
**UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ**

***CAMPUS* LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS**

**CIÊNCIA DA COMPUTAÇÃO**

**LUAN DE SOUZA MELO**

# UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS HIBERNATE BASEADO EM MDD E POA

Bandeirantes

2016

**LUAN DE SOUZA MELO**

# UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS HIBERNATE BASEADO EM MDD E POA

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. André Luís Andrade Menolli

Bandeirantes
2016

**LUAN DE SOUZA MELO**

# UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS HIBERNATE BASEADO EM MDD E POA

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

**COMISSÃO EXAMINADORA**

_____
Prof. Dr. André Luís Andrade Menolli
UENP –*Campus* Luiz Meneghel

_____
Prof. Me. GlaucoCarlos Silva
UENP – *Campus* Luiz Meneghel

_____
Prof. Dr. Maurício Massaru Arimoto
UENP – *Campus* Luiz Meneghel

Bandeirantes, 12 de dezembro de 2016

Dedico este trabalho aos meus pais Edilson e Kátia, sem os quais eu não teria tido forças de realizá-lo.

# AGRADECIMENTOS

Agradeço primeiramente a Deus, por ser essencial em minha vida, autor de meu destino e ser meu guia.

Aos meus pais Edilson e Kátia, que me apoiaram e estavam comigo em todos os momentos que precisava de apoio.

Aos meus avós Saul e Juraci, por torcerem e sempre intercedendo a Deus sobre minha vida e nos meus estudos.

Aos meus tios Julielson, Patrícia, Rosilene, Valdirene e Fábio, que sempre me motivaram e me deram forças para que eu pudesse seguir meus sonhos e atingir meus objetivos.

Ao meu tio Heitor, que no começo da graduação me amparou em sua casa, tornando mais fácil essa jornada.

À minha prima e amiga Bárbara, que esteve comigo em todos os anos, sempre desejando que eu conseguisse realizar esta etapa da minha vida.

Ao meu orientador Menolli, por ter me inserido na iniciação científica, desde o início de minha graduação fazendo que eu procure buscar novos conhecimento, e forçar-me a aprender mais sobre vários assuntos.

Ao professor Merlin, que me fez perceber que temos que caminhar com as próprias pernas, e que nem sempre teremos os professores para nos ajudar fora da graduação.

Aos meus amigos, que estiveram comigo todos esses anos, em especial a Letícia e a Andressa, no qual as considero como irmãs.

Aqueles que são loucos
suficiente para achar
que podem mudar o
mundo, são afinal os
que têm alguma chance
em fazê-lo.
*(Steve Jobs*

# RESUMO

Ao longo dos anos, novas abordagens,visando a melhoria da qualidade do produto final, vêm sendo propostas para desenvolvimento de software. Dentre as principais abordagens existentes, destaca-se a programação orientada a aspectos (POA), que pode ser vista como uma evolução da programação orientada a objetos (POO). A POA visa principalmente melhorar a separação de interesses, que afeta diretamente na coesão e no acoplamento de software. Outra abordagem significativa abordada no trabalho, é o desenvolvimento dirigido a modelos (MDD), que possibilita a geração de códigos de forma automática por meio de modelos. Esta abordagem visa manter a coerência entre o modelo e o código gerado, e agilizar o processo de desenvolvimento de software. Sendo assim, com o intuito de utilizar estas abordagens em um domínio conhecido, é proposta uma abordagem e um framework para gerar códigos de persistência de dados *Hibernate*, baseado no MDD e na POA. Espera-se que com o uso da ferramenta, mesmo programadoresinexperientes possam criar aplicações de persistência de dados de forma produtiva e de qualidade. Para isto, foi aplicado um estudo experimental, que procurou auxiliar a utilização do framework para implementar uma parte de um sistema de uma biblioteca, utilizando a análise quantitativa dos dados. Os resultados obtidos através do experimento demonstram que a abordagem proposta é viável para tanto para redução do tempo de desenvolvimento do software quanto para melhoria da qualidade do código em comparação com a implementação tradicional, que usa programação orientada a objetos.



**Palavras-chave:** Desenvolvimento dirigido a modelos. Programação orientada a aspectos. Persistência de dados.*Hibernate.*

# ABSTRACT

Over the years, new approaches that aim to make the final product one with more quality, are being proposed for software development. Of the main proposed approaches, stands out the aspect-oriented programming, which is an evolution of object-oriented programming, which mainly aims to improve the separation of interest, which directly affects the cohesion and the software coupling. Another significant approach addressed in the paper, is the models-directed development, which purpose is to automatically generate codes through models. This approach aims to facilitate the coherence between the model and the generated code, and streamline the software development process. Therefore, in order to use these approaches in a notorious domain, a framework approach to generating Hibernate data persistence codes based on model-driven development and on aspect-oriented programming is proposed. It is expected that with the use of the tool, even inexperienced programmers, may be able to create data persistence applications with quality and in a productive way.



**Keywords:** Model-driven development. Aspect-oriented programming. Data persistence. Hibernate.

# LISTA DE FIGURAS

# LISTA DE QUADROS E TABELAS

# LISTA DE SIGLAS

| | |
|---|---|
| ATL | *ATLAS Transformation Language* |
| CA | Acoplamento Aferente |
| *CDC* | Difusão do interesse sobre os componentes |
| CDLOC | Difusão do interesse sobre as linhas de códigos |
| CDO | Difusão do interesse sobre as operações |
| CE | Acoplamento Eferente |
| DAO | *Data Access Object* |
| DIT | Profundidade da árvore de herança |
| DSLs | Linguagem de Domínio Especifico |
| EMF | *Eclipse-Modelling Framework* |
| IDE | *Integrated Development Environment* |
| JDBC | *Java Database Connectivity* |
| JPA | *Java Persistence API* |
| LCOO | Falta de coesão nas operações |
| LOC | Linhas de código |
| MDA | *Model-Driven Architecture* |
| MDD | *Model-Driven Development* |
| MDE | *Model-Driven Engineering* |
| MOF | *Model To Text* |
| NOA | Número de atributos |
| OMG | *Object Management Group* |
| OO | Orientado a objetos |
| ORM | *Object-Relational Mapping* |
| PIMs | *Plataform-Independent Models* |
| POA | Programação Orientada a Aspecto |
| POO | Programação Orientado a Objetos |
| QVT | *Query-View Transformation Language* |
| SGBDR | Sistema Gerenciador de Banco de Dados Relacional |
| SQL | *Structured Query Language* |
| SysML | *Systems Modeling Language* |

| | |
|---|---|
| UML | *Unified Modeling Language* |
| WOC | Comprimento das operações por componentes |
| XSLT | *Extensible Stylesheet Language Transformation* |

# SUMÁRIO

# 1. INTRODUÇÃO

Neste capítulo introdutório é apresentado a contextualização e problematização do projeto, a justificativa, os objetivos e a organização do trabalho.

## 1.1 CONTEXTUALIZAÇÃO E PROBLEMATIZAÇÃO

No desenvolvimento de software, uma grande quantidade de aplicações, sejam elasmóvel*, desktop, web* ou qualquer outro tipo, necessita que suasinformações sejam armazenadas de forma persistente. Atualmente,existem um grande número de métodos e ferramentas para este fim. Todavia, esta diversidade de opções não é apropriada, pois, na maioria das vezes transfere a responsabilidade pela escolha do método de persistência de dados aos desenvolvedores,que geralmente não estão aptos para fazer a melhor escolha.

O método de persistência deve ser escolhido ao nível do projeto, pois o projetista deve decidir qual a melhor maneira de se implementar a persistência para uma determinada situação. Entretanto, muitas vezes isto não ocorre, ademais, muitas equipes de desenvolvimento não tem uma separação clara entre os papéis de programador e projetista de software. Sendo assim, a escolha do método de persistência acaba sendo realizada por um método padrão no qual a empresa adota, ou seja, escolhe-se um método padrão que a empresa adota ou, uma técnica de domínio do responsável pela escolha. Porém, existem algumas desvantagens ao escolher desta maneira, tais como:escolha pelo padrão da empresa, em que pode não ser a melhor para um projeto ou não ser o domínio dos programadores; ou escolha individual, em que pode dificultar a manutenção e reusabilidade do software.

Contudo, há diversos estudos relacionados a este tema, e tentam definir quais técnicas e ferramentas são apropriadas para cada tipo de situação. Dentre estes estudos, pode-se destacar o trabalho de Pothu (2012) que realizou uma análise comparativa dos métodos de mapeamento objeto relacional para Java, bem como o trabalho de Barcia *et. al.* (2008) que realizou uma revisão sobre as principais tecnologias de persistência de dados.

Os estudos citados anteriormente descrevem em quais casos diferentes técnicas ou métodos são mais adequados. Porém,a escolha da técnica ou métodos,

é apenas um passo para implementação da persistência de dados, pois, os desenvolvedores precisam ter um domínio sobre estas diferentes técnicas e métodos para que se possa implementá-las. Uma alternativa para tratar esta adversidade éutilizar o desenvolvimento dirigido a modelos (MDD, do inglês *Model Driven Development*). O MDD viabiliza desenvolver códigos a partir de modelos de alto nível, e sua utilidade é basicamente transformar um modelo de software em código executável, seja integralmente ou parcialmente, diversificando de acordo com as necessidades.

O uso do MDD auxilia os programadores e todos os envolvidos no desenvolvimento de software, em queos códigos são gerados automaticamente, sem que haja uma necessidade de serem produzidos totalmente de forma manual. O MDD auxilia na padronização e na utilização de códigos, pois a implementação dos códigos deve adotar um modelo de alto nível, facilitando assim a manutenção de software e minimização de erros, além de agilizar o processo de entrega do produto final. Segundo Jiang, Zhang e Miyake (2007), o MDD é uma evolução da engenharia de software, no que contribui para sanar um grande problema no desenvolvimento de software, a incompatibilidade entre o modelo e código criado.

Desta maneira, umaferramenta baseada em MDD, que propicie no desenvolvimento, códigos de persistência de dados, pode contribuir para evitar problemas recorrentes neste tipo de aplicação, além de mitigar contrapontos comuns no desenvolvimento de software.

Outras abordagens da engenharia de software podem estar correlacionadas no desenvolvimento de aplicações de persistência de dados, por exemplo programação orientada a aspecto (POA). Segundo Soares *et. al.* (2002) a POA surgiu para solucionar um problema da programação orientada a objetos (POO) em capturar algumas decisões de projeto que um sistema deve implementar, no qual torna a distribuição do código em diversos pressupostos. Este entrelaçamento e espalhamento torna-se difícil para o processo de desenvolvimento e manutenção destes sistemas. Com isto, a POA aumenta a modularidade, pois separa o código que implementam funções específicas, afetando diferentes partes do sistema, chamadas preocupações ortogonais (*crosscutting concern*) (SOARES *et. al.* 2002).

A POA é constituída por interesses, que são requisitos, uma propriedade ou uma funcionalidade do sistema. Estes interesses podem ser implementados tanto

por classes, em que agrupam e modularizam uma funcionalidade, e por aspectos, que implementam códigos que estariam dispersos em um sistema OO e que afetaria o comportamento do sistema, os quais são denominados interesses transversais. Assim, propõe-se utilizar o POA em conjunto com MDD para criar aplicações de persistência de dados que sejam condizentes com as melhores práticas de engenharia de software.

## 1.2 OBJETIVOS

Nesta seção é apresentado o objetivo geral e específico desta presente pesquisa.

### 1.2.1 Objetivo Geral

Este trabalho visa propor um framework de geração de códigos de persistência de dadospara a linguagem Java, integrado ao ambiente de desenvolvimento Eclipse[1]. O objetivo é auxiliar os desenvolvedores a criar aplicações de persistência de dados, Hibernate, mais adequadamente aos objetivos da aplicação, por meio do desenvolvimento orientado a modelos e programação orientada a aspecto.

### 1.2.2 Objetivos Específicos

Como objetivos específicos podem ser destacados:

- Desenvolvimento de uma abordagem adequada para o framework;
- Implementar o framework;
- Elaborar experimento para testar a abordagem;
- Análise quantitativa dos dados obtidos por meio dos experimentos.

## 1.3 JUSTIFICATIVA

*Desenvolvimento dirigido a modelos*(MDD, do inglês *Model Driven Development*) é uma área da engenharia de software cujo objetivo é aumentar o nível de abstração das aplicações e, consequentemente, simplificar e formalizar as

[1]http://www.eclipse.org

tarefas e as fases do ciclo de vida de um software, utilizando modelos e tecnologias de modelagem (HAILPERN; TARR, 2006). Segundo Vidyapettham (2009), devido sua flexibilidade e aplicabilidade, o MDD tem progredido significativamente.

Os desenvolvedores ao utilizar MDD são capazes de, simultaneamente, projetar soluções e construir artefatos nos quais farão parte do produto final (VIDYAPEETHAM, 2009). Selic (2008)afirma que a base elementar do MDD não são programas de computadores e sim modelos. Além disso, a vantagem do MDD é a expressão de modelos usando conceitos que são ligados a linguagem de baixo nível (SELIC, 2008).

Muitos desenvolvedores têm utilizado o MDD para criar aplicações em menos tempo, simples e confiáveis. Como exemplo, Heitkötter, Majchrzak e Kuchen (2013), que mostrou uma linguagem de domínio específico (DSLs) para desenvolvimento de dispositivos móveis multiplataformas, denominada MD2, no qual produz aplicações nativas para sistemas Android e iOS. Outro exemplo, é o estudo de Steiner et. al. (2002), que traz uma DSL para desenvolvimento de aplicações integradas à nuvem.

A persistência de dados é um exemplo de aplicação que é possível deve ser solucionado por meio de MDD. Implementações deste domínio são comuns a muitas aplicações, porém, na maioria das vezes são implementadas de maneira errada.

Na literatura podem-se encontrar vários padrões de persistência de dados. Como exemplo, no trabalho de Nock (2003) é descrito um conjunto de padrões de acesso a dados que apresentam estratégias para desacoplar os componentes de acesso a dados de outras partes da aplicação. A existência destes padrões indica a possibilidade de criar meta-modelos de referências para facilitar uma implementação satisfatória dos mesmos.

Dentre os padrões de persistência existentes, um dos mais utilizados é o *Data Access Object* (DAO), que permite aos objetos de negócio utilizarem a fonte de dados sem ter o conhecimento dos detalhes específicos de sua implementação. Tal padrão fornece um acesso transparente a suas funcionalidades, uma vez que os detalhes de implementação da fonte de dados são encapsulados dentro do objeto de acesso a dados (SUN MICROSYSTEMS, 2002).

No entanto, apesar de ser um padrão consolidado, o DAO apresenta algumas limitações que podem ser atenuadas utilizando POA. No trabalho de Oliveira, Menolli e Coelho (2008), foi aplicado POA no padrão DAO, o que proporcionou o encapsulamento das funções de acesso a dados da aplicação desenvolvida. uma vez que os objetos de negócio e outros componentes da aplicação não necessitam acessar as funções de acesso a dados da aplicação.Com isto, todo o controle de chamadas a operações de acesso e persistência de dados é efetuado pelos aspectos. Sendo assim, os demais objetos da aplicação não mais necessitam conhecer os detalhes de implementação das classes e métodos responsáveis pelas funções de acesso a dados da aplicação, otimizando a modularidade e a legibilidade da aplicação.

Alémdos padrões de persistência de dados,os programadores de software podem utilizar diferentes tecnologias para desenvolver códigos persistência de dados, tais como *Java Persistence API*(JPA); *Structured Query Language* (SQL); e *EclipseLink* utilizando a técnica de mapeamento objeto relacional, ou seja, uma técnica traduz informações contidas em objetos em informações que possam ser entendidas por um banco de dados, auxiliandoo desenvolvedor a lidar com modelos diferentes: modelo relacional e modelo orientado a objetos.

Com isto, a aplicação de POA no padrão DAO é apenas um exemplo de como padrões de acesso a dados podem ter suas implementações aprimoradas, tratando o interesse transversal de persistência. Estes interesses transversais de acordo com Soares *et. al.* (2002), consistem em funcionalidades responsáveis pelo armazenamento persistente de informações manipuladas pelo sistema. Essas funcionalidades correspondem ao controle de conexões, gerenciamento de transações, carregamento parcial e cache de objeto para melhorar o desempenho e a sincronização de estados de objetos com as entidades de banco de dados correspondentes para assegurar a consistência. Além disso, verificou-se em Soares *et al.* (2002), que o interesse de persistência pode ser dividido em vários sub-interesses, como classes de acesso a dados, controle de conexões, gerenciamento de transações e customizações de classes de acesso a dados.

Entretanto,o uso, e principalmente a aplicação de POA para refinar esses padrões não é uma tarefa simples. Sendo assim, umframeworkbaseado em MDD que auxilie o desenvolvedor a implementar o padrão DAO, para persistir dados, com

uso da tecnologia *Hibernate*proporcionará um melhor desempenho e produtividade do programador.

## 1.4 ORGANIZAÇÃO DO TRABALHO

O Capítulo 1, aqui apresentado, visa contextualizar o leitor, sobre do que se trata o trabalho; justificar e exibir o objetivo geral e os específicos.

No Capítulo 2 é apresentado a fundamentação teórica que contém os seguintes conceitos: Desenvolvimento Dirigido a modelos; Programação Orientada a Aspectos; *Hibernate*; AspectJ; e Acceleo.

No Capítulo 3 é apontado um posicionamento metodológico, bem como define a estruturação detalhada da pesquisa, com suas propostas iniciais baseado na investigação bibliográfica.

No capítulo 4 é apresentado a abordagem proposta, bem como a visão geral da mesma, o meta-modelo e o gerador de códigos.

No capítulo 5 é apresentado os detalhes do experimento, tais como: as definições das hipóteses; seleção dos participantes; descrição do experimento e da análise; e como ele foi realizado.

No capítulo 6 é apresentado tanto uma análise quanto uma discussão sobre os resultados obtidos no experimento, baseado nas métricas de software, para medir a qualidade das implementações do *Hibernate* e JDBC utilizando a abordagem.

No capítulo 7 é concluídoeste trabalho, apresentando as considerações finais, descrevendo a relevância do estudo e as contribuições da pesquisa.Este capítulo também expõe trabalhos futuros de pesquisa para utilização da abordagem.

Por fim os apêndices contidos neste documento, apresentam:

- Apêndice A e B apresentam os artefatos para geração do framework, tais como: o meta-modelo e *templates* respectivamente.
- Apêndice C apresenta o termo de consentimento e esclarecimento do experimento realizado.
- Apêndice D até o G apresentam os artefatos utilizados para

realização do experimento

- Apêndice H apresenta os artefatos gerados pelo framework, ou seja, o código de persistência de dados.

# 2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas introduções aos conceitos de *Model-Driven Development,* Programação Orientada a Modelo, Hibernate, Acceleo e AspectJ.

## 2.1 MODEL-DRIVEN DEVELOPMENT

Desde a última década, *Model-Driven Development,* ou MDD, evoluiu significantemente devido a sua flexibilidade e aplicabilidade (VIDYAPEETHAM, 2009). MDD ou engenharia dirigida a modelos (MDE, do inglês *Model-DrivenEngineering*) (SHMIDT, 2006), é uma abordagem da engenharia de software em que os modelos, em vez de programas, são as saídas principais do processo de desenvolvimento (KENT 2002; SCHMIDT, 2006). Segundo Hailpern (2009), o objetivo do MDD é aumentar o nível de abstração de uma aplicação e, consequentemente, simplificar e formalizar as tarefas e as fases do ciclo de vida de um software usando modelos e tecnologias de modelagem.

Fernandes, Machado e Carvalho (2008), apontam que o MDD implica no aumento da abstração de linguagens de programação de alto nível para linguagens de modelagem.

De acordo com Singh e Sood (2009), a principal diferença entre MDD quando comparada com desenvolvimento de software tradicionais refere-se à exploração de modelos como base para gerar código fonte de baixo nível.

Vidyapeetham (2009) destaca que o MDD compreende o uso de modelos no ciclo de vida em desenvolvimento de software e argumenta que o MDD automatiza o desenvolvimento de software via processamento de modelos, transformação de modelos e técnicas de geração de código.

Como a ideia do MDD é representar um domínio ou problema por meio de modelos, Mizuno, Matsumoto e Mori (2010), apresentam o MDD como uma abordagem capaz de lidar satisfatoriamente com variações de lógicas de negócios e tecnologias de baixo nível. Além disso, os autores também afirmam que o MDD não somente reduz o custo de desenvolvimento, mas também conflitos entre o *design* da aplicação e implementações por meio da adaptação de modelos.

Além disso, Heitkötter, Majchrzak e Kuchen (2013) argumentam que o MDD

pode refletir uma aplicação por meio de modelos, e sobre estes modelos, um é capaz de realizar transformações de forma automáticas,afim de gerar o código fonte de baixo nível. Ou seja, desenvolvedores podem criar aplicativos usando especificações de alto nível e a partir destas especificações, gerar um produto de trabalho final. Assim, o MDD pode não trazer uma implementação específica de uma aplicação, mas uma abstração de alto nível destas aplicações, levando a um rápido ciclo de desenvolvimento (HEITKÖTTER*et. al.*, 2013).

Convencionalmente, de acordo com Vidyapeetham (2009), desenvolvedores empregam duas abordagens diferentes em desenvolvimento de software:

1. Projetar a solução de forma visual; e
2. Codificar solução baseando-se em requisitos funcionais.

Apesar deambas as abordagens poderem apresentar vantagens e desvantagens, oMDD tende asera melhor solução. Por meio do MDD, desenvolvedores são capazes não somente de projetarem uma solução, mas também de construir artefatos parciais que farão parte do produto final (VIDYAPEETHAM, 2009).

Stahl e Voelter (2016) elegeram as principais metas do MDD como:

1. Aumentar a rapidez do desenvolvimento através da automação: código-fonte de baixo nível podem ser gerados a partir de modelos via transformações de modelos.
2. Melhorar a qualidade de software, especialmente porque a arquitetura de software deverá repetir-seuniformemente uma vez que foi definido em um modelo e gerado por transformações.
3. Aspectos de aplicações transversais podem ser especificadas nas regras de transformações,que também se aplica para resolver erros nas saídas dos códigos. Separação de Interesses não leva somente para um melhor manutenibilidade de software por meio da anulação da redundância, como também para o melhor gerenciamento de mudanças tecnológicas.
4. Modelos e tecnologias podem ser adaptados em uma linha de

produção para construção de um software, conduzindo para um maior nível de reusabilidade.

