelo da estrutura do banco de dados, ao invés de utilizar seguidas transformações de modelos, implicando em uma forma mais simples e rápida de extrair conhecimento a partir do banco de dados, considerando a disponibilidade de tecnologias existentes.

6.3.7 Reengineering of Java Legacy System Based on Aspect Oriented Programming

Chen et al. (2010) propõem uma abordagem para apoiar a reengenharia de sistemas legados desenvolvidos em Java a partir do código executável do programa. Durante a abordagem, são utilizadas técnicas de reflexão e descompilação com o objetivo de gerar diagramas de classes e diagramas de sequência a partir do *bytecode* do sistema.

Em um primeiro momento, técnicas de metaprogramação são utilizadas por meio do uso do mecanismo de reflexão da linguagem Java para obter as informações a respeito dos nomes das classes, interfaces, super classes, modificadores, construtores e assinaturas de métodos. A partir dessas definições, um diagrama de classes pode ser construído.

Em seguida, um diagrama de sequência preliminar é obtido a partir do uso de ferramentas de descompilação (e.g Javap.exe ou ASM) no arquivo executável do sistema. A partir das chamadas de métodos identificadas, usando a linguagem AspectJ[4], arquivos de aspectos são construídos com pontos de cortes específicos para identificar possíveis padrões no código legado, tais como uso de interfaces e injeção de dependência.

Portanto, o trabalho propõe uma abordagem para apoiar a reengenharia de sistemas legados por meio da reconstrução de diagramas de classe e sequência a partir do *bytecode* executável de um sistema, construído usando a linguagem Java e programação orientada a aspectos (OA). De forma análoga ao processo proposto, na geração de modelos, técnicas de metaprogramação reflexiva são utilizadas. No entanto, o processo proposto se difere desse trabalho porque não se concentra na utilização do paradigma OA e possibilita realizar a reengenharia a partir do banco de dados do sistema legado, definindo diretivas para a etapa de engenharia avante (geração de código a partir dos modelos recuperados), ao invés do trabalho em questão cujo enfoque está na etapa de engenharia reversa.

[4] AspectJ - http://eclipse.org/aspectj/

6.4 Resumo Comparativo Entre os Trabalhos

A partir da apresentação de diferentes pesquisas relacionadas, nota-se que o trabalho desenvolvido possui diversas características abordadas nas pesquisas descritas anteriormente. No entanto, o enfoque dado e a forma com que os tais características foram utilizadas fazem com que o trabalho proposto apresente suas próprias contribuições.

De modo geral, os trabalhos apresentados diferem-se do trabalho desenvolvido em relação ao emprego de um processo de propósito geral para realizar geração de código a partir de modelos, ao uso de metaprogramação para realizar geração de código no processo, à adoção de padrões de projeto na geração de código e apoio ao processo de reengenharia de software.

Ideias e conceitos foram aproveitados de diferentes trabalhos, embora outros tenham sido adicionados à proposta no sentido de obter resultados a partir de experimentos e estudos de caso práticos, sobretudo, com relação ao uso de mecanismos de geração de código. No entanto, diferentemente da maioria dos trabalhos, o trabalho proposto possui ênfase na análise estatística quantitativa de dados obtidos durante as avaliações realizadas. Além disso, os resultados deste trabalho estão direcionados a um escopo especificamente relacionado à utilização de geração de código a partir de modelos e o estudo do respectivo impacto no desenvolvimento de software.

O Quadro 6.1 resume a análise comparativa do trabalho proposto com os trabalhos correlatos.

6.5 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

Quadro 6.1: Análise comparativa do processo proposto com os trabalhos correlatos apresentados.

Características	Trabalhos							
	Proposta	UWE4JSF: A model-driven generation approach for web applications	VULCAN: A Tool for Automatically Generating Code from Design Patterns	The Design of an Automated Unit Test Code Generation System	Generating Java code from UML Class and Sequence Diagrams	Modernization of Legacy Web Applications into Rich Internet Applications	Reengineering of Java Legacy System Based on Aspect-Oriented Programming	Model-Driven Reengineering of Database
Realiza geração de código a partir de modelos	✓	✓	✗	✗	✓	✓	✗	✓
Utiliza metaprogramação na geração de código	✓	✗	✗	✓	✗	✗	✓	✗
Suporta geração de código de padrões de projeto	✓	✗	✓	✗	✗	✓	✗	✗
Proposta independente de plataforma	✓	✗	✗	✗	✗	✓	✗	✗
Oferece suporte ao processo de reengenharia	✓	✗	✗	✗	✗	✓	✓	✓
Propõe a criação de mecanismos reutilizáveis	✓	✓	✓	✓	P	✓	P	✓
Prevê utilização de banco de dados	✓	✗	✗	✗	✗	✗	✗	✓
Oferece ferramental de suporte	P	✓	✓	P	P	✗	✗	✗

o suporte ao desenvolvimento de software por meio da utilização técnicas de geração de código a partir de modelos.

Capítulo 7

CONCLUSÃO

Há tempo que, no mercado de trabalho, existe uma grande demanda pelo desenvolvimento de aplicações com alto nível de qualidade em um espaço de tempo cada vez menor. 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 que viabilizem o aumento de produtividade na construção de sistemas computacionais.

Nesse sentido, o estudo e utilização de técnicas de geração de código foi um dos principais pontos explorados neste trabalho. Um processo de software foi proposto com base nas concepções de Desenvolvimento Dirigido a Modelos (MDD) e de metaprogramação. O processo proposto define atividades e artefatos (Engenharia de Domínio - ED) com o objetivo de auxiliar as etapas de modelagem e de geração de código ao longo do desenvolvimento de uma aplicação (Engenharia de Aplicação - EA).

Com o objetivo de aplicar o processo proposto em um contexto de reengenharia de software, uma adaptação no processo foi proposta visando aproveitar os artefatos do processo e os conceitos de MDD e metaprogramação para apoiar a reengenharia de sistemas legados. A EA do processo de reengenharia é dividida em Engenharia Reversa (ER) e Engenharia Avante (EAv). Na ER, um modelo OO da aplicação legada, no padrão XMI, é obtido e usado como base para a reconstrução da nova aplicação, seguindo os princípios do MDD, durante a EAv.

Além disso, para testar a aplicabilidade e quantificar os resultados obtidos por meio da utilização do processo proposto, duas avaliações foram conduzidas. A primeira avaliação consistiu de um estudo experimental, por meio da comparação dos

tempos gastos por grupos de estudantes participantes na construção de uma aplicação Web do tipo CRUD, utilizando tanto o processo proposto quanto o processo com base no ciclo de vida clássico do software. A segunda avaliação consistiu de um estudo de caso, no qual uma reengenharia da atual arquitetura de software do sistema PrograduWeb da UFSCar foi realizada e, por meio de um cálculo empírico, estimou-se valores de tempo e esforço possivelmente poupados com a utilização do processo proposto.

Este capítulo está organizado da seguinte forma: na Seção 7.1 são destacadas as contribuições deste trabalho, discutindo os principais resultados obtidos. As limitações referentes ao trabalho desenvolvido são identificadas e apresentadas na Seção 7.2. Por fim, na Seção 7.3, são apresentadas algumas possibilidades de trabalhos futuros do trabalho desenvolvido.

7.1 Contribuições e Síntese dos Principais Resultados

Uma das contribuições do trabalho desenvolvido é a construção e utilização de mecanismos de geração de código, combinando os conceitos e técnicas relacionados ao MDD e à metaprogramação. Em grande parte das abordagens que utilizam o MDD, o código da aplicação e suas funcionalidades é gerado apenas a partir dos modelos previamente construídos. Indiretamente, esta característica acaba aumentando o nível de complexidade das transformações de modelos construídas, de modo que sua manutenção e utilização para diferentes situações se tornam onerosas dependendo da complexidade da aplicação. Além disso, no caso de sistemas que já existam e não possuem modelagem ou documentação, existe um grande trabalho para construir os modelos e geradores necessários.

