UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MAIKON ANDRÉ FERREIRA DE MELO

REFATORAÇÃO DA CAMADA DE PERSISTÊNCIA DO FRAMEMK

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

MAIKON ANDRÉ FERREIRA DE MELO

REFATORAÇÃO DA CAMADA DE PERSISTÊNCIA DO FRAMEMK

Trabalho de Conclusão de Curso apresentada como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof.ª Dr.ª Simone Nasser Matos

PONTA GROSSA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

REFATORAÇÃO DA CAMADA DE PERSISTÊNCIA DO FRAMEMK

por

MAIKON ANDRÉ FERREIRA DE MELO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 13 de novembro de 2018 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof.ª Dr.ª Simone Nasser Matos
Orientadora

_____
Prof.ª Dr.ª Simone de Almeida
Membro

_____
Prof.ª Dr.ª Eliana Cláudia Mayumi Ishikawa
Membro

Prof.ª Dr.ª Helyane B. Borges
Responsável pelo Trabalho de Conclusão de Curso

Prof. Dr. André Pinz Borges
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Agradeço a Deus pela sabedoria, saúde e forças para seguir em frente.

Agradeço a minha família, que mesmo distante sempre confiaram no meu sucesso e compreenderam minha ausência ao longo desses anos.

Agradeço a minha namorada Debora que esteve presente nos momentos mais difíceis, sempre incentivando e me apoiando quando tanto precisei.

Agradeço a minha orientadora Simone Nasser Matos, pelos conhecimentos passados ao longo desse trabalho, pela paciência quando as coisas não ocorriam como o esperado, pelas palavras de incentivo que foram essenciais para que eu conseguisse finalizar este trabalho, serei sempre grato.

Agradeço a oportunidade de uma formação superior em uma das melhores universidades do país, por toda a assistência estudantil disponibilizada pela universidade desde o meu ingresso e a cada servidor que trabalhou para que sempre tivesse as melhores condições para desenvolvimento das minhas atividades.

Agradeço aos meus amigos que me acompanharam nessa caminhada e tornaram a tarefa menos árdua.

Dedico esse trabalho ao meu irmão Julian, que pela vontade de Deus teve que nos deixar ainda cedo, porém sua presença segue comigo a cada dia em forma de amor e determinação para seguir em busca dos objetivos.

RESUMO

MELO, M. A. F. **Refatoração da Camada de Persistência do FrameMK.** 2018. 64 f. Trabalho de Conclusão do Curso Superior Tecnologia em Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

O FrameMK é um framework de domínio que tem como objetivo estabelecer o preço de venda de um produto ou serviço. Como o software vem sendo desenvolvido por acadêmicos desta instituição ao longo dos anos, diferentes grupos de desenvolvedores acabam trabalhando no projeto o que pode gerar um sistema com o código de difícil entendimento, dificultando a manutenção e evolução do software. As técnicas de refatoração de software catalogadas por especialistas da área surgem como um meio de amenizar esse tipo de problema, tornando o software mais legível, flexível e de manutenção menos custosa. Para auxiliar no processo de refatoração surgem na literatura os métodos de refatoração, que são constituídos por uma sequência de passos definidos que envolvem em sua maioria, análise de projeto e aplicação de técnicas de refatoração. Na literatura existem métodos de refatoração que tratam de diferentes aspectos, podendo ser voltado para uma determinada linguagem de programação, diferentes domínios ou mesmo os que podem ser utilizados de forma geral. Este trabalho realizou um estudo de três métodos de refatoração: baseado em padrões de projeto para sistemas desenvolvidos em linguagem Java, usado de forma geral e o voltado para frameworks de domínio, que é baseado nos dois métodos citados anteriormente e foi utilizado neste trabalho. O método fundamentado em framework de domínio foi adaptado para melhor atender as necessidades encontradas durante a análise da camada de persistência do framework, visando aproveitar o processo já definido. A adaptação do método se mostrou eficiente para os objetivos deste trabalho, tendo em vista que o mesmo auxiliou na detecção de códigos com sintomas de má concepção ou implementação de escolhas, os *bad smells* em todas as classes da camada trabalhada. A aplicação da refatoração gerou dados quantitativos que possibilitaram analisar o impacto causado pelas técnicas aplicadas em cada uma das classes da camada de persistência do framework. Ao final do processo de refatoração foram aplicadas seis técnicas diferentes, separadas em 5 categorias distintas. As classes refatoradas apresentaram mudanças referentes a quantidade de métodos e linhas de códigos, porém, os resultados relevantes foram as melhorias implementadas na camada trabalhada, reduzindo a complexidade e aumentaram a manutenibilidade do código.

Palavras-chave: Refatoração. FrameMK. Método de Refatoração. Framework de Domínio.

ABSTRACT

MELO, M. A. F. **Refactoring of Persistence Layer of FrameMK**. 2018. 64 p. Completion Work for Higher Education Technology in Systems Analysis and Development – Federal Technological University of Paraná. Ponta Grossa, 2018.