5. Através da abstração, melhorar a capacidade de gerenciamento da complexidade, como modelos permite aos desenvolvedores programar em um maior nível de abstração.

Além disso, Fernandes, Machado e Carvalho (2007), apontam as principais contribuições de MDD como sendo:

1. Produtividade: a produtividade do MDD pode ser alcançado através de dois fatores: a curto prazo e a longo prazo. Produtividade a curto prazo é obtida sobre o quanto a funcionalidade de um artefato de software pode fornecer. A longo prazo a produtividade é obtida através do aumento da longevidade em direção dos artefatos de software.
2. Conceitos de desenvolvimento mais próximo do domínio: o MDD utiliza conceitos menos relacionados com a própria tecnologia, e mais próximo do domínio real do problema, no qual permite que software sejam produzidos por pessoas que podem não ser especialistas no domínio.
3. Automação: as transformações automáticas de projetos de alto nível em códigos de baixo nível e a facilidade de uso na gestão de modelos, que são também menos propensas e erros.
4. Captura de Conhecimento Específico e Reutilização: funções de mapeamento que transmitem dados das transformações de modelos podem resultar na captura de conhecimento específico. Permite a reutilização de tal conhecimento, quando acontece uma mudança em uma aplicação ou em uma implementação particular, bem como permitir que uma evolução independente de modelos,pode não levar a modelos que apresentam uma maior longevidade.

Além disso, Selic (2008) salienta,como a principal vantagem da adoção do MDD, a expressão de modelos usando conceitos de que não são estritamente associados com linguagens de baixo nível, apesar de estar perto do domínio real do problema.

## 2.2 PROGRAMAÇÃO ORIENTADA A ASPECTOS

O uso do paradigma orientado a objetos (OO) no desenvolvimento de software teve um grande aumento devido à necessidade de se desenvolver software de qualidade, buscando maiores níveis de reuso e manutenibilidade, aumentando assim a produtividade no desenvolvimento e o suporte a mudanças de requisitos (MEYER, 1997).No entanto, apesar da OO ser o paradigma de programação mais popular, as abstrações são reconhecidas como sendo incapazes de capturar todas as preocupações de interesse no sistema de software (KICZALES, 1997).

Segundo Soares e Borba (2002), para solucionar a limitação da POO em capturar algumas decisões de projeto que um sistema deve implementar, causando a distribuição e espalhamento de códigos com diferentes propósitos, surgiu a programação orientada a aspectos (POA). Com isto, há uma dificuldade em se desenvolver e dar manutenção a estes sistemas, devido ao entrelaçamento e o espalhamento de códigos.

Oconceito de aspecto veio da ideia da abstração básica da POO e que considera classes, objetos, métodos e atributos, e que não são suficientes para separar algumas preocupações, especialmente em sistemas complexos. Separação de interesses é um princípio bem estabelecido em engenharia de software, e um interesse é parte do problema que visa tratar como uma única unidade conceitual (TARR, P. et al., 2009). Interesses são modularizados (Figura 2.1), ao longo do desenvolvimento de software usando diferentes abstrações providas por linguagens, métodos e ferramentas (CHAVES; GARCIA, 2003). Estes interesses são chamados de interesses ortogonais (*crosscutting* concerns*)* uma vez que naturalmente atravessam o modularidade de outros interesses (SANT'ANNA *et al.,* 2003). A falta de meios adequados para separação e modularização, interesses ortogonais tende a ser dispersos e confusos com outros interesses. Como consequência, pode-se encontrar outros problemas, tais como: a compreensão dos códigos, e também, uma baixa reutilização de artefatos de software.

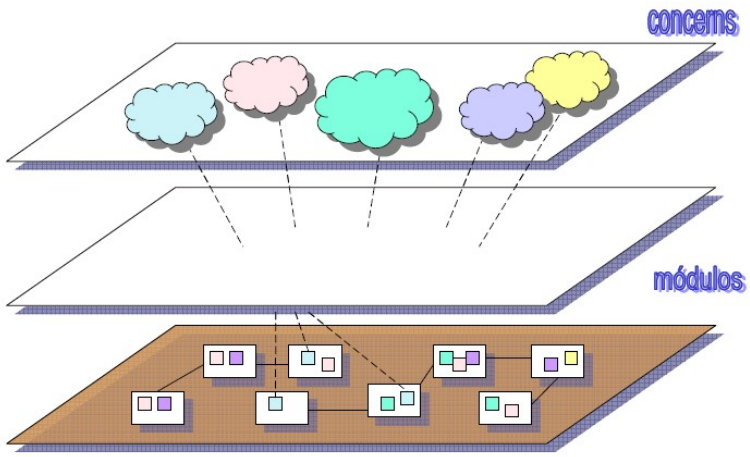
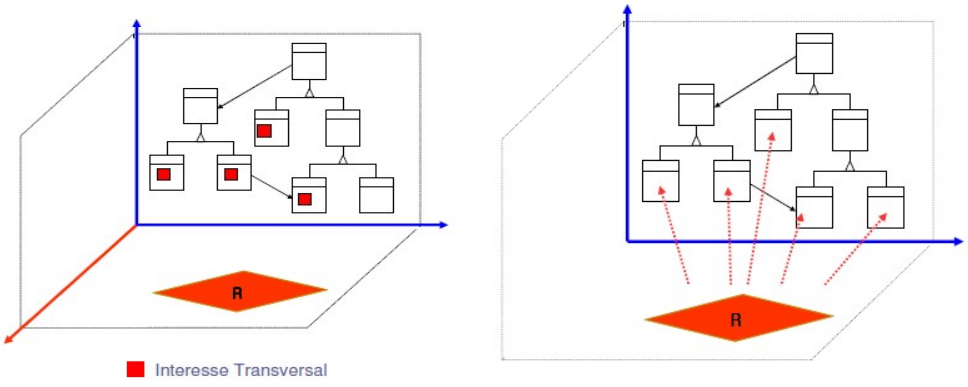



Figura 2.1- Separação de interesses com POA. (FONTE: Chavez & Garcia, 2003)

POA foi proposta como uma técnica para melhorar a separação de interesses na construção de software OO e melhorar a usabilidade e a facilidade de manutenção (KICZALES, 1997). Como mostrado naFigura 2.2, POO fornece uma separação de interesses ortogonais na modularização usando aspecto como uma abstração e fornece um mecanismo para compor aspectos e componentes (classes, métodos, etc.) em específicos pontos de junção. Por exemplo, a Figura 2.2ilustra diferentes maneiras de combinar classes e aspectos, que são possíveis fontes de acoplamento em um sistema orientado a aspecto. A parte (A) representa um sistema OO, em que os interesses ortogonais são dispersos e confusos com outros interesses da classe. A parte (B) mostra como esses interesses ortogonais são removidos das classes e inseridos no aspecto. O ponto de junção representado por flechas, indica o lugar que o aspecto irá operar na classe.




Parte A - POO

Parte B - POA

Figura 2.2 - Comparação de interesses ortogonais em POO e POA (CHAVES; KULESZA, 2003)

### 2.2.1 Composição de um sistema orientado a aspectos

Piveta (2001), afirma que um sistema que utiliza a POA é composto pelos seguintes componentes:

- **Linguagem de componentes:** deve permitir que o programador implemente as funcionalidades básicas de um sistema; não preveem nada a respeito do que deve ser implementado na linguagem de aspectos. Exemplos: Java, C++, C#, PHP.
- **Linguagem de aspectos:** deve suportar a implementação das propriedades desejadas de forma clara e concisa, fornecendo construções necessárias para que o desenvolvedor crie estruturas que descrevam o comportamento dos aspectos e definam em que situações elas ocorrem.
- **Combinador de aspectos:** o combinador de aspectos (*aspectwaver*) tem a função de unir os programas escritos em linguagens de componentes com os escritos em linguagem de aspectos.

### 2.2.2 AspectJ

A linguagem AspectJ, uma extensão orientada a aspectos, de propósito geral, para linguagem Java (GOSLING *et. al.,* 2000). Abaixo são mostrados alguns conceitos básicos para desenvolver códigos para AspectJ e que são utilizados a POA.

- **Pontos de junção** (*join points*): um ponto de junção é qualquer ponto identificável pelo AspectJ durante a execução de um programa, como por exemplo, uma chamada de método, ou execução de método. É aplicado em métodos (chamadas ou execução), construtor (chamadas ou execução de métodos), execução de tratamento de exceções e atributos (consulta, modificações de valores) (CHAVES; GARCIA, 2003).

- **Pontos de corte** (*pointcuts*): pontos de corte são construções de programa que permitem a seleção de um ou mais pontos de junção. Têm como objetivo, a criação de regras gerais para definição de

eventos que serão os pontos de junção.

- **Advice:** *Advice* é um trecho de código a ser executado quando um ponto de corte for atingido (CHAVES e GARCIA, 2003). São compostos pelos pontos de atuação, no qual selecionam-se apenas os pontos de junção.

- **Declarações Inter tipos** (*inter-type declarations*): Aspectos podem declarar membros (campos, métodos e construtores) que pertencem a outros tipos. Estes são chamados de membros *inter-type*. Podem também declarar novas interfaces de outros tipos ou estender uma classe.

- **Aspectos:** o aspecto é a unidade modular principal em AspectJ, em que a classe é a unidade principal em Java. Um aspecto agrupa pontos de corte, *advice* e Inter tipos.

Sendo assim, na Figura 2.3pode ilustrar a importância da POA nesta linguagem, no qual a Figura é parte de um diagrama de classes de um editor de imagens. As classes *Line* e *Point* desse editor deveriam conter apenas código para implementação de linha e ponto, mas como pode se observar, o interesse em destaque é a atualização da imagem (*Display Update*), ou seja, toda vez que uma linha ou ponto é criado ou modificado, é necessário que se atualize a imagem, e este interesse está espalhado pelas classes *Line* e *Point*.

Com isto, nesta figura pode-se perceber que os conceitos de modularização e encapsulamento são quebrados, pois as classes*Point* e *Line* atendem interesses externos.

Para resolver este problema basta colocar este interesse transversal em um aspecto, como pode ser visto na Figura 2.4. Utilizando esta solução, o interesse transversal ficaria modularizado em um aspecto, e as classes *Line* e *Point*, só teriam código para implementação de Linha e do Ponto.

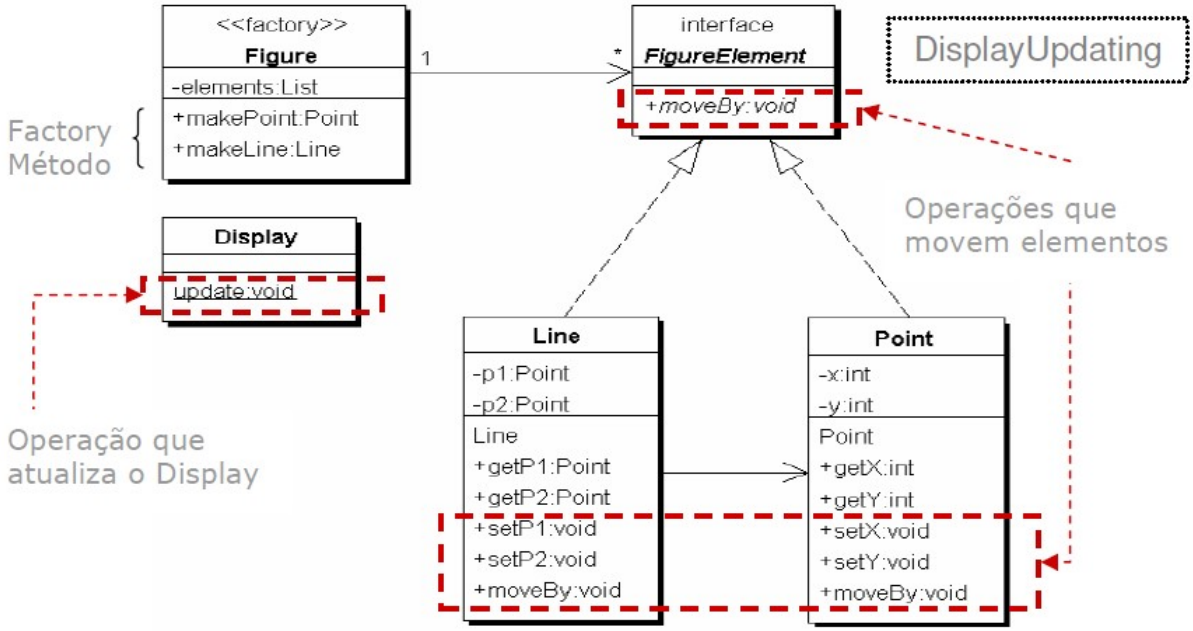



Figura 2.3- Interesses Transversais na atualização da imagem.

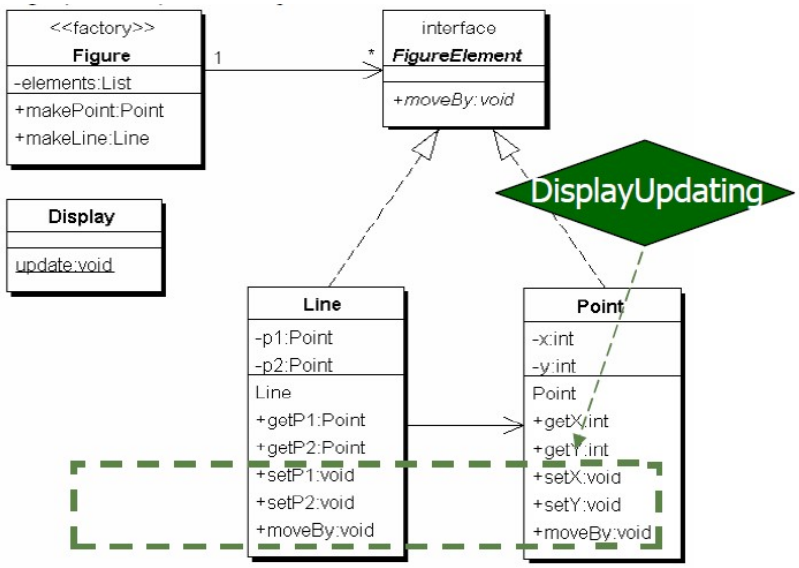



Figura 2.4 - Interesses Transversais na atualização da imagem em um Aspecto.

## 2.3 HIBERNATE

Segundo Konda (2014), existem dois lados do software: um deles é o Java, por exemplo, em que é conhecido somente objetos, enquanto o outro lado é o banco de dados, tal como o banco de dados relacional, no qual os dados são persistidos.

Os desenvolvedores Java sempre trabalham com objetos representando estados e comportamentos de um problema do mundo real. Com isto, a persistência

de objetos é um requisito fundamental ao desenvolver aplicações em Java (KONDA, 2014). Os estados são modelados de modo a ser persistidos em repositórios, em que os dados são armazenados de forma permanente.

Com isto, no momento de armazenar os dados, é necessário ter um banco de dados relacional, em que os dados são representados por linhas e colunas, com relacionamentos e associações (KONDA, 2014). O processo de trazer os dados do Java para o relacional não é uma tarefa simples. Este processo é chamado de *Object-Relational Mapping* (ORM).

Sendo assim, partindo do princípio que quase todas as aplicações precisam de dados persistentes para que possam ser recuperados posteriormente, o gerenciamento desses dados é de fundamental importância para que um sistema de informação esteja funcionando (LUCKOW; MELO, 2015).

*Hibernate*é um *framework* simples, criado para eliminar problemas encontrados no espaço de mapeamento ORM (KONDA, 2014). O seu interesse está relacionado na persistência de dados aplicada ao sistema gerenciador de banco de dados relacional (SGBDR) (Hibernate ORM, 2016).

Segundo Hibernate ORM (2016), os modelos objeto e modelo relacional não trabalham muito bem juntos. Os SGBDRs representam os dados de forma tabular, enquanto que as linguagens orientadas a objeto, como Java, representa os dados como objeto de grafos interligados. Há cinco problemas de incompatibilidade que podem ser expostos usando banco de dados relacional e grafos de objetos (Hibernate ORM, 2016):

- **Granularidade:** Pode acontecer de ter um modelo de objetos com um número maior de classes do que a de tabelas correspondentes no banco de dados.
- **Herança:** Herança é um paradigma natural em linguagens de programação orientado a objetos. Entretanto, SGBDR não define algo semelhante no seu conjunto de funcionalidades.
- **Incompatibilidade de Identidade:** Os objetos em uma aplicação Java, por exemplo, apresentam tanto identidade quanto igualdade.
- **Associações:** Associações são representadas como referencias

unidirecionais em linguagens orientadas objetos, enquanto SGBDR usa a notação de chaves estrangeiras. Se caso precisar de relações bidirecionais em Java, por exemplo, deve-se definir a associação duas vezes.

- **Dados de Navegação:** O caminho para acessar dados em Java é diferente do caminho em banco de dados relacionais. Em Java, navega-se de uma associação para um outro caminho da rede de objetos.

Ressaltando, o *Hibernate* foi desenvolvido para eliminar os problemas encontrados no mapeamento de objeto relacional. Isto acontece, pois, este ORM abstrai o código SQL, toda a camada JDBC (*Java Database Connectivity*),e por fim o SQL é gerado em tempo de execução. Com isto, além de gerar o SQL que servirá para um determinado banco, ele também abre a possiblidade de se alterar o banco de dados sem a necessidade de se alterar o código Java.

Na Figura 2.5, pode ser ilustrado o funcionamento do Hibernate. O mapeamento de atributos de um objeto Java (A), para um banco de dados relacional (C), é realizado através do ORM do Hibernate (B), no qual este é criado através de arquivos XML de configuração.

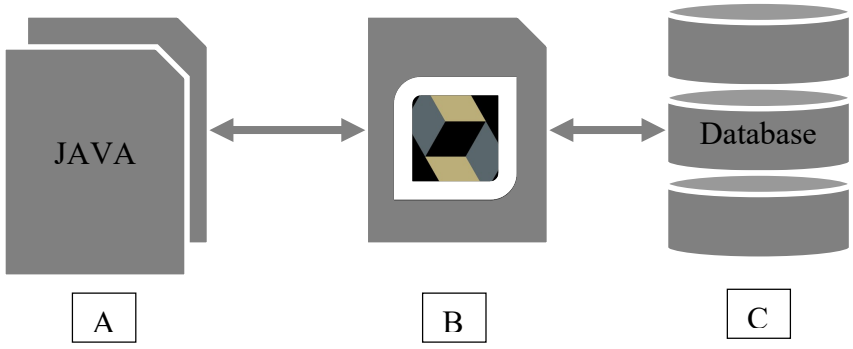



Figura 2.5 - Funcionamento do Mapeamento Objeto-Relacional usando *Hibernate*

## 2.4 ACCELEO

Atualmente, diferentes ferramentas para transformação de modelos orientadas a uma variedade de propósitos têm sido explorados na literatura. Por Exemplo, XSLT (*Extensible Stylesheet Language Transformation*), QVT (*Query-*

*View Transformation Language*), ATL (*ATLAS Transformation Language*) e Acceleo.

Acceleo, criado em 2005,é uma implementação do padrão Model To Text (MOF)dirigido pela *Object Management Group* (OMG) (ECLIPSE, 2016). O Acceleo é uma ferramenta em que foi elaborada para dar suporte ao desenvolvimento de software, no qual utiliza o conceito *de Model-Driven Architecture* (MDA).

O Acceleo, oferece suporte ao desenvolvedor vários recursos que podem ser esperados de uma IDE (*Integrated Development Environment*) de alta qualidade, tais como, sintaxe simples, geração de código eficiente e ferramentas avançadas (ACCELEO, 2016).

O Acceleo foi criado com o objetivo de se ter o melhor conjunto de ferramentas para geração de códigos. Como tal, o Acceleo possui várias características-chavescomo um editor com destaque de sintaxe, detecções de erros e *auto complete* (ACCELEO, 2016). A Quadro 2.1demonstra as funcionalidades da ferramenta Acceleo, e as denotadas por fundo verde estão disponível pelo Acceleo e as denotadas por fundo amarelo não estão disponíveis.

Quadro 2.1- Funcionalidades do Acceleo

| | | | | |
|---|---|---|---|---|
| Destaque de sintaxe | Realização | Dobramento de código | Compilador | Refatoração (modelo extrato) |
| Modelos de código personalizáveis | Abrir declaração | Profiler | Debugger | Refatoração (renomear) |
| Mecanismo de geração | Esboço | Stand Alone runtime | Detecção de erros | Refatoração (consulta de extração) |
| Suporte documentação | Flutuar | Correções rápidas | Rastreabilidade | Refatoração (puxar para cima) |
| Caracteres em branco visíveis | Esboço rápido | Pesquisa ocorrências | Framework de testes | Geração incremental |
| Tipo de hierarquia rápida | Chamada vista hierarquia | Avisos | Suporte Ant | Suporte Maven |
| Ver declaração | Integração Java Avançado | Recipiente classpath | Suporte Checkstyle | Suporte case Camel |
| Análise distribuída | Integração BIRT | Folha de dicas | Geração de documentação | Exportação de perfil de dados gprof |

Fonte: ACCELEO 2016

Desta forma, o Acceleo permite importar diagramas Linguagem de

Modelagem Unificada (UML, do inglês *Unified Modeling Language*) ou *Systems Modeling Language* (SysML), e realizar uma transformação para linguagens de programação, como por exemplo, C#, C++ e Java. Entretanto, o Acceleo não modela os diagramas, fazendo necessário o uso de outra tecnologia para utilizar o Acceleo.

# 3. ESTRUTURAÇÃO DA PESQUISA

Nestecapítuloserá apresentado a caracterização e estruturação da pesquisa.

## 3.1 CARACTERIZAÇÃO DA PESQUISA

Segundo Gil (2002), há uma necessidade de se construir um modelo conceitual e operacional da pesquisa, como forma de comparação entre a visão teórica com os dados da realidade.