No trabalho desenvolvido, várias etapas de geração de código foram realizadas durante a construção da aplicação. Na definição do processo proposto, o código gerado a partir de modelos se restringiu apenas à informação explicitamente definida pelos modelos construídos. Por exemplo, a partir de um diagrama de classes, apenas os código de cada classe, seus respectivos atributos, métodos e relacionamentos foi gerado. Dessa forma, código referente às funcionalidades específicas (e.g camada de persistência, interfaces gráficas e etc) não foi gerado por meio de transformações M2C. Toda a geração de funcionalidades foi realizada por meio de metaprogramas específicos para cada caso (e.g metaprograma para geração de interfaces CRUD e

metaprograma para geração da camada de persistência e etc).

Dentre as vantagens dessa estratégia utilizada, tem-se que os geradores de código ficaram mais modulares, permitindo que diferentes metaprogramas sejam executados a partir de um mesmo conjunto de classes gerados por uma transformação M2C. Além disso, devido a essa maior modularidade, a manutenção dos geradores de código tende a ser menos onerosa em comparação com geradores únicos e complexos, que englobam a geração de todas as partes da aplicação. No entanto, dentre as possíveis desvantagens, existe a dificuldade em manter um maior número de geradores de código 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 da realização de um experimento para avaliar a utilização do processo proposto no desenvolvimento de sistemas. A população de participantes do experimento foi composta por alunos de graduação dos últimos anos da UFSCar que desenvolveram uma aplicação do tipo CRUD com base no sistema PrograduWeb, seguindo tanto uma abordagem de desenvolvimento com base no ciclo de vida clássico, quanto à utilização de mecanismos de geração de código a partir de modelos no processo proposto. 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, uma análise da opinião dos participantes apontou que o processo proposto contribuiu, dentre outros benefícios, para reduzir as dificuldades encontradas pelos participantes do experimento.

Os resultados da avaliação experimental reafirmam os benefícios obtidos por meio do uso das abordagens de geração de código dirigida por modelos. Por exemplo, o aumento de produtividade e a redução de erros cometidos no decorrer do desenvolvimento foram os pontos mais explorados durante a avaliação realizada. Embora o estudo tenha sido realizado em ambiente universitário, os resultados obtidos possuem validade e, respeitando as condições adotadas, podem ser utilizados em outros projetos de pesquisa que visem avançar ainda mais no que diz respeito ao estado da arte nas áreas de pesquisa relacionadas.

Com relação à avaliação do processo proposto no contexto de reengenharia de software, um estudo de caso foi conduzido para verificar a viabilidade da utilização do processo no apoio à reengenharia do sistema PrograduWeb da UFSCar. Analisando os

resultados com base em um modelo empírico de estimativa de software, observou-se um ganho no tempo de desenvolvimento e no esforço despendido para a realização da reengenharia. Para alcançar esse ganho, a metaprogramação teve papel fundamental, oferecendo facilidade na obtenção do modelo OO representado em UML e na geração de grande parte do código. Além disso, o processo proposto possibilita auxiliar na portabilidade da aplicação legada para um processo de desenvolvimento dirigido a modelos, oferecendo maior facilidade e flexibilidade para a realização de manutenções futuras.

De forma resumida, o trabalho desenvolvido contribui para a Engenharia de Software nas seguintes áreas:

- **Geração de Código**: o estudo da combinação de técnicas de geração de código a partir de modelos e via metaprogramação contribui para a construção de geradores modulares, eficazes e melhor adaptáveis a diferentes contextos e domínios.
- **Reúso de Software**: as atividades de Engenharia de Domínio do processo proposto, promove a construção de transformações M2C e metaprogramas que podem ser aproveitados no desenvolvimento de outros sistemas, incentivando os desenvolvedores de software na prática do reúso.
- **Metaprogramação**: o uso de metaprogramação possibilita o processamento da estrutura das classes e a geração de artefatos, tais como o modelo XMI das classes e código fonte para diferentes plataformas, sendo possível ganhar tempo tanto na codificação quanto na recuperação de modelos mais abstratos. Dessa forma, menos esforço é despendido em tarefas repetitivas, passíveis de automatização.
- **Reengenharia de Software**: o processo proposto oferece um guia geral para apoiar a reengenharia de sistemas legados, auxiliando na adoção do MDD. A recuperação do modelo OO e a geração parcial do código contribuem para a redução de tempo e esforços gastos, possibilitando aos desenvolvedores manter o foco maior em tarefas mais relevantes do desenvolvimento de software.
- **Suporte Computacional**: as transformações M2C e metaprogramas construídos para teste e avaliação do trabalho desenvolvido oferece um suporte computacional para apoiar Engenheiros de Software em diferentes domínios de aplicações.

7.2 Limitações

Uma vez tendo concluído o desenvolvimento do trabalho proposto, algumas limitações foram identificados por meio de uma análise crítica.

O processo proposto foi planejado 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, de forma semelhante ao feito na definição do processo proposto para apoiar a reengenharia de sistemas legados (Seção 3.3).

Outra limitação existente no trabalho desenvolvido está relacionada aos resultados obtidos a partir do estudo experimental apresentado no Capítulo 4. Embora os resultados do estudo tenham evidenciado ganhos no tempo de desenvolvimento dos participantes quando seguiram o processo proposto, é necessário considerar que esses resultados estão limitados ao escopo de desenvolvedores de software em ambiente universitário e às tecnologias adotadas. Considerando questões de validade, para estender e generalizar os resultados obtidos para um contexto mais amplo, torna-se necessária a reaplicação do estudo experimental em ambientes industriais e, preferencialmente, com maior número de participantes.

No estudo de caso descrito no Capítulo 5, uma limitação dos resultados obtidos refere-se ao método de estimativa empírico utilizado. Nesse método, proposto por Putnam e Myers (1992), os valores estimados de tempo e esforço obtidos pelo cálculo tiveram como base a medida do sistema em LOC. Embora o uso da quantidade de LOC de um sistema seja uma medida fácil e rápida de ser calculada, a mesma apresenta uma grande variação dependendo da linguagem de programação utilizada na implementação. Além disso, apesar das evidências favoráveis nos resultados obtidos, essa variação pode gerar imprecisões e novas avaliações, usando, por exemplo, métricas orientadas à função, são pontos relevantes a ser explorados.

Os resultados obtidos, tanto na experimentação quanto no estudo de caso realizado compararam o processo proposto com o desenvolvimento com base no ciclo de vida clássico, efetuado de forma manual. Uma outra avaliação, que não foi considerada neste trabalho, refere-se à realização de um estudo comparativo com outras

abordagens de geração de código com técnicas diferentes às utilizadas pelo processo proposto.

7.3 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 do processo proposto;
- A construção de novos metaprogramas para dar suporte à geração de código para diferentes tipos de arquiteturas de software, como por exemplo, arquitetura *Mobile*;
- A realização de otimizações nas transformações M2C e metaprogramas desenvolvidos, visando maximizar a geração de código.
- A reaplicação do experimento, descrito no Capitulo 4, em um ambiente industrial; e novos experimentos comparando a geração de código a partir de modelos com outras abordagens que não utilizam modelos;
- A realização de uma análise das diferenças existentes nos códigos implementados pelos grupos (manual e gerado) e a relação com os tempos gastos pelos grupos;
- A pesquisa de novas técnicas de geração de código, não abordadas neste trabalho, para apoiar a reengenharia de sistemas legados e comparação com o trabalho desenvolvido; e
- A sistematização dos testes dos artefatos reutilizáveis construídos ao longo do processo proposto a fim de maximizar a correção de eventuais defeitos existentes.