FrameMK is a domain framework that aims to establish the selling price of a product or service. As software has been developed by scholars of this institution over the years, different groups of developers end up working on the project, which can generate a system with difficult code understanding, making it difficult to maintain and evolve the software. Software refactoring techniques cataloged by area experts come as a means of mitigating this type of problem, making software more readable, flexible, and maintenance less costly. In order to aid in the refactoring process, refactoring methods appear in the literature, which consist of a sequence of defined steps that mostly involve design analysis and the application of refactoring techniques. In the literature there are methods of refactoring that deal with different aspects, and may be aimed at a particular programming language, different domains or even those that can be used in general. This work has carried out a study of three methods of refactoring: based on design patterns for systems developed in Java language, generally used and the one oriented towards domain frameworks, which is based on the two methods mentioned previously and was used in this work. The method based on the domain framework was adapted to better meet the needs encountered during the analysis of the persistence layer of the framework, in order to take advantage of the already defined process. The adaptation of the method was efficient for the purposes of this work, since it helped in the detection of codes with symptoms of bad design or implementation of bad smells in all classes of the layer worked. The application of the refactoring generated quantitative data that allowed to analyze the impact caused by the applied techniques in each of the classes of the layer of persistence of the framework. At the end of the refactoring process, six different techniques were applied, separated into five different categories. The refactored classes presented changes related to the number of methods and lines of code, but the relevant results were the improvements implemented in the layer, reducing the complexity and increasing the maintainability of the code.

Keywords: Refactoring. FrameMK. Refactoring Method. Domain Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS E ACRÔNIMOS

ABC	*Activity Based Costing* (Custeio Baseado em Atividades)
ABES	Associação Brasileira das Empresas de Software
BD	Banco de Dados
FAQ	*Frequently Asked Questions* (Perguntas Frequentes)
FPV	Formação de Preço de Venda
FrameMK	Framework de Formação de Preço de Venda
GPSI	Grupo de Pesquisa em Sistemas de Informação
GUI	*Graphical User Interface* (Interface Gráfica do Usuário)
Java	Linguagem de Programação Orientada a Objetos
JDK	*Java SE Development Kit*
LPES	Linha de Pesquisa de Engenharia de Software
LPS	Linhas de Produtos de Software
ROIC	*Return on Invested Capital* (Retorno Sobre o Capital Investido)
SEBRAE	Serviço Brasileiro de Apoio às Micro e Pequenas Empresas
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

O software sofre, inevitavelmente, modificações após ser entregue ao cliente final (PRESSMAN, 2005). Além disso, sua manutenção têm sido uma das mais difíceis e custosas fases do ciclo de vida, especialmente em grandes sistemas (BOURGE; BROWN, 2000).

A refatoração de software surge como uma técnica de manutenção que auxilia programadores na codificação de maneira eficiente, economizando recursos e tempo, e foi apresentada originalmente por Martin Fowler (FOWLER, 1999).

Refatoração é o processo de mudar a estrutura interna de um sistema de software de forma a não alterar o comportamento externo do código, melhorando sua estrutura interna e tornando o sistema fácil de ser entendido e de manutenção menos dispendiosa. É um caminho disciplinado para limpar o código e minimizar as chances de introdução de erros (FOWLER, 1999).

O processo de refatoração é composto por uma sequência de passos, podendo haver movimentação de um atributo de uma classe para outra, remoção de parte do código de um método para dar origem a um novo método, inserção de algum código acima ou abaixo da hierarquia, etc. O efeito cumulativo dessas pequenas mudanças é que pode melhorar o projeto (FOWLER, 1999).

As técnicas de refatoração podem ser encontradas em catálogos propostos por especialistas da engenharia de software, como em Fowler (1999) que apresenta 91 técnicas de refatoração classificadas em 17 categorias e em Kerievsky (2004) que contem 24 técnicas com ênfase em padrões de projeto.

Para melhores resultados de aplicação da refatoração pode-se utilizar métodos que auxiliam no processo, facilitando na identificação de partes do código que podem ser refatorados e indicando as técnicas apropriadas para cada problema.

Os métodos de refatoração encontrados na literatura são definidos como genéricos ou específicos. Os métodos genéricos são voltados a diferentes sistemas, linguagens e estruturas. Os métodos específicos são voltados para um determinado sistema, assim como pode atender a um tipo específico de linguagem de programação.

O desenvolvimento deste trabalho tem como base alguns métodos de refatoração presentes na literatura: o método de Rapeli (2006), Mens e Tourwé (2004) e Barros (2015).

Rapeli (2006) apresentou em sua dissertação um método de refatoração específico, focado em sistemas construídos na linguagem Java utilizando padrões de projeto. O método proposto serve como auxílio para quem precisa refatorar sistemas com estas características e, sendo composto de 3 etapas: entender, refatorar utilizando padrões de projeto e verificar o sistema após a refatoração.