Desta forma, esta pesquisa pode ser classificada como um estudo descritivo (GIL, 1994). Descritivo, uma vez que, baseado no conhecimento teórico e nas análises realizadas durante o trabalho, busca-se a compreensão e a descrição de como esta proposta pode auxiliar os projetistas de software a decidir qual a melhor maneira de se implementar a persistência de dados para uma situação especifica.

Com isto, esta pesquisa é quantitativa, buscando verificar se a abordagem proposta é viável, através da coleta de dados, por meio de experimentos, após aplicação da abordagem em alguns alunos do curso de Ciência da Computação da Universidade Estadual do Norte do Paraná.

## 3.2 ESTRUTURAÇÃO DA PESQUISA

Segundo Miguel (2007) pode-se considerar a pesquisa como um planejamento estratégico, em que uma abordagem metodológica compreende diversos níveis de abrangência e profundidade. Algumas decisões são estratégicas, tais como a escolha da abordagem e do método, e outras são mais relacionadas a ordem tática e operacional do trabalho, como por exemplo: procedimentos de conclusão da pesquisa (REINEHR, 2008).

Sendo assim, foi definido uma estrutura de pesquisa a ser seguida para cumprir com os objetivos iniciais, como pode ser visto naFigura 3.1. Basicamente esta estrutura é composta por quatro etapas (Planejamento Inicial, Fase Exploratória, Desenvolvimento, Avaliação e Conclusão), em que estas fases podem possuir uma ou mais etapas que são ligadas entre si através de seta que indicam a direção fluxo e a ação decorrente.

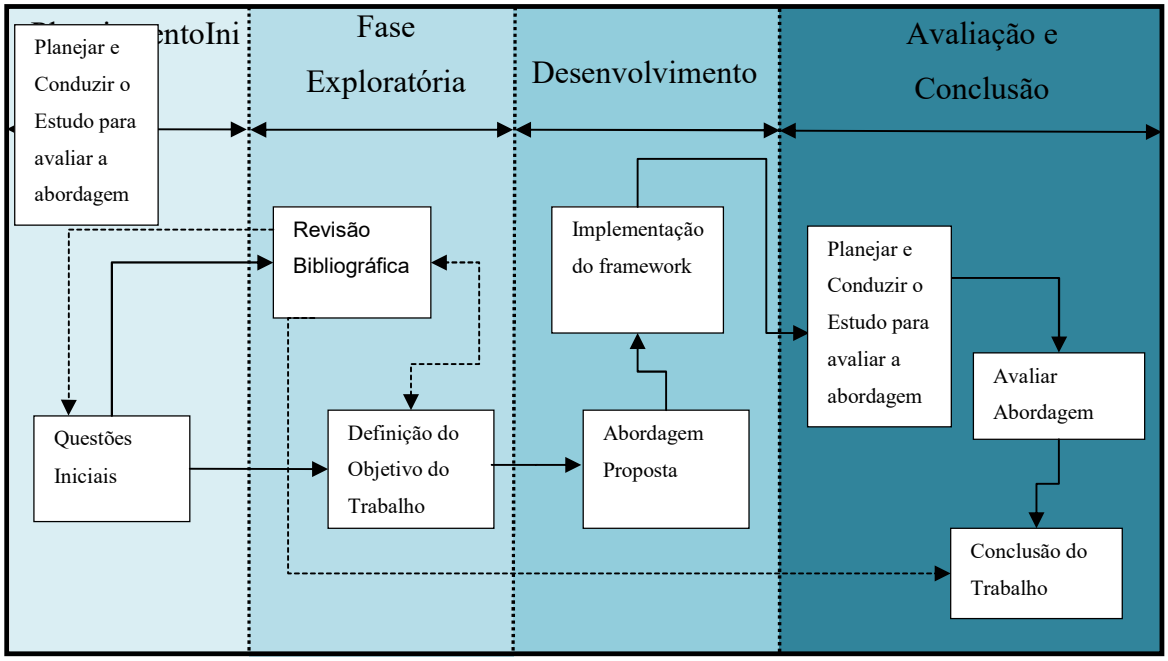



Figura 3.1 - Estrutura da Pesquisa

### 3.2.1 Planejamento inicial

Após definir o campo de pesquisa para o trabalho, algumas questões iniciais foram abordadas para assim dar início ao estudo, tais como, **Como o MDD e POA em conjunto com o *Hibernate* são utilizados na engenharia de software? - Quais os prós e contras em utilizar MDD, POA e *Hibernate*? - Existem ambientes que auxiliam no processo de geração de códigos de persistência de dados?**

Entretanto, as questões iniciais levantadas servem apenas para traçar uma trajetória a ser seguida, desta forma, contribui para um direcionamento que possa enfatizar a pesquisa proposta. Após definir essas questões estabelecidas inicialmente, pode-se delimitar a área de estudo e o objetivo a ser tratado. Assim, para definir as hipóteses de trabalho, é necessário explorar o campo de estudo, seguindo para a próxima fase, a exploratória.

### 3.2.2 Fase Exploratória

Segundo Gil (2002), a fase exploratória tem como objetivo proporcionar maior familiaridade com vistas a torná-lo explícito ou a construir hipóteses. Desta forma, é necessário explorar o objetivo de estudo para conduzir o trabalho.

Revisão Bibliográfica

Segundo Mafra e Travassos (2006), o levantamento bibliográfico se faz necessário, sendo que a revisão de literatura é o meio pelo qual o pesquisar pode identificar o conhecimento científico existente em uma determinada área, de forma a planejar sua pesquisa, evitando a duplicação de esforços e a repetição de erros passados.

Definição do Objetivo do Trabalho

As questões iniciais delimitaram o escopo de trabalho, auxiliando a identificar os problemas a serem tratados. Além disso, a revisão bibliográficademonstra uma visão geral do campo de estudo, dando suporte à definição das hipóteses propostas, que foram expostas Capítulo 1. Portanto, as fases subsequentes têm por objetivo dar subsídio para comprovar ou refutar as hipóteses.

### 3.2.3 Desenvolvimento

O desenvolvimento é a fase que utiliza todas as questões iniciais levantadas na fase do planejamento inicial. Além disso, todas as ideias e conhecimento adquirido na fase exploratória são colocados em prática a fim de responder as questões propostas.

Para atingir os objetivos propostos neste trabalho, o desenvolvimento será dividido em duas etapas subsequentes e dependentes entre si e descritas na sequência.

Abordagem Proposta

O objetivo deste trabalho é apresentar uma abordagem, no qual será necessário propor uma arquitetura para implementá-la. Esta arquitetura definirá e organizará os componentes necessários para implementar o ambiente voltado a geração de código de persistência de dados *Hibernate.*

Implementação do Ambiente

Para atingir o objetivo principal deste trabalho, a implementação de uma versão inicial do ambiente baseada na arquitetura e abordagem proposta foi realizada. Os componentes definidos na arquitetura serão implementados na linguagem de programação Java.

### 3.2.4 Avaliação e Conclusão

A última fase da estrutura de pesquisa é a de avaliação e conclusão do trabalho. Esta fase foi dividida em três etapas, sendo que as duas primeirassão referentes à avaliação da abordagem. Por fim, após avaliar a abordagem, a última etapa, consiste de conclusão do trabalho.

Planejar e Conduzir o Estudo para Avaliar a Abordagem

O planejamento e a condução do experimento para avaliação da abordagem se deram inicialmente escolhendo a melhor forma de ser realizada. Portanto, foi definida a utilização de métricas, que tem como intuito analisar os dados obtidos em um ambiente controlado para confirmar ou negar as hipóteses iniciais. Os participantes querealizarão o experimento serão os alunos do curso de Ciência da Computação da Universidade Estadual do Norte do Paraná.

Avaliar a Abordagem

Após o domínio ter sido definido, o experimento será concebido e após executado, tendo como a população alunos de Ciência da Computação e Sistemas de Informação da Universidade Estadual do Norte do Paraná. A análise dos dados será realizada de forma quantitativa.

Conclusões do Trabalho

Por fim, após o planejamento, execução e avaliação da abordagem, será possível concluir o trabalho. Utilizando a análise dos dados da avaliação da abordagem, em conjunto com todo o referencial teórico obtidos no decorrer da pesquisa, as conclusões finais do trabalho poderão ser expostas, apresentando elementos que indicam efetividade da abordagem proposta.

# 4. ABORDAGEM PROPOSTA

Na área de desenvolvimento de software, novas técnicas e métodos estão sempre sendo propostos, a fim de minimizar a complexidade do desenvolvimento de software. Entretanto, aplicar novos métodos e técnicas, muitas vezes não é trivial, uma vez que exige que os programadores aprendam essas técnicas e saibam como e onde aplicá-las. Além disso, é sempre desejável que as empresas apliquem padrões de programação, isto é, todos os programadores tenham o mesmo entendimento de como aplicar algum conceito. Por exemplo, suponha-se que uma empresa queira adotar a programação orientada a aspecto, sua equipe precisará saber em quais situações o uso dessa abordagem é vantajosa, além de precisar saber como implementá-la.

Portanto, uma implementação correta da POA não é simples. Muitos estudos indicam em que situações esta abordagem é viável, assim, uma boa estratégia é que um especialista defina quando ela pode ser usada. Destemodo, a abordagem proposta para a ferramenta auxiliará os desenvolvedores a definir como e onde POA pode ser aplicada, utilizando uma abordagem orientada a modelos.

Sendo assim, nesta seção será apresentada uma visão geral da abordagem proposta. Além do mais, partindo da visão geral, será apresentado detalhadamente os componentes do framework, como meta-modelo, que reflete o domínio de persistência de dados e o gerador de código.

## 4.1 O FRAMEWORK– VISÃO GERAL

O framework emprega uma abordagem orientada a modelos. O framework é composto por um meta-modelo, que descreve os conceitos essenciais do domínio de geração de códigos de persistência de dados *Hibernate*, e um gerador de código para geração de códigos fontes.

Sendo assim, a abordagem é composta sobretudo por cincopassos, como é apresentado na Figura 4.1.

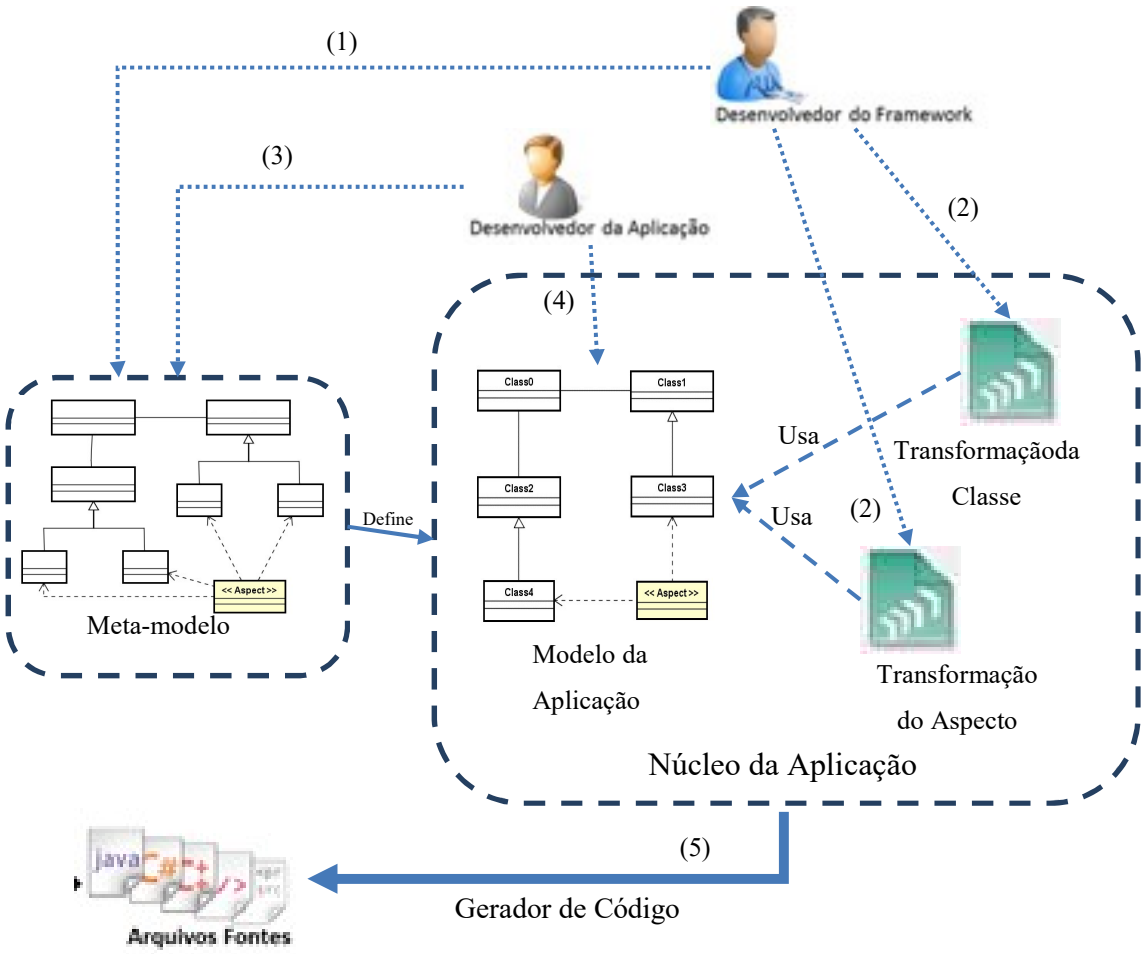



Figura 4.1 – Abordagem proposta

Na abordagem proposta há dois papéis

- O desenvolvedor do framework: responsável por criar os Meta-Modelos e as transformações.
- O desenvolvedor da aplicação: responsável por criar um modelo e posteriormente uma aplicação baseada no modelo, meta-modelo e nas transformações existentes.

O primeiro passo (1) desta abordagem é a criação do meta-modelo.O meta-modelo descreve os conceitos essenciais que uma aplicação deve ter para um domínio específico, portanto este meta-modelo é um guia para criar aplicações deste domínio. Dessa forma, qualquer aplicação desenvolvida pelo Desenvolvedor de Aplicação deve ser criada a partir do meta-modelo.

O meta-modelo é desenvolvido por meio do Framework de Modelagem do Eclipse (EMF, do inglês *Eclipse Modelling Framework*). O EMF é instalado como

um*plugin* da plataforma eclipse e o processo de modelagem, por completo, é realizado por meio de mecanismos oferecidos pelo*plugin*, como também pela IDE. Contudo, esta etapa somente será utilizada quando o meta-modelo for criado ou alterado. Assim, o desenvolvedor começará pela fase 2.

Uma vez que o meta-modelo é construído, o segundo passo (2) é criar as transformações do modelo. Para isto, é proposto a utilização do *Acceleo* para realizar estas transformações (verseção 4.2), em que é projetado o gerador de códigos. Por meio de um projeto do *Acceleo*, os*templates*parageração de códigos são criados.

Após os meta-modelos e as transformações serem definidas, as aplicações podem ser desenvolvidas. O terceiro passo (3), o Desenvolvedor da aplicação escolhe o meta-modelo para implementação da persistência de dados *Hibernate*, e a partir disso, pode-se iniciar o desenvolvimento da aplicação.

Finalizado, o passo quatro (4) constitui-se pela criação do modelo da aplicação. Neste passo, o desenvolvedor cria o modelo da aplicação, que é o modelo criado e configurado com base no meta-modelo. O modelo criado em conjunto com as transformações constitui o núcleo da aplicação e, a partir destes elementos, pode ser gerado o código fonte.

Assim, por fim, no quinto passo (5), o gerador de código exige como entrada um modelo de aplicação e as transformações de classe e aspectos para geração de código fonte de baixo nível como saída.

## 4.2 O META-MODELO

O desenvolvimento dirigido a modelos pode ser em muitos casos iniciado a partir de um meta-modelo que contém regras, especificações e informações relevantes que construirão os modelos. De certa forma, é comum a existência de um meta-modelo por trás das cenas que decidirá o que é possível de ser realizado por meio de determinado modelo. De acordo com Guedes (2012), o meta-modelo define uma linguagem para expressar modelos, sendo mais compacto que um modelo que ele descreve.

Segundo Koch (2012), Eclipse é uma plataforma de desenvolvimento baseada em plugins que suporta uma variedade de linguagens de programação, e.g. Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Natural, Perl,

PHP, Prolog, Python, R, Ruby, Scala, Clojure, Groovy, Scheme, e Erlang, e outras ferramentas. Conforme citado por Koch (2012), Eclipse apresenta características importantes:

a) Livre integração com outras tecnologias;

b) Vasto suporte;

c) Comunidade atualmente ativa;

d) Disponibilidade de plugins;

e) Suporte para múltiplas linguagens de programação e modelagem; e

f) Ampla adoção e uso por desenvolvedores em serviço.

Ainda, segundo Kolovos *et. al.* (2010), Eclipse e EMF tem se tornado os padrões na comunidade de engenharia orientada a modelos, como também, na maioria das ferramentas MDE no atual mercado.

EMF é um framework de modelagem e mecanismo de geração de código para desenvolvimento de aplicações baseadas em um modelo de dados estruturado (NELLAIAPPAN; 2009). Entretanto, devido À ampla popularidade e ferramentas disponíveis, Eclipse e EMF podem ser vistas como alternativas mais apropriadas para o desenvolvimento do meta-modelo para o framework proposto nesse trabalho.

O meta-modelo é construído por meio do EMF, que por sua vez, utiliza o Ecore para construção dos domínios em questão. O Ecore corresponde a uma extensão do XML que utiliza *namespaces* para definir novas entidades e atributos. Não é adicionado a estrutura do meta-modelo em Ecore neste capítulo por questões de espaço. No entanto, este pode ser encontrado no Apêndice A.

Abaixo será apresentado uma breve explicação sobre cada componente composto no meta-modelo (**Erro! Fonte de referência não encontrada.**):

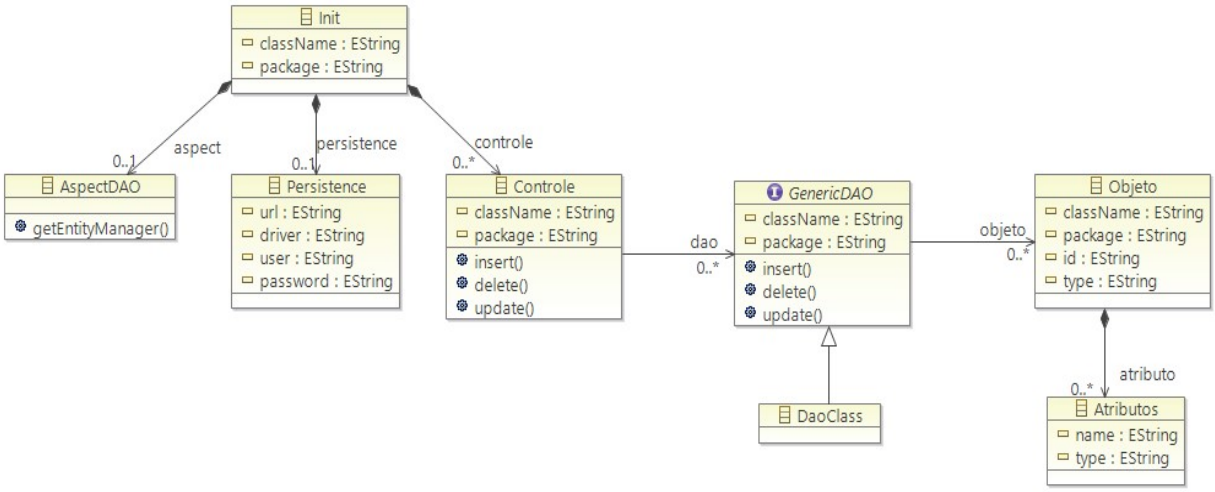



Figura 4.2 - Figura 4.2 – Meta-modelo do framework proposto (Elaborado pelo autor)

1. ***Init***: Tem como função ser o ponto de partida na construção do modelo. Sendo assim, é possível criar as entidades AspectDAO, Persistence e Controle.

2. **HibernateUtil**: Este componente é responsável pelas informações do banco de dados tais como: usuário, senha, URL e Driver.

- **AspectDAO**: O *AspectDAO* detém as informações da conexão para o banco de dados que foram definidas na entidade *Persistence*, e define um ponto de corte que é acionado assim que o método *insert*, *delete* ou *update* for executado, assim realizando uma consulta ou persistindo o dado.

3. ***Control*** (controle): A controle contém um conjunto de interfaces equivalentes ao conjunto de interfaces trazido pelo DaoClass, que pode ser utilizado para controlar as regras de negócio.

4. ***Object****:* Um POJO que determina na entidade de domínio os seus atributos e comportamentos. Em outras palavras, no domínio de persistência de dados, o *Objeto* também poderia ser descrito como um elemento da camada de modelo.Nesta entidade precisa ser definido qual é o seu *id* e *type*, ressaltando que, como se utiliza a tecnologia *Hibernate*, entende-se que cada *Objeto* é uma entidade no banco de dados.

5. ***Atributos:***Este componente está ligado aos itens que compõe um objeto.

6. ***GenericDAO:***Um esqueleto para o DaoClass. O GenericDAO é uma entidade “interface” que contém as declarações dos métodos (insert, delete, update) necessários de um DaoClass. Além disso, contém a funcionalidade que o AspectDAO usará como ponto de corte.
7. **DaoClass**: compõe o padrão de persistência de dados no qual permite separar regras de negócio das regras de acesso a banco de dados

## 4.3 O GERADOR DE CÓDIGOS

O gerador de código da ferramenta proposta neste projeto é construído com base no *Acceleo*e requer uma instância do meta-modelo de persistência de dados *Hibernate*como entrada para geração de código. Atualmente, diferentes ferramentas para transformação de modelos orientados a uma variedade de propósito têm sido exploradas, tais como: XSLT, QVT, ATL e Acceleo (KOCH, 2007).

O *Acceleo*[2]é uma implementação do padrão MOF. *Acceleo*prove um mecanismo simples para geração de códigos e desenvolvimento por meio de*templates*.