PUBLICAÇÕES

Durante o desenvolvimento da pesquisa apresentada nesta dissertação, os seguintes artigos científicos foram publicados em colaboração com diferentes pesquisadores:

PAPOTTI, P. E.; PRADO, A. F. do; SOUZA, W. L. de; CIRILO, C. E.; PIRES, L. F. A Quantitative Analysis of Model-Driven Code Generation Through Software Experimentation. In: *Proceedings of the 2013 International Conference on Advanced Information Systems Engineering (CAISE'13)*. Valencia, Spain. 2013 (em fase de publicação).

PAPOTTI, P. E.; PRADO, A. F. do; SOUZA, W. L. de. An approach to support legacy systems reengineering to mdd using metaprogramming. In: *XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*. Medellín, Colombia. 2012. p. 1-10.

PAPOTTI, P. E.; PRADO, A. F. do; SOUZA, W. L. de. Reducing time and effort in legacy systems reengineering to mdd using metaprogramming. In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. New York, NY, USA: ACM, 2012. (RACS'12), p. 348-355. ISBN 978-1-4503-1492-3.

PRADO, A. F. do; PAPOTTI, P. E. Reengenharia e Reutilização de Software. *Revista T.I.S - Tecnologias, Infraestrutura e Software*. ISSN 2316-2872, v. 1, n. 3, 2013 (em fase de publicação).

REFERÊNCIAS BIBLIOGRÁFICAS

ATKINSON, C.; KÜHNE, T. Model-driven development: A metamodeling foundation. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 20, p. 36–41, September 2003. ISSN 0740-7459.

BATORY, D. Multilevel models in model-driven engineering, product lines, and metaprogramming. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 45, p. 527–539, July 2006. ISSN 0018-8670.

BAUER, C.; KING, G. *Java Persistance with Hibernate*. [S.l.]: Dreamtech Press, 2006.

BERGSTEN, H. *JavaServer pages*. [S.l.]: O'Reilly & Associates, Inc., 2003.

BHANOT, V. et al. Using domain-specific modeling to develop software defined radio components and applications. In: *Proceedings of the 5th OOPSLA Workshop on Domain-Specific Modeling*. San Diego, CA, USA: [s.n.], 2005.

BITTAR, T. J. et al. Web communication and interaction modeling using model-driven development. In: *Proceedings of the 27th ACM international conference on Design of communication*. New York, NY, USA: ACM, 2009. (SIGDOC '09), p. 193–198. ISBN 978-1-60558-559-8.

BLOIS, A.; WERNER, C.; BECKER, K. Towards a components grouping technique within a domain engineering process. In: IEEE. *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*. [S.l.], 2005. p. 18–25.

BOEHM, B. Improving software productivity. *Computer*, v. 20, n. 9, p. 43 –57, sept. 1987. ISSN 0018-9162.

BOHLEN, M. *AndroMDA*. 2003. Acesso em: 8 fev 2012. Disponível em: <http://www.andromda.org/>.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. [S.l.]: Addison-Wesley Professional, 2005. ISBN 0321267974.

BOOCOCK, P. *Jamda: The Java Model Driven Architecture*. 2003. Acesso em: 8 fev 2012. Disponível em: <http://sourceforge.net/projects/jamda/>.

BOSCH, J. Software architecture: The next step. *Software architecture*, Springer, p. 194–199, 2004.

BOSCH, J.; BOSCH-SIJTSEMA, P. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, Elsevier, v. 83, n. 1, p. 67–76, 2010.

BUDINSKY, F. et al. Automatic code generation from design patterns. *IBM Systems Journal*, IBM, v. 35, n. 2, p. 151–171, 1996.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. Past, present, and future trends in software patterns. *Software, IEEE*, IEEE, v. 24, n. 4, p. 31–37, 2007.

CASTILLO, R. del; GARCÍA-RODRÍGUEZ, I.; CABALLERO, I. Preciso: a reengineering process and a tool for database modernisation through web services. In: ACM. *Proceedings of the 2009 ACM symposium on Applied Computing*. [S.l.], 2009. p. 2126–2133.

CERNY, T.; SONG, E. A profile approach to using uml models for rich form generation. In: IEEE. *Information Science and Applications (ICISA), 2010 International Conference on*. [S.l.], 2010. p. 1–8.

CHEN, L. et al. Reengineering of java legacy system based on aspect-oriented programming. In: IEEE. *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*. [S.l.], 2010. v. 3, p. 220–223.

CHIKOFSKY, E.; CROSS, J. et al. Reverse engineering and design recovery: A taxonomy. *Software, IEEE*, IEEE, v. 7, n. 1, p. 13–17, jan. 1990. ISSN 0740-7459.

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.

CIRILO, C. et al. Experimentation of the model driven richubi process in the adaptive rich interfaces development. In: IEEE. *Software Engineering (SBES), 2011 25th Brazilian Symposium on*. [S.l.], 2011. p. 184–193.

CONRADI, R. et al. A pragmatic documents standard for an experience library: Roles, documen, contents and structure. UM Computer Science Department; CS-TR-4235, 2001.

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.

CORREIA, R. et al. Architecture migration driven by code categorization. *Software Architecture*, Springer, p. 115–122, 2007.

CROSS, J. K.; SCHMIDT, D. C. Meta-programming techniques for distributed real-time and embedded systems. In: *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*. Washington, DC, USA: IEEE Computer Society, 2002. (WORDS '02), p. 3–.

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: LOPES, C. et al. (Ed.). *Workshop on Aspects and Dimensions of Concerns (ECOOP 2000)*. [S.l.: s.n.], 2000.

CZARNECKI, K.; HELSEN, S. Classification of model transformation approaches. In: *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*. [S.l.: s.n.], 2003.

CZARNECKI, K.; HELSEN, S. Feature-based survey of model transformation approaches. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 45, p. 621–645, July 2006. ISSN 0018-8670.

DAMAŠEVIČIUS, R.; ŠTUIKYS, V. Taxonomy of the fundamental concepts of metaprogramming. *Information Technology and Control*, Citeseer, v. 37, n. 2, p. 124–132, 2008.

DEURSEN, A. van; KLINT, P. Little languages: little maintenance. *Journal of Software Maintenance*, John Wiley & Sons, Inc., New York, NY, USA, v. 10, p. 75–92, March 1998. ISSN 1040-550X.

DUVALL, P.; MATYAS, S.; GLOVER, A. *Continuous integration: improving software quality and reducing risk*. [S.l.]: Addison-Wesley Professional, 2007.

EFFTINGE, S.; KADURA, C. *OpenArchitectureWare 4.1 Xpand Language Reference*. 2006. Acesso em: 12 mar 2013. Disponível em: <http://www.openarchitectureware.org/pub/documentation/4.1/r20_xPandReference.pdf>.

FISCHER, G.; NAKAKOJI, K.; YE, Y. Metadesign: Guidelines for supporting domain experts in software development. *Software, IEEE*, v. 26, n. 5, p. 37 –44, sept.-oct. 2009. ISSN 0740-7459.

FIX, G. The design of an automated unit test code generation system. In: *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*. [S.l.: s.n.], 2009. p. 743 –747.

FORMAN, I. R.; FORMAN, N. *Java Reflection in Action (In Action series)*. Greenwich, CT, USA: Manning Publications Co., 2004. ISBN 1932394184.