*Templates*são motores para geração de código no *Acceleo.* Como apresentado *na* ***Erro! Fonte de referência não encontrada.****,* os*templates* são constituídos por dois tipos de sintaxes: (1) a sintaxe literal, e (2) sintaxe dinâmica. A sintaxe literal (em fonte de cor preta na ***Erro! Fonte de referência não encontrada.***) diz respeito a tudo que será gerado no arquivo final, enquanto a sintaxe dinâmica, escrita entre colchetes, refere-se ao aspecto dinâmico do gerador, isto é, processamento e utilização das entidades e atributos instanciadas no modelo. O*template* apresentado na ***Erro! Fonte de referência não encontrada.*** gera uma classe simples em Java, constituída pelo método principal*main*, como demonstrado abaixo:

```
public class MyClass {
        public static void main(String[] args) {
        System.out.println("Hello World!");
```

[2] http://www.eclipse.org/acceleo

```
        }
    }
```

```
[comment encoding = UTF-8 /]
[module generate('http://pretcc.com')]


[template public generateElement(aMyClass : MyClass)]
[comment @main/]
[file (aMyClass.att1.concat('.java'), false, 'UTF-8')]
public class [aMyClass.att1/] {
    public static void main(String['['/]']'/] args) {
        System.out.println("Hello [aMyClass.att2/]!");
    }
}
[/file]
[/template]
```

Figura 4.3 - Exemplo de um template do Acceleo que gera uma classe Java.

O conteúdo da classe acima é gerado em um arquivo denominado MyClass.java, em que o nome do arquivo, como também o nome da classe, é um atributo dinâmico, como também, a palavra "*World*", que está contida no atributo dinâmico att2 em aMyClass.

Sendo assim, para atender os requerimentos do framework proposto, o *Acceleo* apresenta ser a alternativa mais adequada para geração de código, não só pela sua simplicidade e facilidade de uso, mas também devido a possibilidade de realizar transformações, de modelos de plataformas independentes (PIMs, do inglês *Plataform-Independent Models*), de forma automática,para códigos fonte de baixo nível (MTSWENI, 2012).

Na ferramenta proposta, o meta-modelo existirá dois tipos de transformações baseadas em *templates Acceleo,* que serão utilizados para gerar o código fonte:

- Transformação da Classe: utiliza o meta-modelo e o modelo para criar código fonte de classe Java.
- Transformação do Aspecto: utiliza o meta-modelo e o modelo para criar código fonte dos aspectos.

Uma vez que o meta-modelo e as transformações estão definidas, as aplicações podem ser desenvolvidas.

## 4.4 OS PAPÉIS

Como apresentado na seção 4.1, aabordagem proposta envolve partes distintas, tais como: (1) criação do meta-modelo; (2) criação das transformações do meta-modelo; (3) definição de qual meta-modela utilizar para desenvolver uma aplicação; (4) especificação do modelo; e (5) geração da aplicação de persistência.

Entretanto diferentes partes podem envolver diferentes papéis. Em outras palavras, dois papeis diferentes estão presentes em nossa abordagem. A Figura 4.4representa uma expansão da Figura 4.1, e atribui o papel presente em cada elemento do framework proposto. Os papeis presentes,na abordagem proposta, são:

a) Desenvolvedor do framework.

b) Desenvolvedor usuário do framework.

Primeiramente, referente a área verde da Figura 4.4, o desenvolvedor do framework está presente nas etapas de criação do meta-modelo, criação do plugin, e construção do gerador de código. Este papel compreende o mais baixo nível na hierarquia em termos de complexidade de suas ações. Este papel também é responsável por futuras extensões do framework.

O segundo papel, referente a área azul da Figura 4.4, compreende o usuário do framework, compreendendo o público alvo. Este papel representa o desenvolvedor que utiliza o framework para desenvolvimento, isto é, geração de códigos de persistência de dados *Hibernate.* Por sua vez, está presente somente na etapa de instância do modelo, que ocorre sob o *plugin* previamente construído. Em outras palavras, este usuário basicamente cria um modelo de aplicação, e o executa, o que resulta na geração do código da aplicação. É importante ressaltar que este papel não está envolvido na criação ou expansão do meta-modelo, e não está envolvido no projeto do gerador de códigos, tampouco requer o conhecimento necessário para tais. Também, é importante notar que o papel desenvolvedor do framework também está presente neste passo, no entanto, como o criador do plugin, e não como usuário.

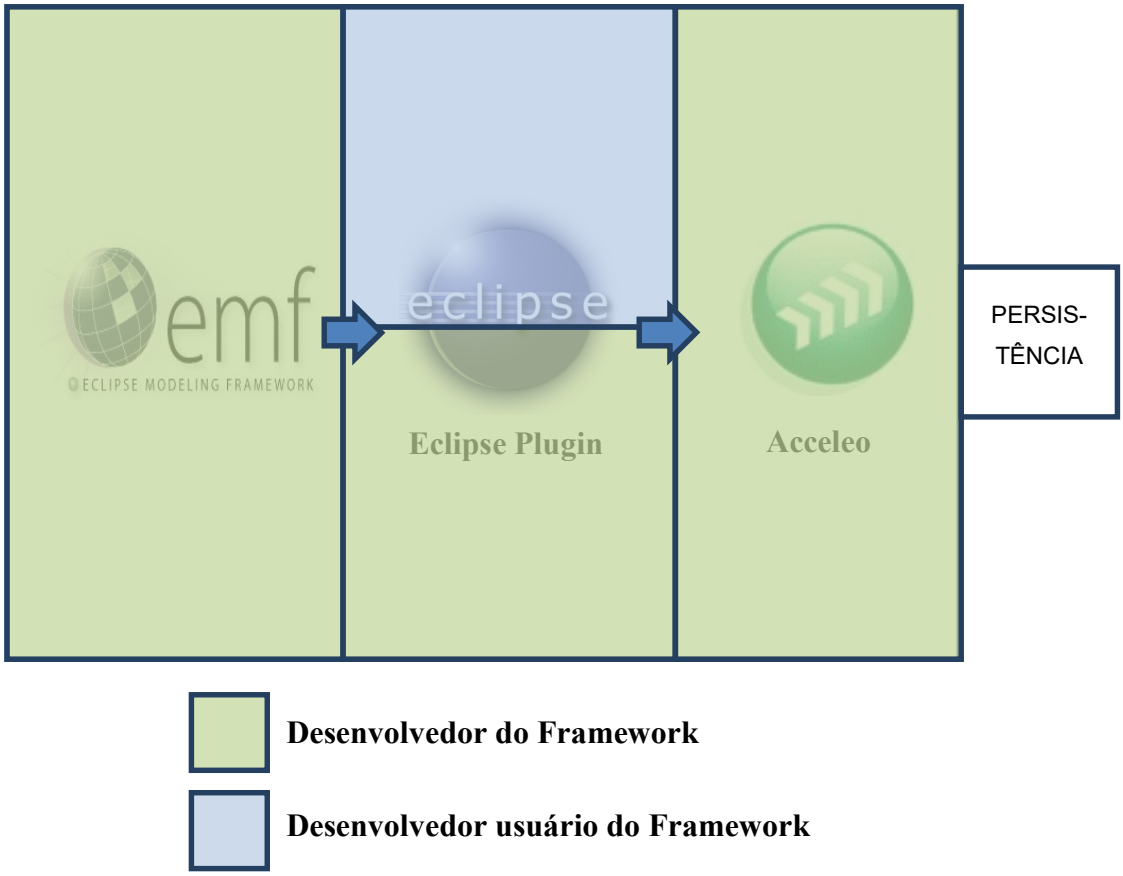



Figura 4.4 – Diferentes papéis presentes no framework proposto (Elaborado pelo autor)

# 5. EXPERIMENTO

Nestecapítulo é apresentado o método experimental. O método experimental é constituído pelos principais pontos abordados no experimento e apresenta tais pontos de forma teórica.Também, nestecapítulo é apresentado o planejamento pelo qual o experimento foi realizado.

## 5.1 MÉTODO EXPERIMENTAL

O presente experimento é caracterizado como um estudo experimental, que visa coletar dados em um ambiente controlado, para confirmar ou negar uma hipótese. O método experimental considera a proposta e avaliação de um modelo com os estudos experimentais (BASILI, 1996). Experimentação provê uma maneira sistemática, disciplinada, quantificável e controlada para avaliar atividades baseada em humanos. Esta é uma das muitas razões pelos o qual pesquisas empíricas são comuns nas ciências sociais e ciências comportamentais (WOHLIN, 2012).

No entanto, é importante ressaltar que os experimentos não provam nada. Nenhum experimento fornece uma prova com certeza absoluta. Os experimentos verificam uma previsão teórica contra a realidade (BASILI, 1996). Neste experimento, foi utilizado um experimento controlado em um pequeno objeto de estudo. A maior vantagem do experimento é o controle total sobre o processo, além da possibilidade de repeti-lo.

Objetivo do Estudo

Avaliar se a abordagem para persistência de dados JPA baseada em MDD e POA é melhor à implementação de persistência de dados da forma tradicional, em relação ao tempo de desenvolvimento e qualidade do código.

Propósito

Propor a implementação de problemas reais e comparar o desenvolvimento utilizando a abordagem baseada em MDD e POA com implementação sem o uso da abordagem. Além disso, é avaliado o uso das tecnologias Hibernate e JDBC para a persistência de dado em conjunto com a implementação da abordagem.

Com foco na Qualidade

A qualidade foi medida através de métricas, que podem ser classificadas

como:

- Tamanho do código: Linha de código (LOC); Número de Atributos (NOA); e Comprimento das Operações por Componentes (WOC).
- Coesão: Falta de coesão nas operações (LCOO).
- Acoplamento: Acoplamento Eferente (CE); Acoplamento Aferente (CA); Profundidade da Árvore de Herança (DIT)
- Separação de Interesses: Difusão do Interesse sobre os Componentes (CDC); Difusão do Interesse sobre as Operações (CDO); Difusão do Interesse sobre as linhas de códigos (CDLOC);
- Tempo de desenvolvimento

Perspectivas

A perspectiva é a partir do ponto de vista dos pesquisadores, ou seja, o pesquisador quer identificar se há alguma diferença na produtividade dos programadores que usam as duas formas distintas de implementação e se há diferença na qualidade do código gerado pelas duas formas de implementação.

Contexto

Avaliar implementações de contratos em um sistema, implementando-os utilizando diferentes técnicas de persistência de dados. Os programadores serão alunos do curso de Ciência da Computação que tenham bom conhecimento de programação orientada a objetos, conheçam a tecnologias de pertinência de dados Hibernatee JDBC. Além disso, os programadores deveriam ter conhecimento sobre os temas abordados no experimento, tais como: conhecimento sobre o padrão DAO; conhecimento sobre as tecnologias utilizadas no experimento (*Hibernate e* JDBC); e conhecimento sobre a abordagem proposta, em que foi dado um treinamento para que ambos os programadores ficassem equivalentes.

## 5.2 PLANEJAMENTO

O planejamento do experimento foi dividido em cinco etapas. Cada etapa é descrita nas seções seguintes.

Visto que a característica desse experimento é um estudo experimental, de acordo com Rauen (2012), o pesquisador assume hipoteticamente a possibilidade

de uma variável ter relação assimétrica com outra variável, ou seja, um fato ou fenômeno é fator de causa para outro fenômeno. Diante disso, foi necessário definir as variáveis dependentes e independentes antes de testar um modelo específico contra as observações dos fenômenos (Pinsonneault; Kraemer, 1993).

#### 5.2.1.1 Variáveis Independentes

Variável independente, X ou VI, é a variável que exerce influência sobre outra variável, ou seja, são os fatores determinantes para que ocorra um determinado resultado, efeito ou consequência (MARCONI; LAKATOS, 2000). Geralmente, são controladas pelo pesquisador em seus experimentos a fim de procurar estabelecer sua relação e influência sobre o resultado de um fenômeno.

As variáveis independentes definidas neste experimento foram os participantes, as tecnologias de persistência de dados JPA e JDBC, o MDD e POA.

#### 5.2.1.2 Variáveis Dependentes

Variável dependente, Y ou VD, de acordo com Marconi e Lakatos (2000), é a variável a ser explicada ou descoberta, em virtude de ser influenciada, determinada ou afetada pela variável independente. O resultado que a variável apresenta pode variar de acordo com o pesquisador, pois, o mesmo pode colocar, tirar ou modificar a variável independente. Segundo Jung (2009) as variáveis dependentes são aquelas cujo comportamento se quer verificar em função das oscilações das variáveis independentes, ou seja, correspondem àquilo que se deseja prever e/ou obter como resultado.

As variáveis dependentes definidas neste experimento foi o tempo de desenvolvimento de desenvolvimento e a qualidade do código das implementações.

### 5.2.2 Definição das hipóteses

Para orientar a pesquisa, quatro hipóteses foram estabelecidas com base no objetivo principal deste experimento,que são descritas abaixo:

- **Hipótese (Ha0)**$\mu_{tradicional} = \mu_{MDD}$: em que existe diferença de produtividade em relação ao uso da abordagem baseada em MDD com aspecto, em relação à implementação tradicional.

- **Hipótese (Ha1):** $\mu_{tradicional} < \mu_{MDD}$, em que abordagem baseada em MDD e Aspecto é superior na produtividade em relação ao desenvolvimento tradicional de software.
- **Hipótese (Hb0):** $\mu_{tradicional} = \mu_{MDD}$, em quenão existe diferença de qualidade em relação ao uso da abordagem baseada em MDD com aspecto, em relação a implementação tradicional.
- **Hipótese (Hb1):** $\mu_{tradicional} < \mu_{MDD}$, em quea abordagem baseada em MDD e Aspecto gera um código superior em qualidade em relação ao desenvolvimento tradicional de software.

### 5.2.3 Seleção de Participantes

Os participantes que colaboraram neste experimento foram estudantes de graduação do curso de Ciência da Computação. Os participantes são indivíduos especialmente selecionados da população de interesse para a realização do experimento, uma vez que o experimento quer simular o comportamento de indivíduos dentro de uma empresa de desenvolvimento de software. Assim, o grupo de participantes tem de ser equivalentes, para isto foi realizado um treinamento, de modo que todos sejam capazes de realizar as funções propostas em um tempo semelhante.

Portanto, na seleção dos participantes, utilizou-se uma amostragem não probabilística, considerando uma amostragem por conveniência, o qual as pessoas mais próximas e mais convenientes são selecionadas (WOHLIN, 2012).

### 5.2.4 Descrição do Experimento

O objetivo deste experimento é a avaliação da abordagem proposta, ou seja, verificar se a mesma pode auxiliar no tempo de desenvolvimento e na qualidade dos códigos.

Sendo assim, os participantes foram solicitados a realizar duas implementações de um dos casos de usos (Apêndice C) de um sistema para biblioteca. O caso de uso está relacionado ao empréstimo de livros de uma biblioteca, desenvolvido na linguagem de programação Java.

Ambas implementações propostas foram aplicação de padrão e tecnologia

de persistência de dados. Na primeira, os participantes foram solicitados a implementar uma parte do sistema, a funcionalidade de empréstimos de livros, aplicando o padrão DAO com *Generic* DAO e a tecnologia *Hibernate.* Entretanto, a implementação foi dividida em pequenas tarefas:

- Implementar a funcionalidade do sistema utilizando a abordagem proposta.
- Implementar a funcionalidade do sistema sem utilizar a abordagem proposta.

Na segunda implementação, os participantes foram solicitados a implementar a mesma funcionalidade, aplicando o mesmo padrão, contudo agora utilizando a tecnologia JDBC. Assim, a segunda implementação foi dividia em duas tarefas:

- Implementar a funcionalidade do sistema utilizando a abordagem de geração de códigos de persistência de dados JDBC.
- Implementar a funcionalidade do sistema sem utilizar a abordagem.

Com isto, de forma a auxiliar os participantes a desenvolverem o caso de uso, foi dado um conjunto de artefatos:

1. Requisitos, no qual foi descrito em Casos de Uso (Apêndice C);
2. Diagrama de Classes (Apêndice D);
3. Diagrama de Sequência da funcionalidade emprestar (Apêndice E);
4. Classes de Domínio; e
5. Script para geração da base de dados relacional do projeto em MySQL (Apêndice F)

### 5.2.5 Descrição da Análise.

Neste experimento, foi utilizada uma análise quantitativa. Experimentos são quase puramente quantitativos, uma vez que têm direcionamento na contagem de diversas variáveis. Durante estas investigações, os dados quantitativos são recolhidos e, em seguida, são aplicados métodos estatísticos.

Sendo assim, a análise foi realizada através de um conjunto de métricas de

separação de interesses, de acoplamento, de coesão e de tamanho (SANT'ANNA et. al., 2003) para avaliar as implementações utilizando a abordagem proposta no presente trabalho e as que não utilizam a abordagem. As métricas citadas já foram utilizadas em outros estudos como; (GARCIA et. al., 2005; SOARES, 2004; OLIVEIRA et. al., 2008), onde foram aplicadas neste trabalho por considerar as abstrações e o mecanismo da POA. Essas métricas foram definidas com base no reuso e refinamento de algumas métricas clássicas OO (CHIDAMBER; KEMERER, 1994; FENTON; PFLEEGER, 1997), que foram estendidas para o paradigma orientado a aspecto (OLIVEIRA *et. al.,* 2008).

Com isto, a fim de responder às hipóteses propostas no Capítulo 5.2.1, é apresentado noQuadro 5.1as métricas utilizadas para avaliar a implementação e na mesma é apresentada uma breve definição de cada métrica aplicada, e associada com os atributos medidos por cada uma.

Quadro5.1 - Métricas

| ATRIBUTOS | MÉTRICAS | DEFINIÇÕES |
|---|---|---|
| **Separação De Interesses** | Difusão do interesse sobre os componentes (CDC) | Conta o número de classes e aspectos que a principal função é contribuir com a implementação de um interesse e o número de outras classes e aspectos que os acessam. |
| | Difusão do interesse sobre as operações (CDO) | Conta o número de métodos e *advices* que a principal função é contribuir com a implementação de um interesse e o número de outros métodos e advices que os acessam. |
| | Difusão do interesse sobre as linhas de códigos (CDLOC) | Conta o número de pontos de transições para cada interesse por meio de linha de código. Ponto de transições são pontos no código em que há uma troca de interesses. |
| **Acoplamento** | Acoplamento Eferente (CE) | Representa a contagem de quantas classes diferentes referem-se à classe atual, por meio de campos ou parâmetros. |
| | Acoplamento Aferente (CA) | Representa a contagem de quantas classes diferentes a classe atual faz referência, por meio de campos ou parâmetros. |
| | Profundidade da árvore de herança (DIT) | Conta a profundidade da hierarquia de herança em que uma classe ou aspecto é declarado. |
| **Coesão** | Falta de coesão nas operações (LCOO) | Mede a falta de coesão de uma classe ou um aspecto em termos da quantidade de pares de métodos e advices que não acessam a |

| ATRIBUTOS | MÉTRICAS | DEFINIÇÕES |
|---|---|---|
| | | mesma variável de instância. |
| **Tamanho** | Linhas de código (LOC) | Conta o número de linhas de código. |
| | Número de atributos (NOA) | Conta o número de atributos de cada classe ou aspecto. |
| | Comprimento das operações por componentes (WOC) | Conta o número de métodos e advices de cada classe ou aspecto e o número de seus parâmetros. |

FONTE: OLIVEIRA et. al. 20

### 5.2.6 Execução do Experimento

Como citado naSeção 5.2.3, cada implementação foi dividida em duas tarefas, em que se resultou em duas implantações, totalizando quatro implantações por participante. As tarefas foram denominadas por 1a, 1b, 2a e 2b. Para realização de todas as tarefas, os participantes tinham conhecimento prévio sobre o padrão DAO e as técnicas de persistência de dados utilizadas, *Hibernate* e JDBC.

Sendo assim, após a separação das tarefas, os participantes foram separados em grupos, denominados G1 e G2, de modo a se ter um experimento equilibrado, com duas equipes de desenvolvimento homogêneas. Os grupos foram separados por nível de conhecimento, em que os integrantes do grupo G1 continha um maior conhecimento sobre a abordagem proposta, e os integrantes do grupo G2 estavam em treinamento, ou seja, estava começando a ter conhecimento sobre o MDD e o POA. Com isto, o método experimental utilizado foi um estudo replicado (experimento controlado), com dois tratamentos: implementando a mesma tarefa com e sem a abordagem proposta.

O experimento foi executado duas vezes, cada execução com um grupo que deveria realizar a implementação proposta. Entretanto, um integrante começava a implementar utilizando a abordagem e o outro sem utiliza-la. Todavia, durante a execução do experimento, os participantes tiveram o livre acesso a materiais externos (livros, internet, etc.) que pudesse ajuda-los a resolver as tarefas.

A Tabela 5.2apresenta a ordem de execução das implantações de cada grupo durante o experimento, nas quais as implementações 1a e 2a se refere as implementações utilizando a tecnologia de persistência de dados *Hibernate* e JDBC respectivamente usado o meta-modelo, enquanto as implementações 1b e 2b não

usaram o meta-modelo.

Tabela 5.2 – Ordem de implementação

| GRUPO | INTEGRANTES | ORDEM DE IMPLEMENTAÇÃO | | | |
|---|---|---|---|---|---|
| G1 | ALUNO 1 | 1a | 1b | 2b | 2ª |
| | ALUNO 2 | 2b | 2a | 1a | 1b |
| G2 | ALUNO 3 | 1b | 1a | 1a | 1b |
| | ALUNO 4 | 2a | 2b | 1a | 1b |

Após a realização dos experimentos, os participantes anotaram o início e o fim de cada implementação, para que se pudesse verificar o tempo de desenvolvimento de cada tarefa. Para este experimento não foi definido um tempo limite de desenvolvimento, sendo assim, os programadores tinham o tempo aberto para realizar o experimento.

De forma a evitar problemas relacionados a confidencialidade do trabalho proposto, foi solicitado aos participantes que assinassem um termo de consentimento e de esclarecimento(Apêndice C), em que neste consta que os participantes não poderiam repassar os dados a terceiros, e que não seriam divulgados os nomes dos participantes neste trabalho.