FRAKES, W.; KANG, K. Software reuse research: Status and future. *Software Engineering, IEEE Transactions on*, IEEE, v. 31, n. 7, p. 529–536, 2005.

FREDERICK, G.; BOND, P.; TILLEY, S. Vulcan: A tool for automatically generating code from design patterns. In: *Systems Conference, 2008 2nd Annual IEEE*. [S.l.: s.n.], 2008. p. 1 –4.

FUHR, A. et al. Model-driven software migration into service-oriented architectures. *Computer Science-Research and Development*, Springer, p. 1–20, 2011.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GOMAA, H. *Designing software product lines with UML*. [S.l.]: Addison-Wesley Boston, USA;, 2004.

GOMES, J.; MONTEIRO, M. Design pattern implementation in object teams. In: ACM. *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.], 2010. p. 2119–2120.

GRISS, M.; FAVARO, J.; D'ALESSANDRO, M. Integrating feature modeling with the rseb. In: IEEE. *Software Reuse, 1998. Proceedings. Fifth International Conference on*. [S.l.], 1998. p. 76–85.

HEIJSTEK, W.; CHAUDRON, M. Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*. [S.l.: s.n.], 2009. p. 113 –120. ISSN 1089-6503.

HUNOLD, S. et al. Pattern-based refactoring of legacy software systems. *Enterprise Information Systems*, Springer, p. 78–89, 2009.

HUNTER, J.; CRAWFORD, W. *Java servlet programming*. [S.l.]: O'Reilly Media, 2001.

HUTCHINSON, J.; ROUNCEFIELD, M.; WHITTLE, J. Model-driven engineering practices in industry. In: ACM. *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. [S.l.], 2011. p. 633–642.

IZQUIERDO, J. C.; MOLINA, J. G. Extracting models from source code in software modernization. *Software and Systems Modeling*, Springer, p. 1–22, 2012.

JACOBSON, I.; LINDSTRÖM, F. Reengineering of old systems to an object-oriented architecture. In: *Conference proceedings on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 1991. (OOPSLA '91), p. 340–350. ISBN 0-201-55417-8.

JARGAS, A. *Shell Script Profissional*. [S.l.]: Novatec Editora, 2008.

KELLER, W. Mapping objects to tables. In: CITESEER. *Proceedings of Second European Conference on Pattern Languages of Programming (EuroPLoP'97). Siemens Technical Report*. [S.l.], 1997. v. 120.

KLEPPE, A. G.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 032119442X.

KRASNER, G.; POPE, S. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, v. 1, n. 3, p. 26–49, 1988.

KROISS, C.; KOCH, N.; KNAPP, A. Uwe4jsf: A model-driven generation approach for web applications. *Web Engineering*, Springer, p. 493–496, 2009.

LUCRÉDIO, D. *Uma abordagem orientada a modelos para reutilização de software*. Tese (Doutorado), Instituto de Ciências Matemáticas e de Computação, São Carlos, SP, 2009. Disponível em: <http://www2.dc.ufscar.br/ daniel/files/teseDoutoradoDanielLucredio.pdf>.

LUCRÉDIO, D. et al. Mvcase tool - working with design patterns. In: *Proceedings of the Third Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2003)*. [S.l.: s.n.], 2003.

MACDONALD, S. et al. Generative design patterns. In: IEEE. *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*. [S.l.], 2002. p. 23–34.

MALENFANT, J.; JACQUES, M.; DEMERS, F.-N. A tutorial on behavioral reflection and its implementation. In: *Proceedings of the Reflection 96 Conference*. San Francisco, CA: [s.n.], 1996. p. 1–20.

MENS, T.; GORP, P. V. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 152, p. 125–142, 2006.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 37, p. 316–344, December 2005. ISSN 0360-0300.

MILLER, J.; MUKERJI, J. *MDA Guide Version 1.0.1*. [S.l.], June 2003. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>.

MOHAGHEGHI, P. et al. Models in software engineering. In: CHAUDRON, M. R. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. MDE Adoption in Industry: Challenges and Success Criteria, p. 54–59. ISBN 978-3-642-01647-9.

MONTGOMERY, D. *Design and analysis of experiments*. [S.l.]: Wiley, 2008.

MUSSET, J. et al. *ACCELEO User Guide*. 2006. Acesso em: 12 mar 2013. Disponível em: <http://www.acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf>.

OLDEVIK, J. *MOFScript user guide Version 0.9 (MOFScript v 1.4.0)*. 2011. Disponível em: <https://eclipse.org/gmt/mofscript/doc/MOFScript-User-Guide-0.9.pdf>.

OMG. *MOF 2 XMI Mapping Specification*. 2011. Acesso em: 12 mar 2013. Disponível em: <http://www.omg.org/spec/XMI/>.

PALMER, Z.; SMITH, S. Backstage java: making a difference in metaprogramming. In: ACM. *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications*. [S.l.], 2011. p. 939–958.

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

PAPOTTI, P. E.; PRADO, A. F. do; SOUZA, W. L. de. Reducing time and effort in legacy systems reengineering to mdd using metaprogramming. In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. New York, NY, USA: ACM, 2012. (RACS '12), p. 348–355. ISBN 978-1-4503-1492-3.

PAPOTTI, P. E. et al. A quantitative analysis of model-driven code generation through software experimentation. In: *Proceedings of the 2013 International Conference on Advanced Information Systems Engineering (CAISE'13)*. [S.l.: s.n.], 2013.

PARADA, A.; SIEGERT, E.; BRISOLARA, L. de. Generating java code from uml class and sequence diagrams. In: *Computing System Engineering (SBESC), 2011 Brazilian Symposium on*. [S.l.: s.n.], 2011. p. 99 –101.

PARTSCH, H.; STEINBRÜGGEN, R. Program transformation systems. *ACM Computing Surveys (CSUR)*, ACM, v. 15, n. 3, p. 199–236, 1983.

PEÑA, Y.; CORREAL, D.; HERNANDEZ, T. Reusing legacy systems in a service-oriented architecture: a model-based analysis. *Advances in Conceptual Modeling–Applications and Challenges*, Springer, p. 86–95, 2010.

PENTEADO, R. A. D. *Um Método para Engenharia Reversa Orientada a Objetos*. Tese (Doutorado) — Universidade de São Paulo. 251p, Instituto Física de São Carlos, SP, 1996.

PFLEEGER, S.; ATLEE, J. *Software engineering*. 3ed. ed. [S.l.]: Prentice Hall, 2005.

PRADO, A. F. do; PAPOTTI, P. E. Reengenharia e reutilização de software. *Revista T.I.S - Tecnologias, Infraestrutura e Software*, v. 1, n. 3, p. 1–10, 2013. ISSN 2316-2872.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 6. ed. New York, NY, USA: McGraw-Hill, Inc., 2005. ISBN 0077227808, 9780077227807.

PRIETO-DIAZ, R.; ARANGO, G. *Domain analysis and software systems modeling*. [S.l.]: IEEE Computer Society, 1991.

PUTNAM, L. H.; MYERS, W. *Measures for Excellence*. [S.l.]: Yourdon Press, 1992.

QUINTERO, J. et al. How face the top mde adoption problems. In: *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*. [S.l.: s.n.], 2012. p. 1 –10.

REEUWIJK, C. van. Rapid and robust compiler construction using template-based metacompilation. In: *Proceedings of the 12th international conference on Compiler construction*. Berlin, Heidelberg: Springer-Verlag, 2003. (CC'03), p. 247–261. ISBN 3-540-00904-3.

RICCA, F. et al. Using unimod for maintenance tasks: An experimental assessment in the context of model driven development. In: *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*. [S.l.: s.n.], 2012. p. 77 –83. ISSN 2156-788.