No capítulo 6, são apresentados os resultados obtidos das análises realizadas, baseando-se nas métricas definidas na Quadro5.1.

# 6. ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta fase do trabalho, os dados são trabalhados e exibidos graficamente para facilitar a análise dos resultados, a fim de se obter um entendimento melhor sobre a amostra.Sendo assim, os dados obtidos foram submetidos a operações estatísticas, tais como: média, mediana, teste binomial, desvio padrão e distribuição normal.

Em conjunto com o trabalho desenvolvido, outro, foi realizado para avaliar a abordagem proposta, neste foi utilizado a tecnologia JPA. Dessa forma, para avaliar se a abordagem proposta, em que soluções que utiliza MDD, podem ser realizadas em um tempo menor que soluções que não utilizam a abordagem, foi-se utilizado os dados gerados neste trabalho. Assim como no presente trabalho, no projeto que utilizou JPA foi aplicadoo mesmo experimento que o apresentado aqui, para confrontar o uso da abordagem em conjunto com a tecnologia JPA em detrimento da tecnologia JDBC. Portanto, considerando que os dois estudos utilizaram a mesma abordagem para comparar o seu desempenho com aplicações desenvolvidas sem o uso da abordagem, diferenciando apenas a tecnologia utilizada, e os dois foram realizados nas mesmas condições, foi possível utilizar os dados destes projetos para analisar o tempo e qualidade de desenvolvimento com ou sem o uso da abordagem proposta.

Sendo assim, nesta seção, baseada nas categorias e questões definidas na seção 5.2.4, foram extraídos os resultados, analisando os dados obtidos através da realização do experimento. Abaixo é apresentada a análise e o resultado de cada uma das categorias.

## 6.1 TEMPO DE DESENVOLVIMENTO

Nesta categoria, apenas uma questão foi elaborada: “Desenvolvedores que utilizam uma abordagem MDD, solucionam as tarefas mais rapidamente do que os não utilizam? ”

**Resultados**

Para começar a analisar os dados, é necessário negar a hipótese nula, Ha0 (definido na seção 5.2.1), isto é, a intenção do experimento é rejeitar a hipótese. (WOHLIN *et. al.,* 2012). De acordo com Wohlin *et. al.* (2012), se a hipótese nula não

for rejeitada, nada pode ser dito sobre o resultado alcançado, no entanto, se a hipótese for rejeitada, pode-se afirmar que a hipótese é falsa.

Sendo assim, foi utilizado o método estatístico não-paramétrico, este método consiste em uma análise de forma mais geral dos dados enquanto o método estatístico paramétrico faz suposições sobre a distribuição dos dados (WHOLIN *et. al*, 2012). A utilização desse método se deu devido ao fato de que foi contado quantas implementações com a utilização da abordagem foram mais rápidas comparado com as implementações sem a abordagem, tratando assim os dados de maneira mais geral.

Para isto, foi realizado o teste binomial para medir o tempo de desenvolvimento de cada implementação e foram classificadas da seguinte maneira: Tradicional (sem utilizar a abordagem) e MDD (utilizando a abordagem). Entretanto, é necessário realizar um teste para verificar se há diferença no tempo de implementação, assim verificar se a hipótese Ha0 é falsa.

Portanto, a hipótese nula pode ser formulada como:

$$\mathbf{Ha0 : P(tradicional) = P(MDD) = ½}$$

Segundo Wholin *et. al.* (2012), é decidido que o $\alpha$ deve ser menor que 0,10. A Tabela 6.1 apresenta os dados obtidos do experimento:

Tabela 6.1–Comparação Implementações com Menor Tempo de implementação (Com abordagem x Sem abordagem)

| Implementação | Número de implementação mais rápidas |
|---|---|
| Com Abordagem | 11 |
| Sem Abordagem | 1 |

Dessa forma, para verificar se a hipótese Ha0 é falsa, é utilizada a seguinte fórmula:

$$\mathbf{P(0\text{-}1\ sem\ abordagem) = \sum_{i=0}^{1} \left(\frac{12}{i}\right) \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{12-i} = \frac{1}{2^{12}} \sum_{0}^{1} \left(\frac{12}{i}\right) = 0.003}$$

Para fazer o teste binomial foi utilizada a linguagem de programação R, pois, a mesma é voltada para análise estatística. Ao realizar o teste binomial, o resultado do α é de 0.003, com isso, pode-se concluir que a hipótese Ha0 é falsa, pois, o $\alpha < 0.10$, indicando o grau de significância da hipótese ser nula. A partir do

momento que a hipótese nula é negada, consegue-se analisar o tempo de desenvolvimento das implementações.

O tempo de desenvolvimento para os dois grupos, em média sempre foi menor para as implantações utilizando o MDD, como pode ser observado no gráfico daFigura 6.1. O gráfico está separado entre grupos dos participantes apresentado na seção 5.2.5. Sendo assim, a medianafoi de 67 minutos para as implementações utilizando o MDD, enquanto para as implementações sem MDD, o tempo mediano foi de 93 minutos. Para as implementações em que não foi utilizado o MDD, o tempo total de desenvolvimento dos dois grupos foi em média 28% superior em relação as implementações que se utilizou o MDD.

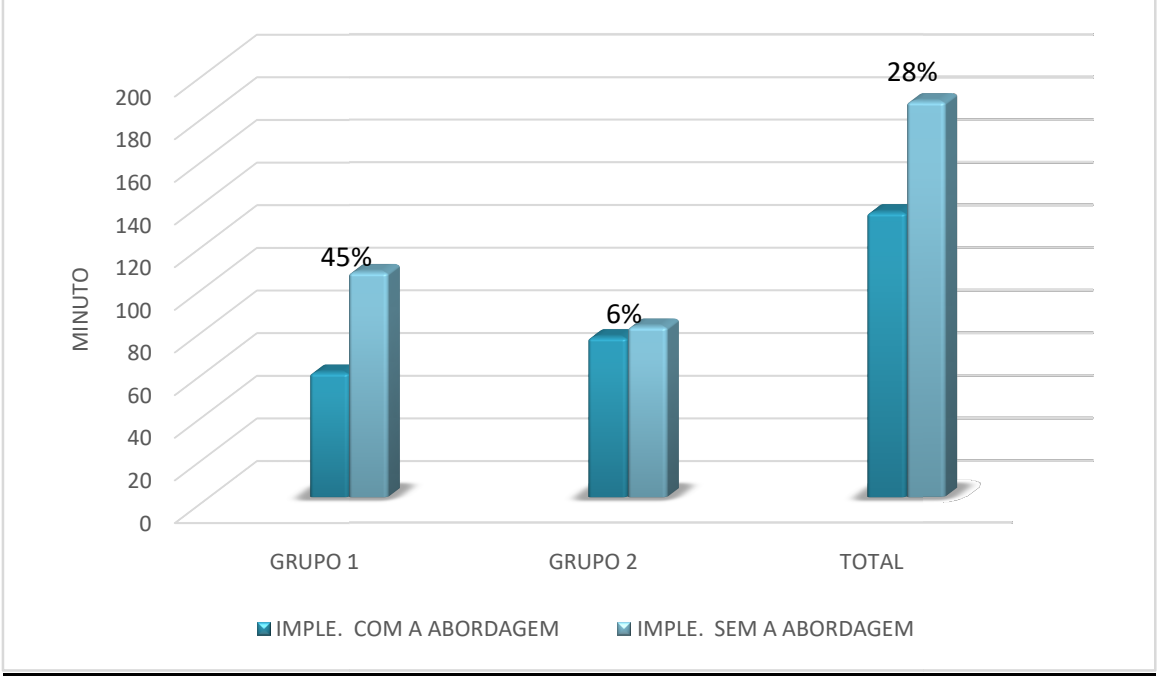



Figura 6.1 – Comparação do tempo de implementação (Implementação com abordagem x sem abordagem)

Entretanto, verificar somente a média do tempo de implementação não se tem uma precisão dos dados, pois podem existir dados muito distintos. Portanto, foi utilizado também, o desvio padrão como forma de aumentar a precisão da análise. Assim, o tempo de desenvolvimento dos dois grupos, para implementações que não utilizam o MDD,de acordo com o desvio padrão, foi 14% superior que a abordagem proposta.

Em conjunto com desvio padrão foi utilizado a distribuição normal, com objetivo de verificar a dispersão dos dados, assim mostrar o quão os dados estão distribuídos e o quão eles são precisos.

Na Figura 6.2 há dois gráficos: tempoA (implementações com o MDD) e o tempoS (implementações sem o MDD). Pode-se notar no gráfico tempoA que os dados estão mais concentrados em relação ao gráfico tempoS que apresenta os dados mais distribuídos.

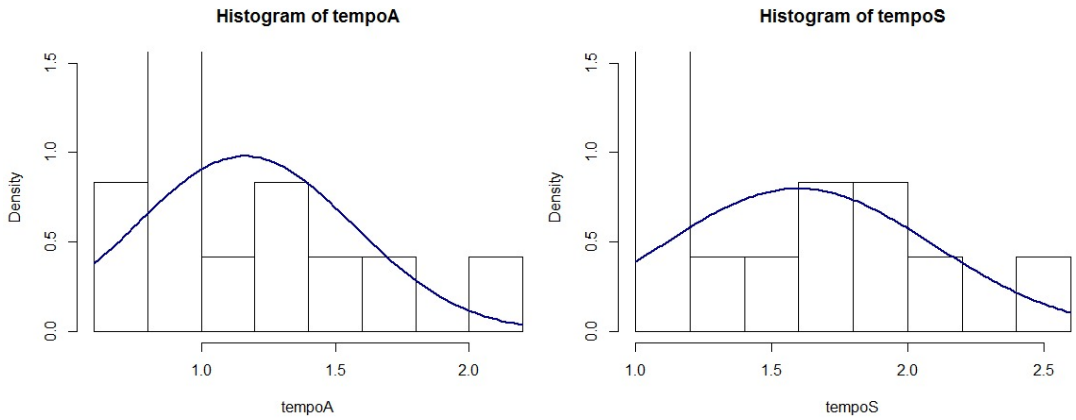



Figura 6.2 - Gráficos de distribuição normal, tempoA x tempoS

Após o tratamento dos dados obtidos através do experimento aplicado, pode-se verificar que o tempo de implementação de persistência de dados se reduz ao utilizar a abordagem proposta, ou seja, o tempo é reduzido em 14%, como apresentado anteriormente. Assim, observando a Figura 6.2, nota-se que em uma implementação sem a utilização de MDD levaria aproximadamente de 80 minutos a 140 minutos, enquanto que com a abordagem o tempo seria de 30 minutos a 110 minutos.

## 6.2 QUALIDADE DA IMPLEMENTAÇÃO

Nesta categoria, apenas uma questão foi elaborada: “A utilização da abordagem proposta, auxilia a melhorar a qualidade das implementações das tarefas propostas? ”.

**Resultados**

Do mesmo modo em que foi realizado no tempo de desenvolvimento, também foi necessário negar a hipótese nula (definido na seção 5.2.1). Sendo assim, foi realizado o teste binominal para medir a qualidade de cada implementação onde forma classificadas como: Tradicional (sem utilizar a abordagem) e MDD (utilizando a abordagem).

Sendo assim, a tabela apresenta os dados obtidos do experimento:

Tabela 6.2 - Qualidade de implementação

| Implementação | Número de Implementação com mais qualidade |
|---|---|
| Abordagem | 70 |
| Sem Abordagem | 50 |

Dessa forma, para verificar se a hipótese Hb0 é falsa, é utilizada a seguinte fórmula:

$$\textbf{P(0-1 sem abordagem)} = \sum_{i=0}^{50} \left(\frac{120}{i}\right) \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{120} = \frac{1}{2^{120}} \sum_{0}^{50} \left(\frac{120}{i}\right) = \textbf{0.041}$$

Para fazer o teste binomial foi utilizada a linguagem de programação R, pois, a mesma é voltada para análise estatística. Ao realizar o teste binomial, o resultado do α é de 0.041, com isso, pode-se concluir que a hipótese Hb0 é falsa, pois, o $\alpha < 0.10$, indicando o grau de significância da hipótese ser nula. A partir do momento que a hipótese nula é negada, consegue-se analisar a qualidade das implementações.

Assim, a comparação foi dividida em duas partes. A seção 6.2.1 apresenta a análise da separação de interesses, da persistência de dados, relacionadascom as implementações com a abordagem e sem a abordagem. Enquanto a seção 6.2.2 apresenta os resultados das métricas de acoplamento, coesão e tamanho

O eixo Y do gráfico apresenta os valores absolutos obtidos pela aplicação das métricas. A cada par de barras, corresponde os valores obtidos de cada implementação, junto a cada barra é apresentado a diferença, em porcentagem, dos resultados obtidos das implementações com e sem o MDD.

### 6.2.1 Separação de Interesses

Observa-se na Figura 6.3, que todas as implementações utilizando a abordagem obteve uma melhor qualidade em relação a separação de interesses das classes que implementam a persistência de dados.

Sendo assim, isto aconteceu devido ao aspecto separar os interesses do sistema. Portanto, como apresentado na Figura 6.3, o CDC (difusão do interesse

sobre os componentes) teve uma melhora de 83%.Enquanto que no CDO (difusão de interesse sobre as operações)teve uma melhora de 86%. E na última métrica, a difusão de interesse sobre as linhas de códigos (CDLOC), obteve-se uma melhora de 91%.

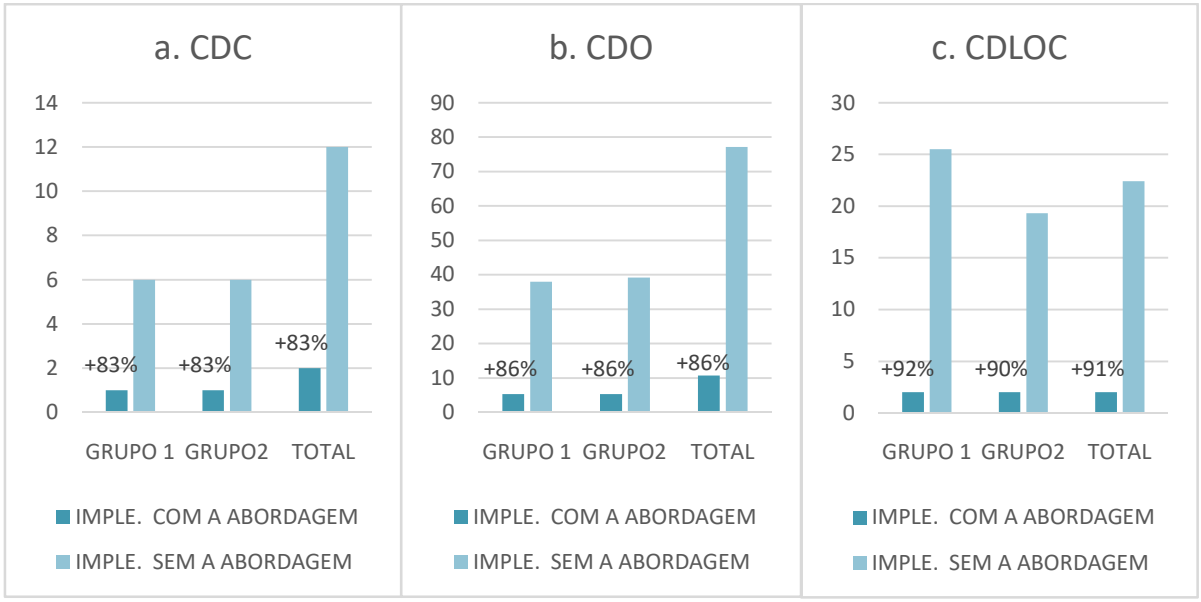



Figura 6.3 – Separação de Interesses

### 6.2.2 Acoplamento, coesão e tamanho

Na Figura 6.4, é apresentado três métricas, a profundidade da árvore de herança (DIT), o acoplamento eferente (CE) e o acoplamento aferente (CA). Nota-se no DIT, não se teve diferença em relação as duas abordagens.

No entanto, os resultados obtidos nas métricas de CE e CA, mostram que as implementações utilizando a abordagem, teve uma redução de acoplamento de Y% e 3% respectivamente.

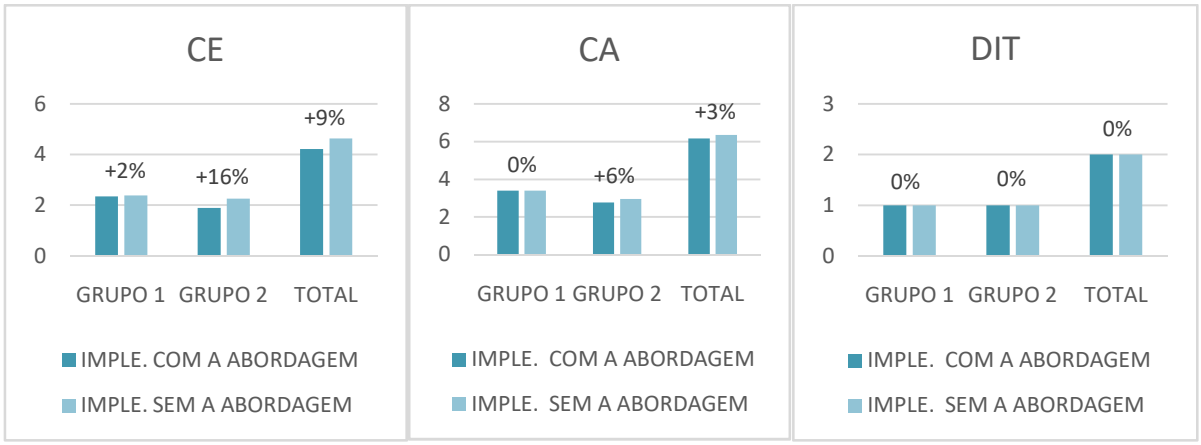



Figura 6.4 – Gráfico de acoplamento dos projetos

Em relação a métrica de falta de coesão nas operações (LCOO), as implementações, com a abordagem proposta, de ambos os grupos, teve-se uma pequena diferença de 4%, como é ilustrado na Figura 6.5.

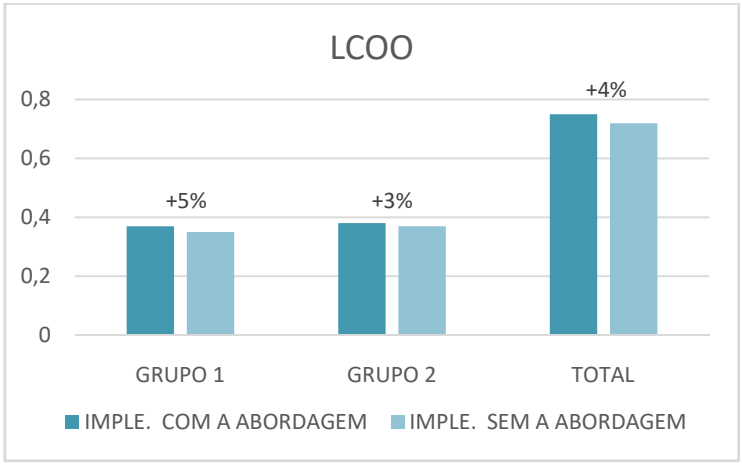



Figura 6.5 – Gráfico de falta de coesão das implementações

Nas últimas métricas, referente ao tamanho do código, são analisados os seguintes dados: linha de códigos (LOC), número de atributos (NOA) e comprimento das operações por componente (WOC).

Como ilustrado na Figura 6.6, a diferença de LOC foi de 4%, onde favoreceu a abordagem proposta. Em relação a métrica NOA, a implementação com abordagem foi 2% melhor que sem a abordagem. Enquanto na WOC, observa-se que houve uma variação de 3%.

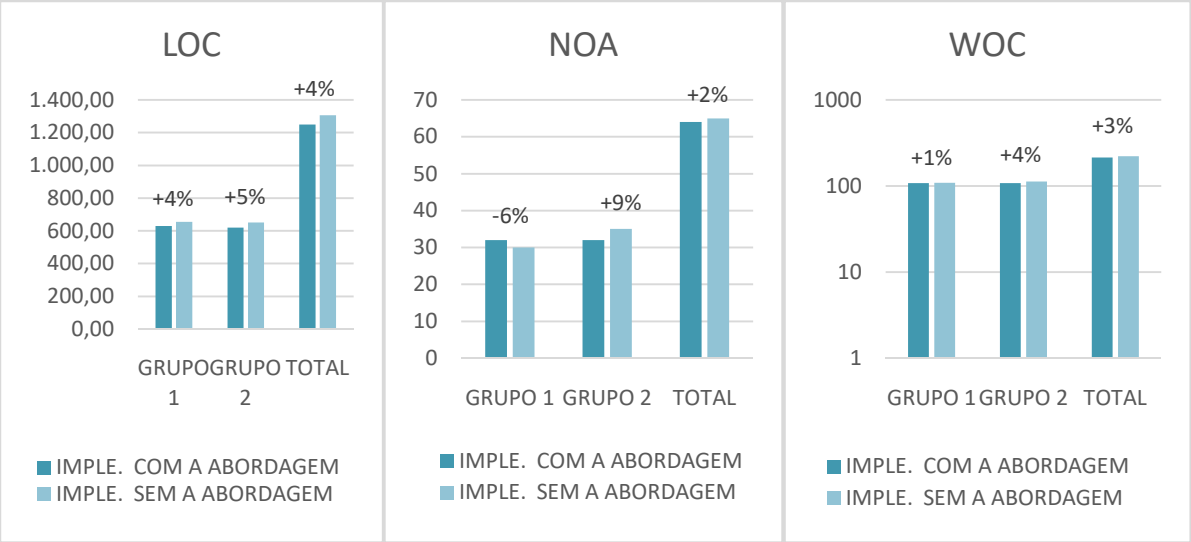



Figura 6.6 – Gráfico de tamanho do código das implementações

### 6.2.3 Discussão das métricas de qualidade

A análise dos resultados obtidos indica que a qualidade na separação de interesse de cada implementação por grupo que utilizaram a abordagem, foi melhor do que as implementações que não utilizaram. Assim, a abordagem se mostra eficiente, principalmente quando se trabalha com o padrão DAO e as tecnologias de persistência de dados, ou seja, diminuindo os interesses transversais de uma classe.