RODRÍGUEZ-ECHEVERRÍA, R. et al. Modernization of legacy web applications into rich internet applications. *Current Trends in Web Engineering*, Springer, p. 236–250, 2012.

ROSS, D. T. Structured analysis (sa): A language for communicating ideas. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 3, p. 16–34, January 1977. ISSN 0098-5589.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. *Unified Modeling Language Reference Manual, The (2nd Edition)*. [S.l.]: Pearson Higher Education, 2004. ISBN 0321245628.

SCHMIDT, D. Guest editor's introduction: Model-driven engineering. *Computer*, v. 39, n. 2, p. 25–31, feb. 2006. ISSN 0018-9162.

SENDALL, S.; KOZACZYNSKI, W. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 20, p. 42–45, September 2003. ISSN 0740-7459.

SHAPIRO, S.; WILK, M. An analysis of variance test for normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965.

SHEARD, T. Accomplishments and research challenges in meta-programming. In: *Proceedings of the 2nd international conference on Semantics, applications, and implementation of program generation*. Berlin, Heidelberg: Springer-Verlag, 2001. (SAIG'01), p. 2–44. ISBN 3-540-42558-6.

SNEED, H. Migrating from cobol to java. In: *Software Maintenance (ICSM), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010. p. 1 –7. ISSN 1063-6773.

ŠTUIKYS, V.; MONTVILAS, M.; DAMAŠEVIČIUS, R. Development of web component generators using one-stage metaprogramming. *Information Technology and Control*, v. 38, n. 2, p. 108–118, 2009.

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

TRUJILLO, S.; AZANZA, M.; DIAZ, O. Generative metaprogramming. In: ACM. *Proceedings of the 6th international conference on Generative programming and component engineering*. [S.l.], 2007. p. 105–114.

UMAR, A.; ZORDAN, A. Reengineering for service oriented architectures: A strategic decision model for integration versus migration. *Journal of Systems and Software*, Elsevier, v. 82, n. 3, p. 448–462, 2009.

VOELTER, M. A catalog of patterns for program generation. In: *Eighth European Conference on Pattern Languages of Programs, Irsee, Germany*. [S.l.: s.n.], 2003.

VÖELTER, M. *MD* Best Practices Version 1.1*. dec 2008. Acesso em: 9 fev 2012. Disponível em: <http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf>.

VÖELTER, M.; GROHER, I. Product line implementation using aspect-oriented and model-driven software development. In: *Proceedings of the 11th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2007. p. 233–242. ISBN 0-7695-2888-0.

VOGEL, L. *Java Emitter Template (JET) - Tutorial*. 2009. Acesso em: 12 mar 2013. Disponível em: <http://www.vogella.de/articles/EclipseJET/article.html>.

WANG, H.; SHEN, B.; CHEN, C. Model-driven reengineering of database. In: IEEE. *Software Engineering, 2009. WCSE'09. WRI World Congress on*. [S.l.], 2009. v. 3, p. 113–117.

WOHLIN, C. et al. *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN 0-7923-8682-5.

YODER, J. W.; JOHNSON, R. E.; WILSON, Q. D. Connecting Business Objects to Relational Databases. In: *Conference on the Pattern Languages of Programs*. St.Louis, Missouri, EUA: [s.n.], 1998.

Apêndice 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: _____
RA: _____
Idade: _____

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

Unified Modeling Language (UML)	0	1	2	3	4
Linguagem de programação JAVA	0	1	2	3	4
IDE Eclipse	0	1	2	3	4
Banco de Dados MySQL	0	1	2	3	4
Java Persistence API (JPA)	0	1	2	3	4
Framework Hibernate	0	1	2	3	4
Linguagem de Marcação HTML/XHTML	0	1	2	3	4
Framework Java Server Faces (JSF)	0	1	2	3	4
Folhas de Estilos (CSS)	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) Projeto de Software com uso de padrões

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

c) Desenvolvimento de Software para Web

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

d) Desenvolvimento Dirigido a Modelos

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

Apêndice B

FORMULÁRIO DE CONSENTIMENTO

Experimento

Este experimento visa avaliar a aplicação do *Processo Dirigido a Modelos para Geração de Código* para apoio à construção de software através de geração de código.

Idade

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

Procedimento

Este experimento ocorrerá em uma única sessão, que incluirá o desenvolvimento de aplicação Web com as funcionalidades CRUD a partir de um modelo representado por um diagrama de classes da UML. O desenvolvimento da aplicação se dará com e sem a aplicação do *Processo Dirigido a Modelos para Geração de Código*, 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

Paulo Eduardo Papotti
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): _____ **RA:** _____

Assinatura: _____

Data: ____/_____/_____

Apêndice C

DESCRIÇÃO DA TAREFA – CICLO DE VIDA CLÁSSICO

Instruções

Este experimento avaliará a aplicação do *Processo Dirigido a Modelos para Geração de Código* para apoio à construção de software através de geração de código. A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Contextualização

O desenvolvimento de software é uma atividade em constantes mudanças ao longo do tempo e novas propostas de se realizar essa atividade têm sido pesquisadas a cada dia. Métodos ágeis de desenvolvimento de software têm ganhado espaço no mercado de trabalho, motivando o surgimento de novos desafios na Engenharia de Software. Diante desses desafios, têm-se a busca por técnicas e ferramentas que viabilizam a agilidade proposta por esses métodos.

Visando solucionar essa busca, um processo dirigido a modelos para realizar geração de código foi pesquisado. O processo tem como base o desenvolvimento dirigido a modelos, que podem ser utilizados como ponto de partida para especificar os requisitos do sistema. Com auxilio de transformações e metaprogramação, realiza-se a geração de grande parte do código, proporcionando economia de tempo, recursos e liberando os desenvolvedores para atuar em tarefas mais importantes da construção do sistema.

Tarefa

Você recebeu a descrição de uma aplicação Web que realiza as funcionalidades CRUD (Create, Retrieve, Update e Delete) a partir das entidades de um diagrama de classes da UML.

Sua tarefa é desenvolver uma aplicação que realiza as funcionalidades CRUD a partir das entidades de um diagrama de classes da UML seguindo as disciplinas convencionais de Análise, Projeto, Implementação e Testes da Engenharia de Software.

Utilize o material de apoio para auxiliá-lo durante a construção da aplicação. Tendo que os requisitos da aplicação já foram identificados e o projeto da aplicação já foi realizado,

inicie a execução do processo a partir da disciplina Implementação. Para cada atividade realizada anote os dados temporais no formulário apropriado.

Ao final do experimento, compacte o projeto da aplicação desenvolvida e envie por email para paulo.papotti@dc.ufscar.br, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] – Experimento Classico"*.

Apêndice D

DESCRIÇÃO DA TAREFA – PROCESSO PROPOSTO

Instruções

Este experimento avaliará a aplicação do *Processo Dirigido a Modelos para Geração de Código*para apoio à construção de software através de geração de código. A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Contextualização

O desenvolvimento de software é uma atividade em constantes mudanças ao longo do tempo e novas propostas de se realizar essa atividade têm sido pesquisadas a cada dia. Métodos ágeis de desenvolvimento de software têm ganhado espaço no mercado de trabalho, motivando o surgimento de novos desafios na Engenharia de Software. Diante desses desafios, têm-se a busca por técnicas e ferramentas que viabilizam a agilidade proposta por esses métodos.

Visando solucionar essa busca, um processo dirigido a modelos para realizar geração de código foi pesquisado. O processo tem como base o desenvolvimento dirigido a modelos, que podem ser utilizados como ponto de partida para especificar os requisitos do sistema. Com auxilio de transformações e metaprogramação, realiza-se a geração de grande parte do código, proporcionando economia de tempo, recursos e liberando os desenvolvedores para atuar em tarefas mais importantes da construção do sistema.