Consequentemente, os resultados das métricas de acoplamento, CE e CA, reforçam a qualidade na separação de interesse, pois, como seus interesses estão em um aspecto, às classes que implementam DAO não precisam persistir os dados como é feito na OO, deixando que o aspecto realize a persistência baseado na sua tecnologia. Entretanto, a métrica DIT, não apresenta nenhuma alteração, visto que, nas implementações não houve heranças.

Além disso, a métrica de coesão também demonstra um resultado positivo, devido ao fato dos resultados anteriores serem satisfatório, isso faz com que a manutenção, reuso e o entendimento do código de persistência de dados fiquem mais fácil e claro para os programadores.

Da mesma forma, observa-se que as métricas de tamanho, LOC, NOA e WOC, mostram resultados positivos. Com a utilização do aspecto, o tamanho do projeto consegue ser reduzido consideravelmente, pois, não há necessidade de repetir o código de persistência de dados, uma vez que, já estão implementados no

aspecto.

Com isso, os resultados obtidos no Capítulo 6.2.2 indicam que a qualidade do código das implementações que utilizaram a abordagem foi melhor do que as que não utilizaram, visto que, todas as métricas analisadas obtiveram resultados positivos.Assim, a hipótese Hb1 é verdadeira.

## 6.3 *HIBERNATE* X JDBC

Para avaliar a diferença entre as tecnologias Hibernate e JDBC não foi proposta uma Hipótese, mas uma questão de pesquisa: “Existe diferença na qualidade de código de persistência de dados Hibernate em relação ao JDBC, usando ou não a abordagem proposta? ”

**Resultados**

Na Figura 6.7, encontre-se um gráfico com uma comparação entre o tempo de implementação do Hibernate e JDBC. Neste gráfico pode-se notar que o *Hibernate* foi sempre mais rápido nas implementações, utilizando ou não a abordagem proposta neste trabalho. O *Hibernate*foi 15% mais rápido que o JDBC, referente ao tempo de desenvolvimento.

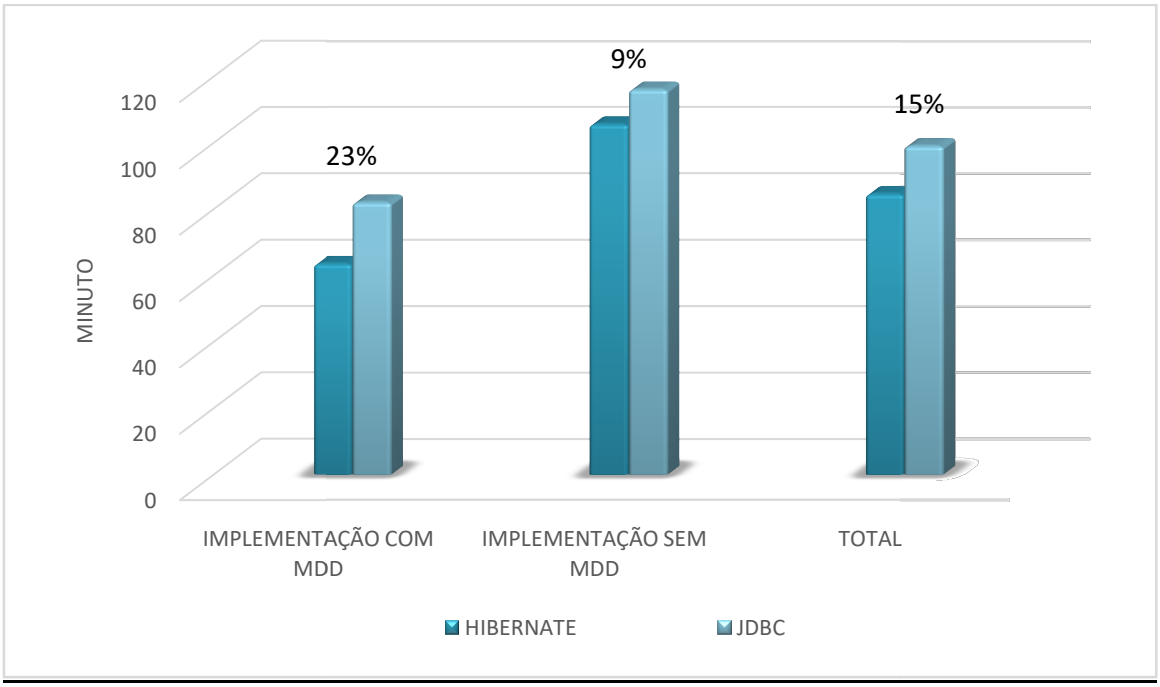



Figura 6.7 - Comparação de tempo de implementação entre *Hibernate* e JDBC

Na Tabela 6.3, é apresentado a comparação entre o *Hibernate* e JDBC, onde as métricas foram separadas por implementações com a abordagem proposta

e com abordagens tradicionais. Desta forma, foi-se necessário utilizar técnicas de estatísticas para unir os dados dos quatro participantes, para assim tornar a análise dos dados mais simples. As técnicas utilizadas foram: média, mediana e desvio padrão.

Tabela 6.3 – Comparação das métricas de qualidade entre *Hibernate* e JDBC

| | | COM ABORDAGEM | | SEM ABORDAGEM | |
|---|---|---|---|---|---|
| | | HIBERNATE | JDBC | HIBERNATE | JDBC |
| CDC | MÉDIA | 1,00 | 1,00 | 6,00 | 6,00 |
| | MEDIANA | 1,00 | 1,00 | 6,00 | 6,00 |
| | DESVIO PADRÃO | 0,00 | 0,00 | 0,00 | 0,00 |
| CDO | MÉDIA | 6,00 | 4,00 | 39,00 | 38,50 |
| | MEDIANA | 6,00 | 4,00 | 40,00 | 39,00 |
| | DESVIO PADRÃO | 0,00 | 0,00 | 2,83 | 1,00 |
| CDLOC | MÉDIA | 2,00 | 2,00 | 23,00 | 22,75 |
| | MEDIANA | 2,00 | 2,00 | 25,50 | 25,00 |
| | DESVIO PADRÃO | 0,00 | 0,00 | 6,16 | 6,13 |
| CE | MÉDIA | 2,28 | 2,11 | 2,45 | 2,25 |
| | MEDIANA | 2,25 | 2,18 | 2,53 | 2,25 |
| | DESVIO PADRÃO | 0,46 | 0,57 | 0,56 | 0,74 |
| CA | MÉDIA | 3,13 | 3,06 | 3,11 | 3,38 |
| | MEDIANA | 3,20 | 3,13 | 3,13 | 3,13 |
| | DESVIO PADRÃO | 0,41 | 0,51 | 0,27 | 0,60 |
| DIT | MÉDIA | 1,00 | 1,00 | 1,00 | 1,00 |
| | MEDIANA | 1,00 | 1,00 | 1,00 | 1,00 |
| | DESVIO PADRÃO | 0,00 | 0,00 | 0,00 | 0,00 |
| LCOO | MÉDIA | 0,36 | 0,39 | 0,35 | 0,36 |
| | MEDIANA | 0,36 | 0,39 | 0,34 | 0,34 |
| | DESVIO PADRÃO | 0,05 | 0,02 | 0,02 | 0,04 |
| LOC | MÉDIA | 571,75 | 716,75 | 644,00 | 681,75 |
| | MEDIANA | 569,00 | 686,00 | 613,00 | 647,50 |
| | DESVIO PADRÃO | 58,17 | 135,62 | 76,19 | 120,92 |
| NOA | MÉDIA | 28,75 | 35,75 | 34,25 | 32,25 |
| | MEDIANA | 29,50 | 35,50 | 33,50 | 32,00 |
| | DESVIO PADRÃO | 2,63 | 0,96 | 5,74 | 3,30 |
| WOA | MÉDIA | 103,00 | 124,50 | 113,00 | 116,75 |
| | MEDIANA | 102,50 | 118,50 | 115,00 | 112,50 |
| | DESVIO PADRÃO | 8,76 | 17,60 | 5,48 | 18,06 |

Sendo assim, após separar os dados de forma a torna-los mais compreensíveis, pode-se verificar através das métricas de separação de interesse que:

- **CDC:**não se teve diferença entre o *Hibernate* e o JDBC, tanto nas implementações com a abordagem quanto nas implementações sem elas.
- **CDO:**referente a métrica de CDO o *Hibernate* não foi superior ao JDBC;
- **CDLOC:** essa métrica também não se teve uma diferença do Hibernate com o JDBC ao utilizar a abordagem, entretanto, o Hibernate ficou inferior ao JDBC sem a utilização da abordagem.

Em relação as métricas de acoplamento e coesão, pode-se notar que:

- **CE e CA:** com relação a essas métrica, o *Hibernate* ficou inferior ao JDBC, em ambas as implementações: com e sem a abordagem.
- **DIT:** o resultado da análise entre *Hibernate* e JDBC não teve diferença com e sem abordagem.
- **LCOO:** referente a métrica de coesão o *Hibernate* não foi superior ao JDBC, ou seja, sua coesão é baixa tanto na utilização da abordagem quanto sem.

Nas métricas de tamanho, pode-se verificar que o *Hibernate*foi melhor que o JDBC tanto nas implementações que utilizaram a abordagem quanto nas implementações que não as utilizaram, deixando a desejar somente no número de atributos ao implementar sem a abordagem, onde o Hibernate ficou inferior ao JDBC.

Sendo assim, após a apresentação dos dados, uma discussão pode ser montada sobre eles. Nas implementações em que não foi utilizado a abordagem proposta, pôde-se avaliar que o Hibernate não é a melhor tecnologia a ser utilizada, ficando inferior ao JDBC baseado nas métricas de qualidade de código, porém, em tempo de implementação o *Hibernate* é melhor em 9% que o JDBC, já que uma vez ele reduz o esforço do programador, na questão de ter que montar uma *query* para cada regra de banco de dados.

Entretanto, pode-se notar que ao utilizar a abordagem proposta neste trabalho, o *Hibernate*passa a ser igual ou superior ao JDBC, deixando a desejar nas métricas de coesão e acoplamento. O tempo de desenvolvimento utilizando a

abordagem passa de 9% para 23% mais rápido que o JDBC.

# 7. CONSIDERAÇÕES FINAIS

Após apresentar os resultados obtidos com esta pesquisa, este capitulo posiciona a sua relevância, limitações e perspectivas de trabalhos futuros.

## 7.1 RELEVÂNCIA DO ESTUDO

No desenvolvimento de software, aplicações necessitam que as informações sejam armazenadas de forma persistente. Todavia, como há um grande número de métodos e ferramentas para este fim, a diversidade de opções não é apropriada, pois, na maioria das vezes transfere a responsabilidade pela escolha do método de persistência de dados aos desenvolvedores que em muitos casos não está apto para fazer a melhor escolha

Na literatura existem diversos estudos relacionados a este tema, em que tentam definir quais técnicas e ferramentas são apropriadas para cada tipo de situação. Os estudos existentes apenas descrevem em quais casos diferentes técnicas ou métodos são mais adequados. Entretanto, a escolha da técnica ou métodos é apenas um passo para implementação da persistência de dados. Sendo assim, uma alternativa pare resolver esta adversidade é a utilização do MDD.

O MDD auxilia a padronização e a utilização de códigos, pois a implementação dos códigos devem adotar um modelo de alto nível, assim, facilitando a manutenção de software, minimização de erros, além de agilizar o processo de entrega do produto final.

Sendo assim, o presente estudo visou propor uma abordagem que utiliza o MDD, pois, com sua utilização consegue-se evitar problemas recorrentes às aplicações de persistência de dados, além de mitigar contrapontos comuns no desenvolvimento de software.

Outra abordagem da engenharia de software, que pode estar correlacionada no desenvolvimento de aplicações de persistência de dados é a POA. A POA aumenta a modularidade, pois separa o código que implementam funções específicas, afetando diferentes partes do sistema (SOARES et. al. 2002).

Com o exposto, é demonstrado que o estudo corrente contribui no assunto em questão, de forma que ao utilizar a abordagem proposta aumenta a produtividade dos programadores e qualidade dos códigos gerados por eles.

## 7.2 CONTRIBUIÇÕES DA PESQUISA

A principal contribuição desta pesquisa é o próprio objetivo estabelecido no começo do trabalho, desenvolver uma abordagem para geração de códigos de persistência de dados Hibernatebaseado em MDD e POA. Para tanto, foi definido uma arquitetura para esta abordagem, detalhando suas camadas e seus componentes. Posteriormente, foi desenvolvida a abordagem de acordo com a sua arquitetura. Considerando o experimento realizado, verificou-se a eficácia desta abordagem baseado no tempo de desenvolvimento e na qualidade da codificação. Por fim, pôde-se observar que a abordagem proposta mitiga problemas encontrados na qualidade do código e o tempo de desenvolvimento de aplicações de persistência de dados Hibernate.

## 7.3 LIMITAÇÕES

A principal limitação da pesquisa está centrada no uso das aplicações necessárias para utilização do framework proposto, onde os participantes tiveram dificuldades em realizar os procedimentos necessários para gerar o modelo do sistema e quando conseguiam gerar, encontravam dificuldades em utilizar o Acceleo para gerar os códigos. Com isto, os participantes perderam muito tempo para se adequar ao uso do framework.

## 7.4 TRABALHOS FUTUROS

Por fim, são identificadas e expostas lacunas que o presente trabalho não conseguir suprir, de forma que possam ser tratados em trabalhos futuros.

O primeiro trabalho identificado é a implementação de uma ferramenta que auxilie no processo de criação do modelo e geração dos códigos de persistência. Além do programador poder escolher a tecnologia e o padrão de persistência que ele pretende usar. Desta forma, a ferramenta terá já armazenado os meta-modelos com seus respectivos *templates* de geração de códigos de persistência.

O segundo trabalho identificado é a criação de um framework juntando as tecnologias JPA e *Hibernate*, visto que ambas conseguem trabalhar em conjunto da maneira tradicional, ou seja, sem utilizar uma abordagem semelhante a este trabalho.

# REFERÊNCIAS

ACCELEO 2016. **The Eclipse Foundation**, 2016. Disponível em: <https://www.eclipse.org/acceleo/>. Acesso em 05 Jul. 2016.

BARCIA, R.; HAMBRICK, G.; BROWN, K.; PETERSON,b R.; BHOGAL, K. S. **Persistence in the Enterprise**: A Guide to Persistence Technologies, 1ª ed., IBM Press, 2008.

BARDIN, L. Análise de Conteúdo. Lisboa, Portugal; Edições 70, LDA, 2009.

BASILI V. **The Role of Experimentation in Software Engineering: Past, Present, Future**. IN: PROC. OF THE 18TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1(2), 1996, p. 133-164.

FERNANDES, J. E. M.; MACHADO, R. J.; CARVALHO, J. A. **Model-Driven Software Development for Pervasive Information Systems Implementation.** In Proceedings of the 6th International Conference on the Quality of Information and Communications Technology, IEEE, 2007, p. 218-222.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2006, p. 175

GIL, A.C. **Métodos e técnicas de pesquisa social**. 4 ed. São Paulo: Atlas, 1994. 207p.

HAILPERN, B; TARR, P. **Model-driven development**: The good, the bad, and the ugly. IBM Systems Journal. Riverton, NJ, USA, v. 45, n. 3, 2006, p. 451- 461.

HEITKÖTTER, H.; MAJCHRZAK, T.A.; KUCHEN, H**. Cross-Platform Model-Driven Development of Mobile Applications with MD2**, SAC' 13, 2013, p. 18-23.
Hibernate ORM. **Object/Relational Mapping**. Disponível em: <http://hibernate.org/orm/>. Acesso em: 06 jul. 2016.

JIANG, K.; ZHANG, L.; MIYAKE, S. **Using OCL in executable UML**. Easst, Beijing, v. 9, 2007, p. 1-12.

KENT, S. **Model-driven engineering**. Proc. 34rd Int. Conf. on Integrated Formal Methods, 2002, p. 286-298;

KICZALES, G. et al. **Aspect -Oriented Programming**. European Conference on Object - Oriented Programming (ECOOP), LNCS (1241), Springer-Verlag, Finland. 1997

KOCH, N. **Classification of model transformation techniques used in UMLbased Web engineering**. Software, IET. v. 1, n. 3, p. 98-111, 2007.

KONDA, Madhusdhan. **Introdução ao Hibernate**. São Paulo: Novatec, 2014.

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para WEB**. 2. ed. São Paulo: Novatec, 2015.

MAFRA, S. N; TRAVASSOS, G. H. **Estudos Primários e Secundários apoiando a busca por Evidência em Engenharia de Software**. Programa de Engenharia de Sistemas e Computação COPPE/UFRJ, 2006.

MEYER, B. **Object–Oriented Software Construction**. Prentice–Hall, second edition, 1997.

MIGUEL, P. A. C. **Estudo de Caso na engenharia de produção: estruturação e recomendações para sua condução**. Produção, v.17, n.1, 2007, p. 216-229.

MIZUNO, T.; MATSUMOTO, K.; MORI, N. **A ModelDriven Development Method for Management Information Systems, Electronics and Communications**. In Japan, 2010, p. 96, 2, 245-252.

MTSWENI, J. **Exploiting UML and Acceleo for Developing Semantic Web Services**. In: The 7th International Conference for Internet Technology and Secured Transactions (ICITST-2012). Pretoria, South Africa, 2012.

POTHU, S. **A Comparative Analysis of Object-relational mappings for Java**, MSc. Thesis, University of Applied Sciences at Braunschweig/Wolfenbuettel, 2008.

REINEHR, S. Reúso **Sistematizado de Software e Linhas de Produto de Software No Setor Financeiro**: Estudos De Caso No Brasil. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo (USP), São Paulo, 2008, 310p.

SANT'ANNA, C.; GARCIA, A.; CHAVEZ, C.; LUCENA, C.; STAA, A. von. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. Proc. of Brazilian Symposium on SoftwareEngineering (SBES'03), Manaus, Brazil, Oct 2003, 19-34.

SANT'ANNA, Cláudio, et al. **On the reuse and maintenance of aspect-oriented software**: An assessment framework. Proceedings of Brazilian symposium on software engineering, Brazil. 2003.

SCHMIDT, D. C. **Model-Driven Engineering**. IEEE Computer. V. 39, n. 2, 2006, p 25-31.

SELIC, B. **The pragmatics of model-driven development**. IBM Rational Software. v. 20, n. 5, 2008, p. 19-25.

SINGH, Y.; SOOD, M. **Model Driven Architecture**: A Perspective in 2009 IEEE International Advance Computing Conference, Patiala, India, 2009, p. 6-7.

SOARES, S. "An Aspect-Oriented Implementation Method". Doctoral Thesis, Federal Univ. of Pernambuco, Oct 2004.

SOARES, S. et. al**. Implementing distribution and persistence aspects with AspectJ**. ACM. In Proceedings of the OOPSLA' 2002, Seattle, USA, 2002, p. 174-190.

SOARES, Sergio; BORBA, Paulo**. AspectJ—Programação orientada a aspectos em Java**, Brazilian Syposium of Language Programming, SBLP, Rio de Janeiro, 2002.

STAHL, T.; VOELTER, M. **Model-Driven Software Development**: Technology, Engineering, Management, Wiley, West Sussex. 2016.

TARR, P. et al. **N Degrees ofSeparation**: Multi Dimensional Separation of Concerns. Proceedings of the 21st International Conference on Software Engineering. 1999.

VIDYAPEETHAM, A. V. **An eclipse-based tool for modeling service-based systems**. 2009. 49p. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – Amrita School of Engineering, Coimbatore, Tamil Nadu, India, 2009.