Tarefa

Você recebeu a descrição de uma aplicação Web que realiza as funcionalidades CRUD (Create, Retrieve, Update e Delete) a partir das entidades de um diagrama de classes da UML.

Sua tarefa é desenvolver uma aplicação que realiza as funcionalidades CRUD a partir das entidades de um diagrama de classes da UML seguindo a etapa de Engenharia de Aplicação do *Processo Dirigido a Modelos para Geração de Código*.

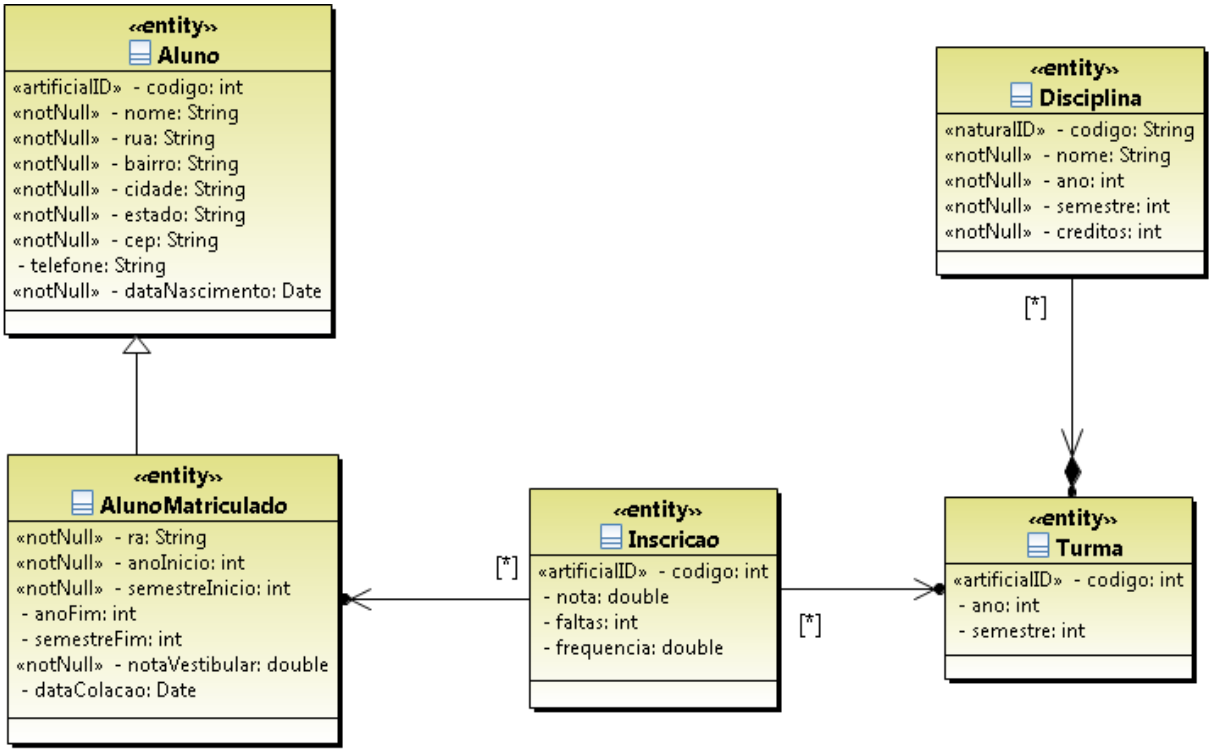
Utilize o material de apoio para auxiliá-lo na execução das atividades. Tendo que os requisitos da aplicação já foram identificados e o projeto da aplicação já foi realizado, inicie a execução do processo a partir da atividade *Implementar*. Para cada atividade realizada anote os dados temporais e de linhas de código no formulário apropriado.

Ao final do experimento, compacte o projeto da aplicação desenvolvida e envie por email para paulo.papotti@dc.ufscar.br, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] – Experimento - Processo"*.

Apêndice E

ROTEIRO DA APLICAÇÃO – CICLO DE VIDA CLÁSSICO

Você deve criar uma aplicação que realiza as funcionalidades CRUD de todas as entidades especificadas pelo diagrama de classes abaixo.



1) Marque o tempo de início da atividade “Criação do Banco de Dados” no formulário de Coleta de Dados.

2) Crie uma nova database dentro do MySQL utilizando o comando:

create database experimento;

3) Marque o tempo de fim da atividade “Criação do Banco de Dados” no formulário de Coleta de Dados.

4) Marque o tempo de início da atividade “Criação do Projeto Web” no formulário de Coleta de Dados.

5) Agora, vamos criar um novo projeto no Eclipse para criar nossa aplicação Web. Na

tela principal do Eclipse vá em File > New > Other... Selecione Dynamic Web Project.

6) Digite o nome do Projeto. Vamos dar o nome de **Classico.**

7) Selecione o Servidor **Apache Tomcat** como Target Runtime. Se não houver essa opção você deve instala-lo em seu computador.

8) Na seção Configuration clique em Modify para alterar a Configuração do Servido Tomcat.

9) Adicione os modulos **JavaServer Faces** versão 2.0 e **JPA** versão 2.0 ao projeto. Clique em **OK**.

10) Clique em **Next** e em seguida **Next** novamente.

11) Selecione a biblioteca **Hibernate** para ser usada como biblioteca de persistência e clique em **Next**.

12) Clique em **Next** na próxima janela.

13) Selecione a biblioteca **JSF** e clique em **Finish**.

14) Uma vez criado o projeto, vamos fazer as últimas configurações. Clique com o botão direito do mouse sobre o projeto **Classico** e selecione **Properties**.

15) Acesse o menu **Java Build Path** a esquerda.

16) Clique em **Add Library**...

17) Selecione **User Library** e clique em **Next**.

18) Selecione **MySQL** e clique em **Finish**

19) Acesse o menu **Deployment Assembly** a esquerda.

20) Caso apareça uma janela perguntando se você deseja salvar as alterações realizadas, clique em **Apply**.

21) Depois de acessar o menu **Deployment Assembly**, clique em **Add.**

22) Na próxima janela selecione **Java Build Path Entries** e clique em **Next**.

23) Na próxima janela selecione **MySQL** e clique em **Finish**.

24) Acesse o menu **JPA** a esquerda.

25) Selecione a opção **Discover annotated classes automatically** e clique em **OK**.

26) Abra o arquivo persistence.xml dentro da pasta src/META-INF e preencha-o com seus dados (usuário e senha) de acesso ao banco de dados MySQL como abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
<persistence-unit name="Classico" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
```

```
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost:3306/experimento"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="javax.persistence.jdbc.driver"
                value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
    </persistence>
```

27) Dentro da pasta src do projeto Classico crie 4 pacotes: modelo, fachada, converters e beansgerenciados. Para criar um novo pacote dentro da pasta src, basta clicar com o botão direito do mouse sobre a pasta src e ir em New > Other.. > Package.

28) A estrutura do Projeto deve ficar como mostra a figura abaixo:

29) Marque o tempo de fim da atividade “Criação do Projeto Web” no formulário de Coleta de Dados.

30) Marque o tempo de início da atividade “Criação das classes referentes ao modelo”.

31) Agora, você deve criar as classes em Java que representam o diagrama de classes. Para isso, devemos criar 5 classes dentro do pacote modelo: Aluno, AlunoMatriculado, Inscricao, Turma e Disciplina. Todas devem conter os atributos especificados no diagrama e as anotações para realizar a persistência em banco de dados.

32) Marque o tempo de fim da atividade “Criação das classes referentes ao modelo”.

33) Marque o tempo de início da atividade “Criação dos beans de sessão no padrão Fachada para acesso a banco de dados”.

34) Agora, você deve criar as classes (beans de sessão) no padrão Fachada para acesso das classes AlunoMatriculado, Inscricao, Turma eDisciplina ao banco de Dados. Crie 5 novas classes dentro do pacote fachada, sendo: AbstractFacade, AlunoMatriculadoFacade, InscricaoFacade, TurmaFacade e DisciplinaFacade.

35) Marque o tempo de fim da atividade “Criação dos beans de sessão no padrão Fachada para acesso a banco de dados”.

36) Marque o tempo de início da atividade “Criação de beans gerenciados”.

37) Agora, você deve criar os Beans Gerenciados que vão receber as requisições das páginas Web da aplicação. Crie 5 beans gerenciados dentro do pacote beansgerenciados: AlunoManagedBean, AlunoMatriculadoManagedBean, InscricaoManagedBean, TurmaManagedBean e DisciplinaManagedBean.

38) Marque o tempo de fim da atividade “Criação de beans gerenciados”.

39) Marque o tempo de início da atividade “Criação de conversores”.

40) Agora, você deve criar as classes de conversão para as classes AlunoMatriculado, Turma e Disciplina. Isso é necessário, pois na interface Web haverá combobox’s para selecionar essas classes. Essa classe converter facilita a conversão do texto

escrito no combobox para o objeto referete aquele texto. Desta forma, você deve criar a classe AlunoMatriculadoConverter, TurmaConverter e DisciplinaConverter dentro do pacote converters.

41) Marque o tempo de fim da atividade “Criação de conversores”.

42) Marque o tempo de início da atividade “Criação de páginas Web”.

43) Agora você deve criar as páginas Web utilizando o framework JSF. Para cada uma das classes do pacote modelo (AlunoMatriculado, Inscricao, Turma e Disciplina) deverão ser criadas 3 páginas Web (cadastrar, editar e listar) e uma página index (página inicial).

44) Dentro da pasta WebContent crie 4 novas pastas: AlunoMatriculado, Inscricao, Turma e Disciplina. Para criar uma nova pasta dentro da pasta WebContent, clique com o botão direito do mouse em cima da pasta WebContent, vá em New > Folder. Digite o nome da nova pasta e clique em OK.

45) Criada as 4 pastas, vamos criar as páginas Web de AlunoMatriculado, Inscricao, Turma e Disciplina. Para criar uma nova página Web vá em New > Other.. E procure por JSP File.

46) Crie as páginas newAlunoMatricualo.jsp, editAlunoMatriculado.jsp e listAlunoMatriculado.jsp.

47) Crie as páginas newInscricao.jsp, editInscricao.jsp e listInscricao.jsp.

48) Crie as páginas newTurma.jsp, editTurma.jsp e listTurma.jsp.

49) Crie as páginas newDisciplina.jsp, editDisciplina.jsp e listDisciplina.jsp.

50) Crie as página Web de index dentro da pasta WebContent.

51) Apenas se julgar necessário, crie regras de navegação no arquivo faces-config.xml que está na pasta WebContent/WEB-INF para realizar o redirecionamento das páginas de sua aplicação.

52) Marque o tempo de fim da atividade “Criação de páginas Web”.

53) Marque o tempo de início da atividade “Testes”.

54) Executando o projeto Web:

a. Clique com o botão direito sobre o projeto e selecione Run > Run on Server.

b. Clique em Finish para executar a aplicação

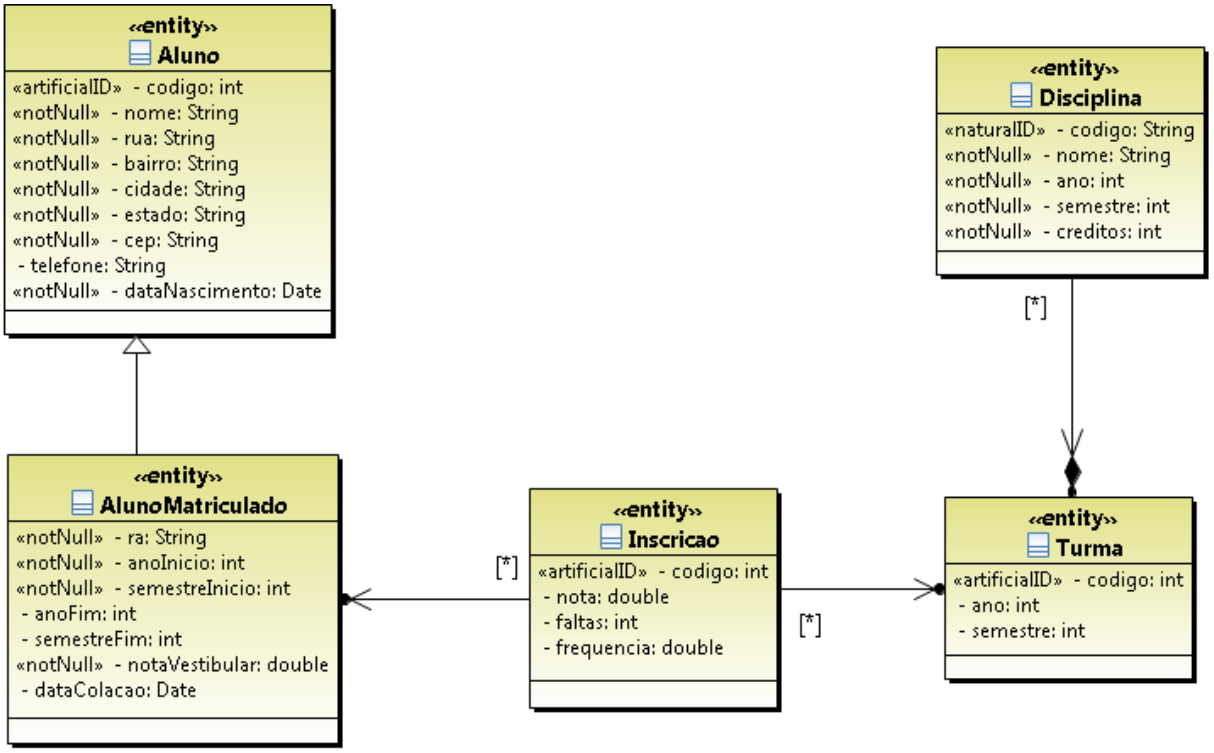
c. Após iniciar a aplicação adicione “/faces/index.jsp” na url para chamar a página inicial da aplicação.

55) Marque o tempo de fim da atividade “Testes”.

Apêndice F

ROTEIRO DA APLICAÇÃO – PROCESSO PROPOSTO

Você deve criar uma aplicação que realiza as funcionalidades CRUD de todas as entidades especificadas pelo diagrama de classes abaixo.



1) Marque o tempo de início da atividade "Criação do Banco de Dados" no formulário de Coleta de Dados.

2) Crie uma nova database dentro do MySQL utilizando o comando:

create database experimento;

3) Marque o tempo de fim da atividade "Criação do Banco de Dados" no formulário de Coleta de Dados.

4) Marque o tempo de início da atividade "Criação do Projeto Web" no formulário de Coleta de Dados.

5) Agora, vamos criar um novo projeto no Eclipse para criar nossa aplicaação Web. Na tela principal do Eclipse vá em File > New > Other... Selecione Dynamic Web Project.

6) Digite o nome do Projeto. Vamos dar o nome de **Processo.**

7) Selecione o Servidor **Apache Tomcat** como Target Runtime.

8) Na seção Configuration clique em Modify para alterar a Configuração do Servido Tomcat.

9) Adicione os modulos **JavaServer Faces** versão 2.0 e **JPA** versão 2.0 ao projeto. Clique em **OK**.

10) Clique em **Next** e em seguida **Next** novamente.

11) Selecione a biblioteca **Hibernate** para ser usada como biblioteca de persistência e clique em **Next**.

12) Clique em **Next** na próxima janela.

13) Selecione a biblioteca **JSF** e clique em **Finish**.

14) Uma vez criado o projeto, vamos fazer as últimas configurações. Clique com o botão direito do mouse sobre o projeto **Processo** e selecione **Properties**.

15) Acesse o menu **Java Build Path** a esquerda.

16) Clique em **Add Library**...

17) Selecione **User Library** e clique em **Next**.

18) Selecione **MySQL** e clique em **Finish**

19) Acesse o menu **Deployment Assembly** a esquerda.

20) Caso apareça uma janela perguntando se você deseja salvar as alterações realizadas, clique em **Apply**.

21) Depois de acessar o menu **Deployment Assembly**, clique em **Add.**

22) Na próxima janela selecione **Java Build Path Entries** e clique em **Next**.