WOHLIN C, RUNESON P, HÖST M, OHLSSON MC, REGNELL B, WESSLÉN A (2012) **Experimentation in software engineering**. Springer, ISBN 978-3-642-29043-5

# APÊNDICE A – ECORE/XML NO NOSSO META-MODELO

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="generichibernatedaopoa" nsURI="http://generichibernatedaopoa/1.0"
    nsPrefix="generichibernatedaopoa">
  <eClassifiers xsi:type="ecore:EClass" name="Init">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="package" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hibernateUtil" eType="#//HibernateUtil"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="aspectDao" eType="#//AspectDao"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="control" upperBound="-1"
        eType="#//Control" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="HibernateUtil">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="driver" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="URL" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="user" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="password" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="AspectDao">
    <eOperations name="getSessionFactory"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Control">
    <eOperations name="insert"/>
    <eOperations name="update"/>
    <eOperations name="delete"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="package" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="objectDAO" upperBound="-1"
        eType="#//ConjuntoObjectDAO" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Object">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="package" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="atributo" upperBound="-1"
        eType="#//Atributos" containment="true"/>
  </eClassifiers>


  <eClassifiers xsi:type="ecore:EClass" name="GenericDao" abstract="true" interface="true">
    <eOperations name="insert"/>
    <eOperations name="update"/>
    <eOperations name="delete"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="package" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DaoClass" eSuperTypes="#//GenericDao"/>
  <eClassifiers xsi:type="ecore:EClass" name="Atributos">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="associacao" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBooleanObject">
      <eAnnotations source="http:///org/eclipse/emf/ecore/util/ExtendedMetaData"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ConjuntoObjectDAO">
    <eStructuralFeatures xsi:type="ecore:EReference" name="object" lowerBound="1"
        eType="#//Object" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="genericDao" lowerBound="1"
        eType="#//GenericDao" containment="true"/>
  </eClassifiers>
</ecore:EPackage>
```

# APÊNDICE B - TEMPLATE ACCELEO PARA GERAÇÃO DE CÓDIGO

```
[comment encoding = UTF-8 /]
[module generate('http://generichibernatedaopoa/1.0')]

[template public generateElement(anInit : Init)]
[comment @main /]
[file (anInit._package.replaceAll('\\.', '/').concat('/').concat(anInit.className).concat('.java'), false, 'UTF-8')]
package [anInit._package/];

import [anInit.control._package/].[anInit.control.className /];

public class [anInit.className /] {
private [anInit.control.className /] control;

public static void main(String['['/']'/] args) throws Exception {
        // performs anything beautifully
    }
}
[/file]
[for (controle : Control | anInit.control)]
[comment Control. /]
[file (controle._package.replace('\\.', '/').concat('/').concat(controle.className.toUpperFirst()).concat('.java'), false, 'UTF-8')]
package [controle._package /];
import org.hibernate.HibernateException;
[for (objeto : ConjuntoObjectDAO | controle.objectDAO)]
import [objeto.object._package.concat('.').concat(objeto.object.className)/];
[/for]

public class [controle.className /] {

}
[/file]
[comment objeto. /]
[for (objeto : ConjuntoObjectDAO | controle.objectDAO) ]
```

```
[file (objeto.object._package.replace('\\.', '/').concat('/').concat(objeto.object.className.toUpperFirst()).concat('.java'), false, 'UTF-8')]
package [objeto.object._package /];
public class [objeto.object.className /] {
    private [objeto.object.type.concat(' ').concat(objeto.object.id)/];
    [for (atributo : Atributos | objeto.object.atributo)]
    private [atributo.type.concat(' ').concat(atributo.name)/];
    [/for]

    public [objeto.object.type/] get[objeto.object.id.toUpperFirst()/](){
        return [objeto.object.id/];
    }

    public [objeto.object.type/] getId(){
        return [objeto.object.id/];
    }

    public void set[objeto.object.id.toUpperFirst()/]([objeto.object.type.concat(' ').concat(objeto.object.id)/]){
        this.[objeto.object.id/] = [objeto.object.id/];
    }

    [for (atributo : Atributos | objeto.object.atributo)]
    public [atributo.type/] get[atributo.name.toUpperFirst()/](){
        return [atributo.name/];
    }

    public void set[atributo.name.toUpperFirst()/]([atributo.type.concat(' ').concat(atributo.name)/]){
        this.[atributo.name/] = [atributo.name/];
    }
    [/for]
}
[/file]
[comment objeto.xml. /]
[file (objeto.object._package.replace('\\.', '/').concat('/').concat(objeto.object.className.toUpperFirst()).concat('.hbm.xml'), false, 'UTF-8')]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="[objeto.object._package.concat('.').concat(objeto.object.className.toUpperFirst())/]" table="[objeto.object.className.toLower()/]">
        <id name="[objeto.object.id/]" column="[objeto.object.id/]" />
        [for (atributo : Atributos | objeto.object.atributo)]
            [if (atributo.associacao = true)]
                <property name="[atributo.name/]" column="[atributo.name/]" />
            [/if]

        [/for]
    </class>
</hibernate-mapping>
[/file]

[comment Dao. /]
[file (objeto.genericDao._package.replace('\\.', '/').concat('/').concat(objeto.genericDao.className.toUpperFirst()).concat('.java'), false, 'UTF-8')]
package [objeto.genericDao._package /];

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import [objeto.object._package.concat('.').concat(objeto.object.className)/];
public class [objeto.genericDao.className /] implements GenericDAO<[objeto.object.className/]> {


    public void insert([objeto.object.className/] o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove ([objeto.object.className/] o) throws HibernateException{
       // Sets the query for removing 'o'.
    }
```

```
    public void update([objeto.object.className/] o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public [objeto.object.className/] findById([objeto.object.className/] o, String coluna, Object id) throws HibernateException{
        return o;
    }
    public ArrayList<[objeto.object.className/]> findAll() throws HibernateException{
        return null;
    }
}
[/file]
[/for]
[/for]

[comment GenericDao. /]
[file ('util/GenericDAO.java', false, 'UTF-8')]
package util;

import org.hibernate.HibernateException;
import java.util.ArrayList;

public interface GenericDAO<T> {

    public void insert(T o) throws HibernateException;

    public void remove(T o) throws HibernateException;

    public void update(T o) throws HibernateException;

    public T findById(T o, String column, Object id) throws HibernateException;

   public ArrayList<T> findAll() throws HibernateException;

}
[/file]

[comment Aspect /]
[file ('util/AspectDAO.aj', false, 'UTF-8')]
package util;

import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Restrictions;


@SuppressWarnings("deprecation")
public aspect AspectDAO{

    private SessionFactory sessionFactory;

    private SessionFactory hibernateUtil() {
        if (sessionFactory == null) {
            try {
                // Create the SessionFactory from standard (hibernate.cfg.xml)
                // config file.
                sessionFactory = new AnnotationConfiguration().configure()
                        .buildSessionFactory();
                return sessionFactory;
            } catch (Throwable ex) {
                // Log the exception.
                System.err.println("Initial SessionFactory creation failed."
                        + ex);
                throw new ExceptionInInitializerError(ex);
            }
        }else{
            return sessionFactory;
        }

    }

    @SuppressWarnings("rawtypes")
    pointcut insert(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.insert(Object));

    @SuppressWarnings("rawtypes")
    pointcut remove(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.remove(Object));

    @SuppressWarnings("rawtypes")
    pointcut update(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.update(Object));

    @SuppressWarnings("rawtypes")
    pointcut findById(GenericDAO x, Object o, String column, Object i) : target(x) && args(o, column, i) && execution(public Object GenericDAO.findBy

    @SuppressWarnings("rawtypes")
    pointcut findAll(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.findAll(Object));


    @SuppressWarnings("rawtypes")
    after(GenericDAO x, Object o) returning() : insert(x, o){
        Session session = hibernateUtil().openSession();
        try{
        session.beginTransaction();
        session.save(o);
        session.getTransaction().commit();
```

```
        }catch(HibernateException ex){
            session.getTransaction().rollback();
            ex.printStackTrace();
        }finally{
            session.flush();
            session.close();
        }
    }
    //public void execute()
    @SuppressWarnings("rawtypes")
    after(GenericDAO x, Object o) returning() : remove(x, o){
        Session session = hibernateUtil().openSession();
        try{
        session.beginTransaction();
        session.delete(o);
        session.getTransaction().commit();
        }catch(HibernateException ex){
            session.getTransaction().rollback();
            ex.printStackTrace();
        }finally{
            session.flush();
            session.close();
        }
    }
    @SuppressWarnings("rawtypes")
    after(GenericDAO x, Object o) returning() : update(x, o){
        Session session = hibernateUtil().openSession();
        try{
        session.beginTransaction();
        session.update(o);
        session.getTransaction().commit();
        }catch(HibernateException ex){
            session.getTransaction().rollback();
```

```
                ex.printStackTrace();
            }finally{
                session.flush();
                session.close();
            }
        }

        @SuppressWarnings("rawtypes")
        Object around(GenericDAO x, Object o, String y, Object d) : findById(x, o, y, d) {
            Session session = hibernateUtil().openSession();
            Object object = null;
            try {
                Criteria criteria = session.createCriteria(o.getClass());
                Criterion name = Restrictions.eq(y, d);
                criteria.add(name);
                session.beginTransaction();
                object = criteria.uniqueResult();
                session.getTransaction().commit();
            } catch (HibernateException ex) {
                session.getTransaction().rollback();
                ex.printStackTrace();
            } finally {
                session.flush();
                session.close();
            }
            return object;
        }

}
[/file]

[comment hibernate./]
[file ('hibernate.cfg.xml', false, 'UTF-8')]
```

```
                ex.printStackTrace();
            } finally {
                session.flush();
                session.close();
            }
            return object;
        }

}
[/file]

[comment hibernate./]
[file ('hibernate.cfg.xml', false, 'UTF-8')]
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">[anInit.hibernateUtil.driver/]</property>
        <property name="hibernate.connection.url">[anInit.hibernateUtil.URL/]</property>
        <property name="hibernate.connection.username">[anInit.hibernateUtil.user/]</property>
        <property name="hibernate.connection.password">[anInit.hibernateUtil.password/]</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <!--Mapeamento-->
        [for (controle : Control | anInit.control)]
        [for (objeto : ConjuntoObjectDAO | controle.objectDAO)]
        <mapping resource="[objeto.object._package.concat('/').concat(objeto.object.className)/].hbm.xml"/>
        [/for]
        [/for]
    </session-factory>
</hibernate-configuration>
[/file]
[/template]
```

# APÊNDICE C –TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

**Uma abordagem para geração de código de persistência de dados hibernate baseado em MDD e POA**

Pesquisa realizada pelaEngenharia de Software e Gestão do Conhecimento CCTda UENP, desenvolvida para o estudo da abordagem de geração de códigos de persistência dados Hibernate baseado em MDD e POA, para o Trabalho de Conclusão de Curso em Bacharelado em Ciência da Computação da Universidade Estadual do Norte do Paraná (UENP), pelo discente Luan de Souza Melo(luan.sm50@gmail.com) e orientador Prof. André Menolli.

Por meio deste Termo de Confidencialidade, os Pesquisadores se comprometem a:

- Portar-se com discrição em todos os momentos da pesquisa acadêmica, não comentando ou divulgando qualquer tipo de informação que tenha sido repassada de forma oral ou escrita;
- Não divulgar nome do participante, em qualquer meio, a menos que expressamente autorizado por este;
- Não divulgar, em qualquer meio, os dados e informações individualizadas coletados durante o processo de pesquisa com o Participante;
- Divulgar, em formato de tese, artigos e apresentações, apenas os dados agregados, dos quais não se possa retirar ou inferir a identificação do Participante.

*Eu,* _____,

*após ter lido e entendido as informações e esclarecido todas as minhas dúvidas referentes a este estudo com o Professor André Luís Andrade Menolli,* ***CONCORDO VOLUNTARIAMENTE****, em participar do mesmo.*

_____ Data: ____/____/_____

Assinatura (do pesquisado ou responsável) ou impressão datiloscópica

Eu, Prof. André Luís Andrade Menolli, declaro que forneci todas as informações referentes ao estudo ao participante.

_____ Data: ____/____/_____

Assinatura

## Equipe:

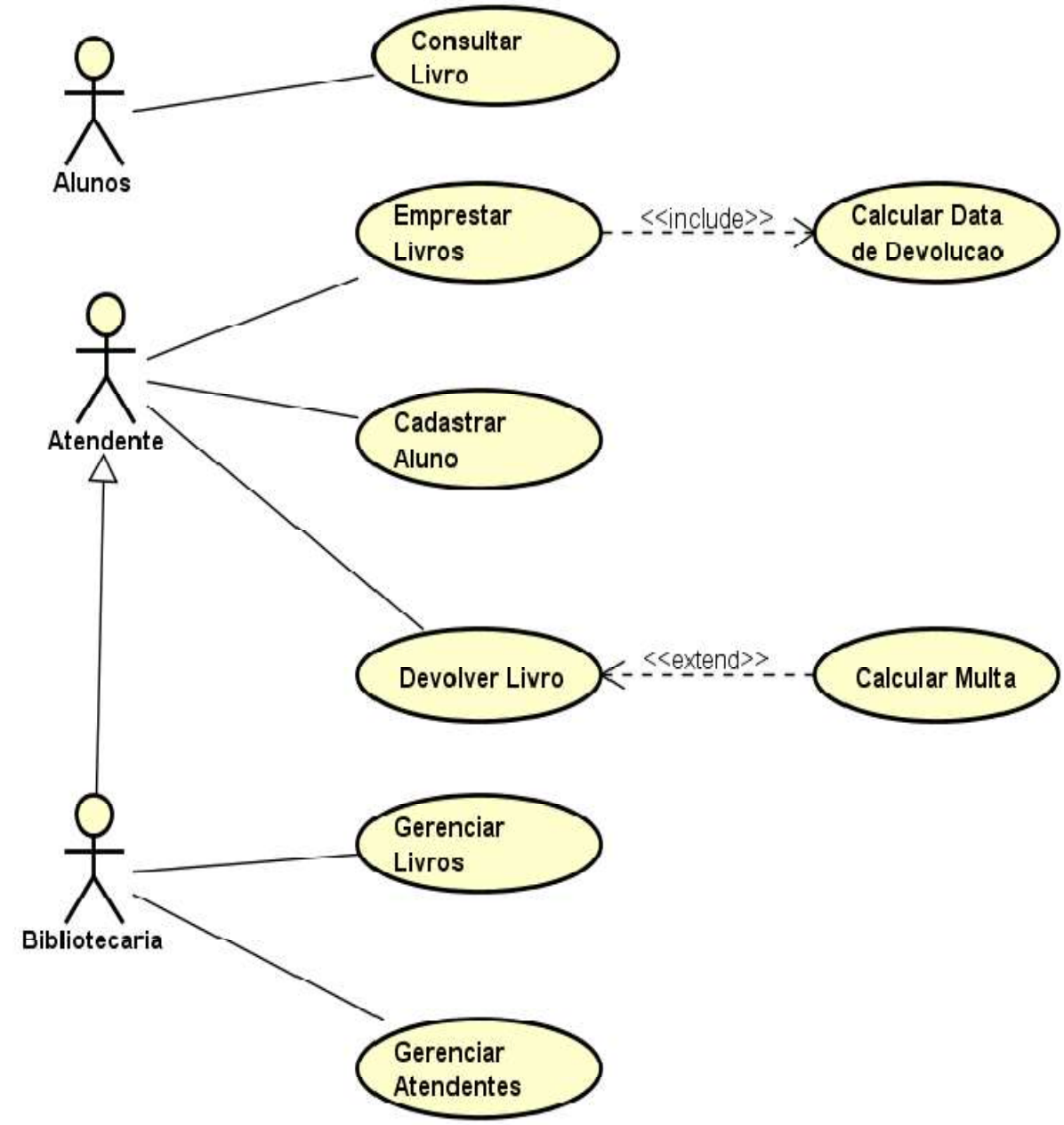
1- Nome: André Luís A. Menolli Telefone: (43) 35428014
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.

2- Nome: Luan de Souza Melo Telefone:(43) 996106734
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.

3- Nome: Felipe Igawa Moskado Telefone: (43) 999791318
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.

# APÊNDICE D – CASO DE USO

Estamos implementando um sistema para gerenciamento de empréstimo de livros em uma biblioteca. O sistema tem 3 tipos de usuários, a bibliotecárias, os atendentes e os alunos. A bibliotecária é responsável pelas funções gerenciais dos sistemas, os atendentes pela função de empréstimo e os alunos apenas podem realizar consultas ao sistema. O sistema está na versão 1.0 e tem apenas funcionalidades básicas como mostrado no diagrama de caso de uso abaixo:

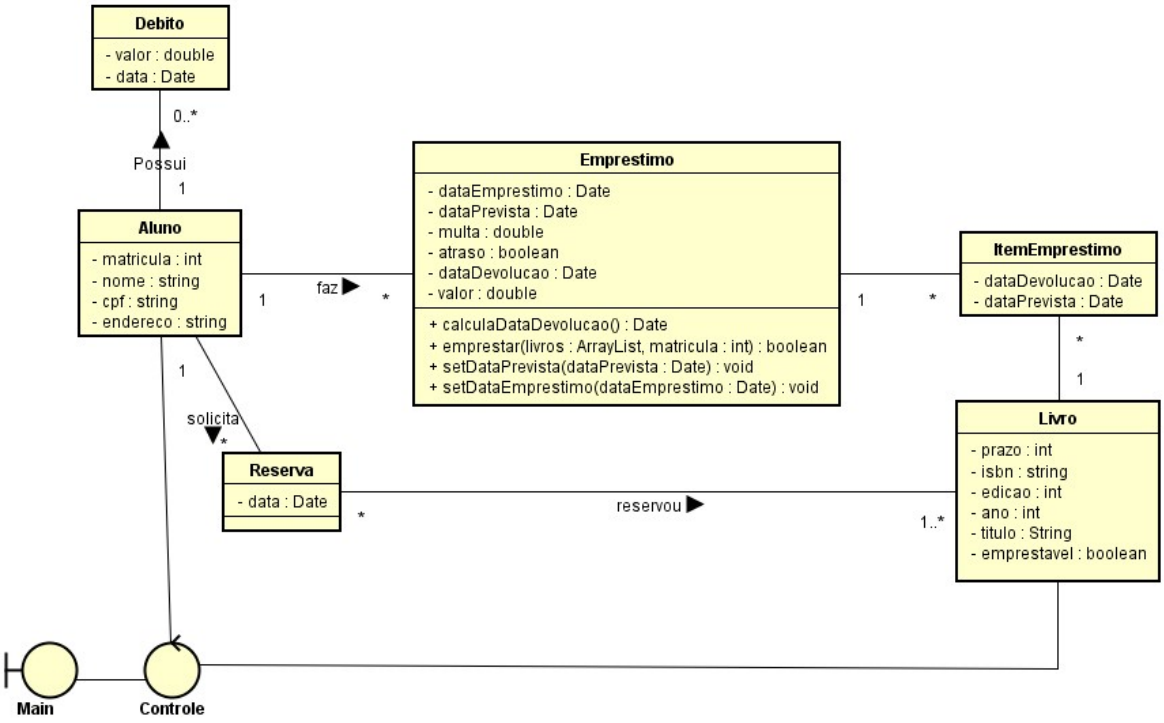


**Requisitos descritivos do Casos de Uso**

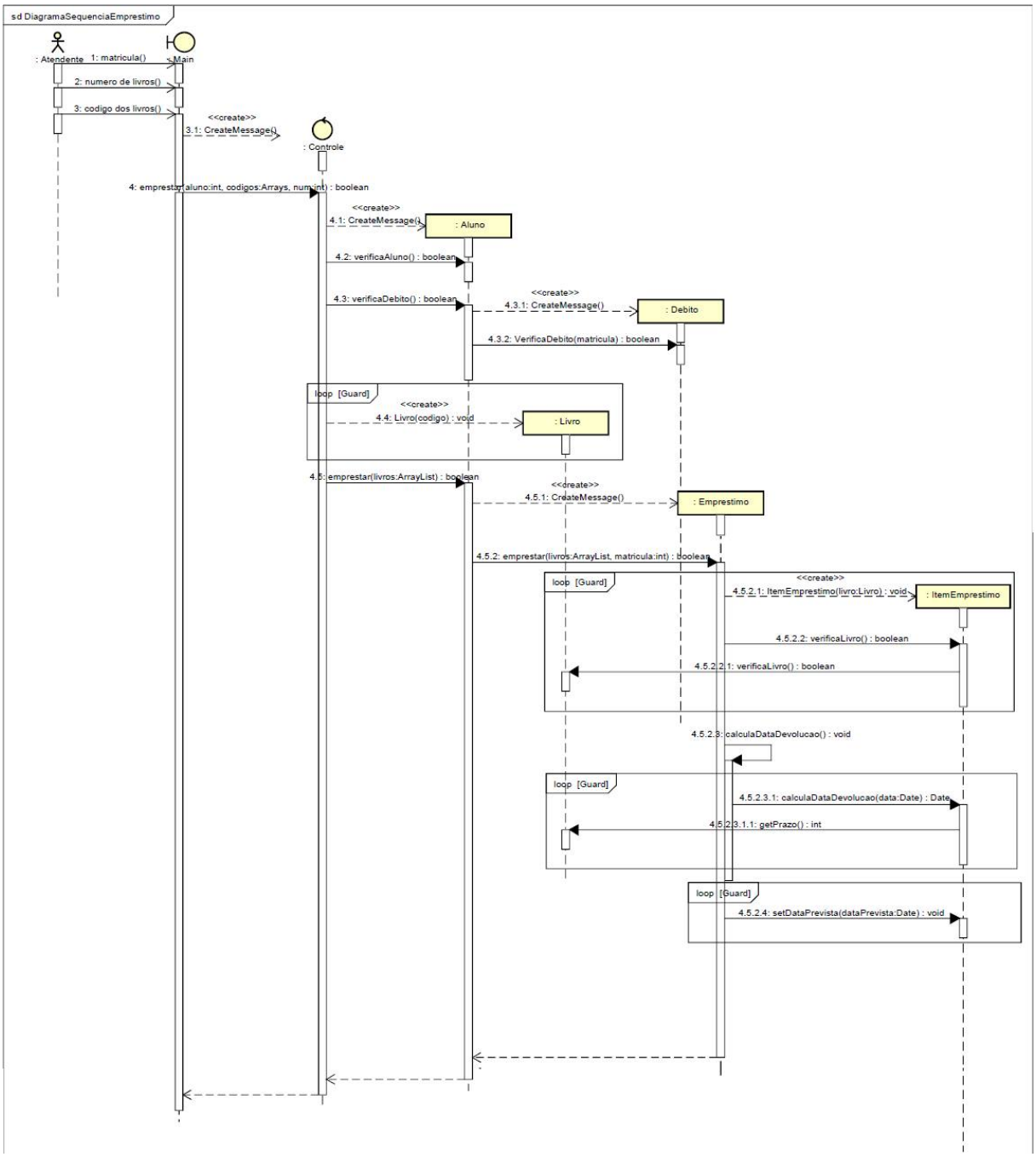
<table>
<tr><th colspan="2">Caso de Uso: Emprestar Livro</th></tr>
<tr><th>Fluxo Principal</th><th>Fluxo Alternativo</th></tr>
<tr><td rowspan="3">Fluxo de Principal (caminho básico):<br>1. O aluno apresenta os livros ao funcionário e a sua identificação.<br>2. O funcionário insere a identificação e os livros no sistema.<br>3. O sistema verifica se o Aluno está cadastrado.<br>4. O sistema verifica se o Aluno possui pendências.<br>5. O sistema cria um empréstimo.<br>6. Para cada livro:<br>6.1. O sistema verifica se não é exemplar que não pode ser emprestado.<br>6.2. O sistema cria um item de empréstimo.<br>6.3. O sistema associa o livro ao item.<br>7. O sistema calcula a data de devolução (Ponto de inclusão Calcula Data Devolução)<br>8. O sistema grava os dados do empréstimo.<br>9. O sistema imprime os dados do empréstimo.</td><td>3.a Aluno não cadastrado<br>3.a.1 – O sistema informa que o aluno não está cadastrado.<br>3.a.2 – O sistema finaliza o caso de uso.</td></tr>
<tr><td>4.a Aluno possui débito<br>4.a.1 – O sistema informa que o aluno está em débito.<br>4.a.2 – O sistema finaliza caso de uso.</td></tr>
<tr><td>6.1.a Livro reservado<br>6.1.a.1 – O sistema informa que o livro está reservado e não pode ser emprestado.<br>6.1.a.2 – O sistema informa a data de devolução do livro.<br>6.1.a.3 – Retorna ao passo 6.</td></tr>
</table>

| **Caso de uso: Calcula Data de Devolução** | |
|---|---|
| **Fluxo Principal** | **Fluxo Alternativo** |
| 1. Pega o número de livros do empréstimo.<br>1.1. Pega o prazo de devolução de cada livro.<br>1.2. Calcula a data de devolução do livro.<br>2. Selecione o maior prazo dentre todos os livros.<br>3. Retorna a data de devolução dos livros. | 2.a. **Caso o cliente empreste 3 ou mais livros**<br>2.a.1 – Adiciona mais dois dias para cada livro após o 2º livro emprestado.<br>2.a.2 – Calcula a nova data.<br>2.a.3 – Retorna ao passo 3. |

# APÊNDICEE - DIAGRAMA DE CLASSES



# APÊNDICE F - DIAGRAMA DE SEQUÊNCIA DA FUNCIONALIDADE EMPRESTAR



# APÊNDICE G – SCRIPT PARA GERAÇÃO DA BASE DE DADOS RELACIONAL DO PROJETO EM MYSQL

```
CREATE DATABASE `biblioteca` /*!40100 DEFAULT CHARACTER SET utf8 */;

CREATE TABLE `aluno` (
  `matricula` int(11) NOT NULL,
  `nome` varchar(45) DEFAULT NULL,
  `cpf` varchar(45) DEFAULT NULL,
  `endereco` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`matricula`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `debito` (
  `idDebito` int(11) NOT NULL,
  `matricula` int(11) DEFAULT NULL,
  `valor` double DEFAULT NULL,
  `data` date DEFAULT NULL,
  PRIMARY KEY (`idDebito`),
  KEY `matricula_idx` (`matricula`),
  CONSTRAINT `matricula` FOREIGN KEY (`matricula`) REFERENCES `aluno` (`matricula`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `emprestimo` (
  `idemprestimo` int(11) NOT NULL,
  `dataPrevista` date DEFAULT NULL,
  `dataDevolucao` date DEFAULT NULL,
  `dataEmprestimo` date DEFAULT NULL,
  `multa` double DEFAULT NULL,
  `atraso` tinyint(4) DEFAULT NULL,
  `valor` double DEFAULT NULL,
  PRIMARY KEY (`idemprestimo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `livro` (
  `idlivro` int(11) NOT NULL,
  `titulo` varchar(45) DEFAULT NULL,
  `ISBN` varchar(45) DEFAULT NULL,
  `edicao` varchar(45) DEFAULT NULL,
  `ano` date DEFAULT NULL,
  `emprestavel` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`idlivro`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `itememprestimo` (
  `idItemEmprestimo` int(11) NOT NULL,
  `idEmprestimo` int(11) DEFAULT NULL,
  `dataPrevista` date DEFAULT NULL,
  `dataDevolucao` date DEFAULT NULL,
  `idLivro` int(11) DEFAULT NULL,
  PRIMARY KEY (`idItemEmprestimo`),
  KEY `idemprestimo_idx` (`idEmprestimo`),
  KEY `idlivro_idx` (`idLivro`),
  CONSTRAINT `idemprestimo` FOREIGN KEY (`idEmprestimo`) REFERENCES `emprestimo` (`idemprestimo`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `idlivro` FOREIGN KEY (`idLivro`) REFERENCES `livro` (`idlivro`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

# APÊNDICE H – ARTEFATOS GERADOS PELO SISTEMA

## Aluno.java

```
package modelo;

public class Aluno {
    private Integer matricula;
    private String nome;
    private String cpf;
    private String endereco;


    public Integer getMatricula(){
        return matricula;
    }

    public Integer getId(){
        return matricula;
    }

    public void setMatricula(Integer matricula){
        this.matricula = matricula;
    }

    public String getNome(){
        return nome;
    }

    public void setNome(String nome){
        this.nome = nome;
    }
    public String getCpf(){
        return cpf;
    }

    public void setCpf(String cpf){
        this.cpf = cpf;
    }
    public String getEndereco(){
        return endereco;
    }

    public void setEndereco(String endereco){
        this.endereco = endereco;
    }
}
```

## Aluno.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="modelo.Aluno" table="aluno">
        <id name="matricula" column="matricula" >
            <generator class="native"></generator>
        </id>
        <property name="nome" column="nome" />
        <property name="cpf" column="cpf" />
        <property name="endereco" column="endereco" />
    </class>
</hibernate-mapping>
```

Debito.java

```java
package modelo;

import java.util.Date;

public class Debito {
    private Integer idDebito;
    private Double valor;
    private Date data;
    private Aluno aluno;

    public Integer getIdDebito(){
        return idDebito;
    }

    public Integer getId(){
        return idDebito;
    }

    public void setIdDebito(Integer idDebito){
        this.idDebito = idDebito;
    }

    public Double getValor(){
        return valor;
    }

    public void setValor(Double valor){
        this.valor = valor;
    }
    public Date getData(){
        return data;
    }

    public void setData(Date data){
        this.data = data;
    }
    public Aluno getAluno(){
        return aluno;
    }

    public void setAluno(Aluno aluno){
        this.aluno = aluno;
    }
}
```

Debito.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="modelo.Debito" table="debito">
        <id name="idDebito" column="idDebito" >
            <generator class="native"/>
        </id>
        <property name="valor" column="valor" />
        <property name="data" column="data" />
        <many-to-one name="aluno" class="modelo.Aluno" fetch="select">
            <column name="matricula" not-null="true"/>
        </many-to-one>

    </class>
</hibernate-mapping>
```

## Emprestimo.java

```java
package modelo;

import java.util.Date;
import java.util.Set;

public class Emprestimo {
    private Integer idemprestimo;
    private Date dataEmprestimo;
    private Date dataPrevista;
    private Double multa;
    private Boolean atraso;
    private Date dataDevolucao;
    private Double valor;
    private Aluno aluno;
    private Set<ItemEmprestimo> itememprestimos;

    public Integer getIdemprestimo(){
        return idemprestimo;
    }

    public Integer getId(){
        return idemprestimo;
    }

    public void setIdemprestimo(Integer idemprestimo){
        this.idemprestimo = idemprestimo;
    }

    public Date getDataEmprestimo(){
        return dataEmprestimo;
    }

    public void setDataEmprestimo(Date dataEmprestimo){
        this.dataEmprestimo = dataEmprestimo;
    }
    public Date getDataPrevista(){
        return dataPrevista;
    }

    public void setDataPrevista(Date dataPrevista){
        this.dataPrevista = dataPrevista;
    }
    public Double getMulta(){
        return multa;
    }
    public void setMulta(Double multa){
        this.multa = multa;
    }
    public Boolean getAtraso(){
        return atraso;
    }

    public void setAtraso(Boolean atraso){
        this.atraso = atraso;
    }
    public Date getDataDevolucao(){
        return dataDevolucao;
    }

    public void setDataDevolucao(Date dataDevolucao){
        this.dataDevolucao = dataDevolucao;
    }
    public Double getValor(){
        return valor;
    }

    public void setValor(Double valor){
        this.valor = valor;
    }
    public Aluno getAluno(){
        return aluno;
    }

    public void setAluno(Aluno aluno){
        this.aluno = aluno;
    }

    public Set<ItemEmprestimo> getItememprestimos() {
        return itememprestimos;
    }

    public void setItememprestimos(Set<ItemEmprestimo> itememprestimos) {
        this.itememprestimos = itememprestimos;
    }


}
```

## Emprestimo.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="modelo.Emprestimo" table="emprestimo">
        <id name="idemprestimo" column="idemprestimo" >
            <generator class="native"/>
        </id>
        <property name="dataEmprestimo" column="dataEmprestimo" />
        <property name="dataPrevista" column="dataPrevista" />
        <property name="multa" column="multa" />
        <property name="atraso" column="atraso" />
        <property name="dataDevolucao" column="dataDevolucao" />
        <property name="valor" column="valor" />
        <set name="itememprestimos" table="itememprestimo" inverse="true" lazy="true" fetch="select" cascade="persist">
            <key>
                <column name="idEmprestimo" not-null="true"/>
            </key>
            <one-to-many class="modelo.ItemEmprestimo"/>
        </set>


    </class>
</hibernate-mapping>
```

ItemEmprestimo.java

```java
package modelo;

import java.util.Calendar;
import java.util.Date;

import dao.ItemEmprestimoDAO;

public class ItemEmprestimo {
    private Integer idItemEmprestimo;
    private Date dataDevolucao;
    private Date dataPrevista;
    private Emprestimo emprestimo;
    private Livro livro;

    public void save(){
        ItemEmprestimoDAO itemEmprestimoDAO = new ItemEmprestimoDAO();
        itemEmprestimoDAO.insert(this);
    }

    public Date calculaDataDevoulcao(Date data){
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(data);
        calendar.add(Calendar.DATE, livro.getPrazo());
        return calendar.getTime();
    }
    public Integer getIdItemEmprestimo(){
        return idItemEmprestimo;
    }

    public Integer getId(){
        return idItemEmprestimo;
    }

    public void setIdItemEmprestimo(Integer idItemEmprestimo){
        this.idItemEmprestimo = idItemEmprestimo;
    }

    public Date getDataDevolucao(){
        return dataDevolucao;
    }

    public void setDataDevolucao(Date dataDevolucao){
        this.dataDevolucao = dataDevolucao;
    }
    public Date getDataPrevista(){
        return dataPrevista;
    }

public void setDataPrevista(Date dataPrevista){
    this.dataPrevista = dataPrevista;
}
public Emprestimo getEmprestimo(){
    return emprestimo;
}

public void setEmprestimo(Emprestimo emprestimo){
    this.emprestimo = emprestimo;
}
public Livro getLivro(){
    return livro;
}

public void setLivro(Livro livro){
    this.livro = livro;
}
```

ItemEmprestimo.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="modelo.ItemEmprestimo" table="itememprestimo">
        <id name="idItemEmprestimo" column="idItemEmprestimo" >
            <generator class="native"/>
        </id>
        <property name="dataDevolucao" column="dataDevolucao" />
        <property name="dataPrevista" column="dataPrevista" />
        <many-to-one name="emprestimo" class="modelo.Emprestimo" fetch="select" cascade="persist">
            <column name="idEmprestimo" not-null="true" ></column>
        </many-to-one>
        <many-to-one name="livro" class="modelo.Livro" fetch="select">
            <column name="idLivro" not-null="true"></column>
        </many-to-one>
    </class>
</hibernate-mapping>
```

## Livro.java

```java
package modelo;

public class Livro {
    private Integer idlivro;
    private Integer prazo;
    private String isbn;
    private Integer edicao;
    private Integer ano;
    private String titulo;
    private Boolean emprestavel;

    public Integer getIdlivro(){
        return idlivro;
    }

    public Integer getId(){
        return idlivro;
    }

    public void setIdlivro(Integer idlivro){
        this.idlivro = idlivro;
    }

    public Integer getPrazo(){
        return prazo;
    }

    public void setPrazo(Integer prazo){
        this.prazo = prazo;
    }
    public String getIsbn(){
        return isbn;
    }

    public void setIsbn(String isbn){
        this.isbn = isbn;
    }
    public Integer getEdicao(){
        return edicao;
    }

    public void setEdicao(Integer edicao){
        this.edicao = edicao;
    }
```

```java
public Integer getAno(){
    return ano;
}

public void setAno(Integer ano){
    this.ano = ano;
}
public String getTitulo(){
    return titulo;
}

public void setTitulo(String titulo){
    this.titulo = titulo;
}
public Boolean getEmprestavel(){
    return emprestavel;
}

public void setEmprestavel(Boolean emprestavel){
    this.emprestavel = emprestavel;
}
```

## Livro.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="modelo.Livro" table="livro">
        <id name="idlivro" column="idlivro" >
            <generator class="native"/>
        </id>
        <property name="prazo" column="prazo" />
        <property name="isbn" column="isbn" />
        <property name="edicao" column="edicao" />
        <property name="ano" column="ano" />
        <property name="titulo" column="titulo" />
        <property name="emprestavel" column="emprestavel" />
    </class>
</hibernate-mapping>
```

# Controle.java

```
package controle;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashSet;
import dao.AlunoDAO;
import dao.DebitoDAO;
import dao.EmprestimoDAO;
import dao.ItemEmprestimoDAO;
import dao.LivroDAO;
import modelo.Debito;
import modelo.Aluno;
import modelo.Emprestimo;
import modelo.ItemEmprestimo;
import modelo.Livro;

public class ControleEmprestimo {
    Emprestimo emprestimo;
    public Boolean empresta(Integer matricula, ArrayList<Integer> codigos, Integer numLivro){
        Aluno aluno = new AlunoDAO().findById(Aluno.class, "matricula", matricula);
        if(aluno != null){
            Debito debito = new DebitoDAO().findById(Debito.class, "aluno.matricula", matricula);
            if(debito == null){
                ArrayList<Livro> livros = new ArrayList<Livro>();
                for(Integer aux : codigos){
                    Livro l = new Livro();
                    l.setIdlivro(aux);
                    livros.add(l);
                }
                ArrayList<ItemEmprestimo> itemEmprestimo = new ArrayList<ItemEmprestimo>();
                for(Livro aux : livros){
                    ItemEmprestimo i = new ItemEmprestimo();
                    aux = new LivroDAO().findById(Livro.class, "idlivro", aux.getId());

                    if(aux != null){
                        if(aux.getEmprestavel()){
                            System.out.println(aux.getPrazo());
                            i.setLivro(aux);
                            itemEmprestimo.add(i);
                        }
                    }
                }

                if(itemEmprestimo.size() > 0)
                {
                    System.out.println(itemEmprestimo.size());
                    emprestimo = new Emprestimo();
                    emprestimo.setAluno(aluno);
                    itemEmprestimo = calculaDataDevolucao(itemEmprestimo);
                    emprestimo.setItememprestimos(new HashSet<ItemEmprestimo>(itemEmprestimo));
                    new EmprestimoDAO().insert(emprestimo);
                    for(ItemEmprestimo aux : itemEmprestimo){

                        new ItemEmprestimoDAO().insert(aux);
                    }
                    return true;
                }
            }
        }
        return false;
    }

    public ArrayList<ItemEmprestimo> calculaDataDevolucao( ArrayList<ItemEmprestimo> itemEmprestimo){
        emprestimo.setDataEmprestimo(Calendar.getInstance().getTime());
        emprestimo.setDataPrevista(emprestimo.getDataEmprestimo());;
        for(ItemEmprestimo item : itemEmprestimo){
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(emprestimo.getDataEmprestimo());
            calendar.add(Calendar.DATE, item.getLivro().getPrazo());
            Date aux = calendar.getTime();
            if(emprestimo.getDataPrevista().compareTo(aux) < 0 ) emprestimo.setDataPrevista(aux);
        }

        if(itemEmprestimo.size() > 2){
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(emprestimo.getDataEmprestimo());
            calendar.add(Calendar.DATE,(itemEmprestimo.size()-2)*2);
            emprestimo.setDataPrevista(calendar.getTime());
        }
        for(ItemEmprestimo aux: itemEmprestimo){
            aux.setDataPrevista(emprestimo.getDataPrevista());
            aux.setEmprestimo(emprestimo);
        }
        return itemEmprestimo;

    }
}
```

## GenericDAO.java

```
package util;

import org.hibernate.HibernateException;
import java.util.ArrayList;

public interface GenericDAO<T> {

    public void insert(T o) throws HibernateException;

    public void remove(T o) throws HibernateException;

    public void update(T o) throws HibernateException;

    public T findById(Class<T> c, String column, Object id) throws HibernateException;

    public ArrayList<T> findAll() throws HibernateException;

}
```

## Main.java

```
package util;

import java.util.ArrayList;

import controle.ControleEmprestimo;

public class Main {
    private static ControleEmprestimo control;

    public static void main(String[] args) throws Exception {
        Integer aluno = 120019;
        ArrayList<Integer> codigos = new ArrayList<Integer>();
        codigos.add(120);
        codigos.add(130);
        codigos.add(140);
        Integer numLivros = codigos.size();
        control = new ControleEmprestimo();
        control.emprestar(aluno, codigos, numLivros);
    }
}
```

## AlunoDAO.java

```
package dao;

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import modelo.Aluno;
public class AlunoDAO implements GenericDAO<Aluno> {


    public void insert(Aluno o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove (Aluno o) throws HibernateException{
       // Sets the query for removing 'o'.
    }

    public void update(Aluno o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public Aluno findById(Class<Aluno> o, String coluna, Object id) throws HibernateException{
        return null;
    }
    public ArrayList<Aluno> findAll() throws HibernateException{
        return null;
    }
}
```

## DebitoDAO.java

```
package dao;

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import modelo.Debito;
public class DebitoDAO implements GenericDAO<Debito> {


    public void insert(Debito o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove (Debito o) throws HibernateException{
       // Sets the query for removing 'o'.
    }

    public void update(Debito o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public Debito findById(Class<Debito> o, String coluna, Object id) throws HibernateException{
        return null;
    }
    public ArrayList<Debito> findAll() throws HibernateException{
        return null;
    }
}
```

## EmprestimoDAO.java

```java
package dao;

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import modelo.Emprestimo;
public class EmprestimoDAO implements GenericDAO<Emprestimo> {


    public void insert(Emprestimo o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove (Emprestimo o) throws HibernateException{
       // Sets the query for removing 'o'.
    }

    public void update(Emprestimo o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public Emprestimo findById(Class<Emprestimo> o, String coluna, Object id) throws HibernateException{
        return null;
    }
    public ArrayList<Emprestimo> findAll() throws HibernateException{
        return null;
    }
}
```

## ItemEmprestimoDAO.java

```java
package dao;

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import modelo.ItemEmprestimo;
public class ItemEmprestimoDAO implements GenericDAO<ItemEmprestimo> {


    public void insert(ItemEmprestimo o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove (ItemEmprestimo o) throws HibernateException{
       // Sets the query for removing 'o'.
    }

    public void update(ItemEmprestimo o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public ItemEmprestimo findById(Class<ItemEmprestimo> o, String coluna, Object id) throws HibernateException{
        return null;
    }
    public ArrayList<ItemEmprestimo> findAll() throws HibernateException{
        return null;
    }
}
```

## Livro.java

```java
package dao;

import org.hibernate.HibernateException;
import util.GenericDAO;
import java.util.ArrayList;
import modelo.Livro;
public class LivroDAO implements GenericDAO<Livro> {


    public void insert(Livro o) throws HibernateException{
        // Sets the query for inserting a 'o'.
    }

    public void remove (Livro o) throws HibernateException{
       // Sets the query for removing 'o'.
    }

    public void update(Livro o) throws HibernateException{
        // Sets the query for updating 'o'.
    }

    public Livro findById(Class<Livro> o, String coluna, Object id) throws HibernateException{
        return null;
    }
    public ArrayList<Livro> findAll() throws HibernateException{
        return null;
    }
}
```

# AspectDAO.aj

```
package util;

import org.hibernate.Criteria;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Restrictions;


@SuppressWarnings("deprecation")
public aspect AspectDAO{

    private SessionFactory sessionFactory;

    private SessionFactory hibernateUtil() {
        if (sessionFactory == null) {
            try {
                // Create the SessionFactory from standard (hibernate.cfg.xml)
                // config file.
                sessionFactory = new AnnotationConfiguration().configure()
                        .buildSessionFactory();
                return sessionFactory;
            } catch (Throwable ex) {
                // Log the exception.
                System.err.println("Initial SessionFactory creation failed."
                        + ex);
                throw new ExceptionInInitializerError(ex);
            }
        }else{
            return sessionFactory;
        }

    }

    @SuppressWarnings("rawtypes")
    pointcut insert(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.insert(Object));

    @SuppressWarnings("rawtypes")
    pointcut remove(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.remove(Object));

    @SuppressWarnings("rawtypes")
    pointcut update(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.update(Object));

   @SuppressWarnings("rawtypes")
   pointcut findById(GenericDAO x, Class o, String column, Object i) : target(x) && args(o, column, i) && execution(public Object GenericDAO.findById(Class, String, Object));

   @SuppressWarnings("rawtypes")
   pointcut findAll(GenericDAO x, Object o) : target(x) && args(o) && execution(public void GenericDAO.findAll(Object));


   @SuppressWarnings("rawtypes")
   after(GenericDAO x, Object o) returning() : insert(x, o){
       Session session = hibernateUtil().openSession();
       try{
       session.beginTransaction();
       session.save(o);
       session.getTransaction().commit();
       }catch(HibernateException ex){
           session.getTransaction().rollback();
           ex.printStackTrace();
       }finally{
           session.flush();
           session.close();
       }
   }
   //public void execute()
   @SuppressWarnings("rawtypes")
   after(GenericDAO x, Object o) returning() : remove(x, o){
       Session session = hibernateUtil().openSession();
       try{
       session.beginTransaction();
       session.delete(o);
       session.getTransaction().commit();
       }catch(HibernateException ex){
           session.getTransaction().rollback();
           ex.printStackTrace();
       }finally{
           session.flush();
           session.close();
       }
   }

   @SuppressWarnings("rawtypes")
   after(GenericDAO x, Object o) returning() : update(x, o){
       Session session = hibernateUtil().openSession();
       try{
       session.beginTransaction();
       session.update(o);
       session.getTransaction().commit();
       }catch(HibernateException ex){
           session.getTransaction().rollback();
           ex.printStackTrace();
       }finally{
           session.flush();
           session.close();
       }
   }

   @SuppressWarnings("rawtypes")
   Object around(GenericDAO x, Class o, String y, Object d) : findById(x, o, y, d) {
       Session session = hibernateUtil().openSession();
       Object object = null;
       try {
           Criteria criteria = session.createCriteria(o);
           Criterion name = Restrictions.eq(y, d);
           criteria.add(name);
           session.beginTransaction();
           object = criteria.uniqueResult();
           session.getTransaction().commit();
       } catch (HibernateException ex) {
           session.getTransaction().rollback();
           ex.printStackTrace();
       } finally {
           session.flush();
           session.close();
       }
       return object;
   }

}
```