23) Na próxima janela selecione **MySQL** e clique em **Finish**.

24) Acesse o menu **JPA** a esquerda.

25) Selecione a opção **Discover annotated classes automatically** e clique em **OK**.

26) Marque o tempo de fim da atividade "Criação do Projeto Web" no formulário de Coleta de Dados.

27) Marque o tempo de início da atividade "Geração das classes referentes ao modelo".

28) Execute o gerador do processo proposto através do comando:

29) java -jar TransformationMetaprogram.jar

30) Executando a Transformação M2C - "UML to Java":

- **a.** Selecione o modelo de entrada (model.uml)
- **b.** Selecione o diretório do projeto Web criado no Eclipse
- **c.** Clique no botão "Execute M2C Trasformation"

d. Caso a transformação M2C execute com sucesso, a seguinte mensagem deve aparecer "Transformation executed succesfully! Now, compile your classes and exexcute a metaprogram."

31) Compilando as classes do modelo que foram geradas:

a. Abra a IDE Eclipse

b. Clique sobre o ícone do projeto Web e pressione F5 para atualizar as estruturas e compilar as classes geradas.

32) Marque o tempo de fim da atividade "Criação das classes referentes ao modelo".

33) Marque o tempo de inicio da atividade "Geração dos beans de sessão no padrão Fachada para acesso a banco de dados, beans gerenciados, conversores e páginas Web".

34) Executando o metaprograma para a geração do CRUD:

a. Na tela do gerador, preencha os campos da seção Metaprogram com os dados para acesso ao banco de dados do seu computador.

b. Clique no botão "Execute Metaprogram"

c. Caso o metaprograma execute com sucesso, a seguinte mensagem deve aparecer "CRUD generated successfully!".

35) Volte a IDE Eclipse e pressione F5 após clicar sobre o ícone do projeto Web para atualizar as estruturas e compilar o código gerado.

36) Marque o tempo de fim da atividade "Geração dos beans de sessão no padrão Fachada para acesso a banco de dados, beans gerenciados, conversores e páginas Web".

37) Marque o tempo de início da atividade "Testes".

38) Executando o projeto Web:

a. Clique com o botão direito sobre o projeto e selecione Run > Run on Server.

b. Clique em Finish para executar a aplicação

c. Após iniciar a aplicação adicione "/faces/index.jsp" na url para chamar a página inicial da aplicação

39) Marque o tempo de fim da atividade "Testes".

Apêndice G

FORMULÁRIO DE COLETA DE DADOS – CICLO DE VIDA CLÁSSICO

Grupo nº: ____

Anote na tabela abaixo os horários de início e fim de cada atividade realizada.

Atividade	Hora início	Hora Fim
Criação do Banco de Dados	___:___h	___:___h
Criação do Projeto Web	___:___h	___:___h
Criação das classes referentes ao modelo	___:___h	___:___h
Criação dos beans de sessão no padrão Fachada para acesso a banco de dados	___:___h	___:___h
Criação de beans gerenciados	___:___h	___:___h
Criação de conversores	___:___h	___:___h
Criação de páginas Web	___:___h	___:___h
Testes	___:___h	___:___h

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	Hora de Identificação	Hora de Resolução
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

Apêndice H

FORMULÁRIO DE COLETA DE DADOS – PROCESSO PROPOSTO

Grupo nº: ____

Anote na tabela abaixo os horários de início e fim de cada atividade realizada.

Atividade	**Hora início**	**Hora Fim**
Criação do Banco de Dados	___:___h	___:___h
Criação do Projeto Web	___:___h	___:___h
Geração das classes referentes ao modelo	___:___h	___:___h
Geração dos beans de sessão no padrão Fachada para acesso a banco de dados, beans gerenciados, conversores e páginas Web	___:___h	___:___h
Testes	___:___h	___:___h

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	**Hora de Identificação**	**Hora de Resolução**
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

	___:___h	___:___h
	___:___h	___:___h

Apêndice I

FORMULÁRIO DE AVALIAÇÃO – CICLO DE VIDA CLÁSSICO

Grupo nº: ____ RA: _____

1) Você utilizou algum mecanismo e/ou ferramenta realizar geração de código para a aplicação? Esse mecanismo/ferramenta atendeu satisfatoriamente suas necessidades?

_____

_____

_____

_____

2) Você sentiu dificuldades no desenvolvimento da aplicação? Especifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

_____

3) Você considera que um processo, baseado em modelos e geração de código para o desenvolvimento de software, em geral, pode contribuir tornar mais fácil o desenvolvimento de software? Justifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

_____

4) Você acha que é viável, sob o ponto de vista de esforço de desenvolvimento, construir aplicações sem a geração de funcionalidades (por exemplo CRUD)? Justifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

_____

Obrigado por sua colaboração!

Apêndice J

FORMULÁRIO DE AVALIAÇÃO – PROCESSO PROPOSTO

Grupo nº: ____ RA: _____

1) Você considera que a aplicação do *Processo Dirigido a Modelos para Geração de Código* auxiliou no desenvolvimento da aplicação? Justifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

2) Você encontrou alguma dificuldade no desenvolvimento da aplicação com o uso do processo? Especifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

3) Quais sugestões você teria para melhorar o processo?

_____

_____

_____

4) Você acredita que o uso de geradores por meio de Transformações Modelo para Código e Metaprogramas contribuíram para o desenvolvimento da aplicação? Justifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

5) Você considera viável, sob o ponto de vista de esforço de desenvolvimento e eficiência a aplicação do processo proposto para outros tipos de sistemas (Arquitetura Mobile, Serviços e etc)? Justifique.

☐ Sim ☐ Não ☐ Parcialmente

_____

_____

_____

6) Selecione entre os itens abaixo quais são, em sua opinião, os maiores pontos positivos na utilização do processo proposto:

☐ Geração de grande parte do código de uma aplicação.

☐ O código gerado faz uso de padrões de software, facilitando a implementação de futuras funcionalidades.

☐ Desenvolver uma nova aplicação com maior foco na modelagem, ao invés da codificação.

☐ Facilidade de realizar manutenções na estrutura da aplicação.

☐ Outro: _____

7) Em uma escala de 0 a 10, em que 0 é nada satisfeito e 10 é totalmente satisfeito, qual foi seu grau de satisfação na aplicação do processo proposto comparado ao ciclo de vida clássico?

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10

8) Você considera viável a utilização do processo proposto para o desenvolvimento de outras aplicações diferentes da aplicação realizada no experimento?

☐ Sim ☐ Não

Obrigado por sua colaboração!