Mens e Tourwé (2004) propõem um método de refatoração genérico, que não atende um tipo de sistema específico ou uma determinada linguagem de programação. O método é composto por 6 passos: identificar onde o software deve ser refatorado, determinar quais refatorações devem ser aplicadas nos lugares identificados, garantir que a refatoração aplicada preserve o comportamento do software, aplicar a refatoração, avaliar os efeitos da refatoração nas características do software e manter a coerência entre o código refatorado e os outros artefatos do software.

Barros (2015) propõe um método de refatoração voltado para frameworks de domínio, e têm como base os dois métodos citados anteriormente, além de possuir etapas que tratam as características como: metapadrões, inversão de controle e padrões de projeto fundamentais na arquitetura dos frameworks. O método é dividido em 3 etapas: entender o sistema, ordenar os módulos e refatorar o módulo. Cada uma dessas etapas é composta por um conjunto de passos que devem ser seguidos.

Este trabalho buscou por métodos de refatoração na literatura a fim de analisar, selecionar e aplicar o que melhor para refatorar a camada de persistência do FrameMK. A análise dos métodos encontrados considerou se o mesmo é voltado para um determinado tipo de framework, se utiliza padrões ou atende uma linguagem de programação, buscando utilizar como base um método que se adeque ao framework utilizado como estudo de caso. Um framework é uma estrutura que tem como objetivo prover uma funcionalidade genérica que serve de apoio para a construção de outra aplicação (FAYAD, 1999). O framework usado como estudo de caso é o Framework de Formação de Preço de Venda (FrameMK) que está em desenvolvimento por acadêmicos da Universidade Tecnológica Federal do Paraná (UTFPR), campus Ponta Grossa. O principal objetivo deste framework é possibilitar ao usuário calcular o preço de venda de um determinado produto ou serviço usando vários métodos de precificação.

1.2 OBJETIVOS

O presente trabalho tem como objetivo geral aplicar um método de refatoração para melhorar a estrutura interna do framework de formação de preço de venda na camada de persistência, buscando acrescentar tratamentos que aumentem a sua qualidade, flexibilidade e reusabilidade.

Os objetivos específicos do trabalho são:

- Identificar as características de métodos de refatoração de software publicados na literatura e ao final escolher o que melhor se ajuste as características do FrameMK.
- Extrair os catálogos de técnicas apresentadas em Fowler (1999) e Kerievsky (2002) e identificar seu processo de funcionamento.
- Analisar os resultados obtidos após a aplicação das técnicas na camada de persistência do FrameMK.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho encontra-se organizado em cinco capítulos. O capítulo 2 apresenta uma visão inicial sobre a refatoração, da difícil tarefa de realizar manutenção a um software, das vantagens proporcionadas pela refatoração neste aspecto, além de citar algumas das técnicas de refatoração conhecidas e frequentemente utilizadas. Ainda neste capítulo são abordados os três métodos de refatoração usados como referência para este trabalho: O Método Baseado em Padrões de Projeto de Rapeli (2006), o Método de Mens e Tourwé (2004) e o método de Barros (2015).

O capítulo 3 apresenta o Framework de Formação de Preço de Venda, o FrameMK, que é o framework utilizado como estudo de caso deste trabalho. O capítulo 4 apresenta os procedimentos executados e o passo a passo da refatoração da camada de persistência do FrameMK. No capítulo 5 é realizada a conclusão do trabalho, avaliando os resultados obtidos com a refatoração do framework.

2 REFATORAÇÃO DE SOFTWARE

Este capítulo apresenta uma visão geral sobre refatoração de software. A seção 2.1 descreve algumas vantagens de se aplicar a refatoração e comenta sobre as técnicas de refatoração publicadas e catalogadas por especialistas da área. A seção 2.2 narra a refatoração de software baseada em padrões de projeto. A seção 2.3 apresenta métodos da literatura que ajudam no processo de refatoração de software. Por fim, a última seção relata as considerações finais deste capítulo.

2.1 TÉCNICAS DE REFATORAÇÃO

A manutenção de software tem sido uma das mais difíceis e custosas fases do ciclo de vida do software, especialmente em grandes sistemas (BURGE; BROWN, 2000). A refatoração de software foi apresentada originalmente por Fowler (1999) que a considera como sendo uma técnica de manutenção que auxilia programadores na codificação de maneira eficiente, economizando recursos e tempo.

Visando tornar a aplicação das técnicas mais eficiente, especialistas publicam catálogos que atendem tanto software de forma geral quanto os voltados a algumas especificidades.

A aplicação das técnicas gera melhorias ao projeto como exemplo, codificação legível, diminuição de códigos duplicados e falhas são encontradas com mais facilidade. Ao término da aplicação de técnicas é notável um software programado disciplinadamente e de manutenção menos dispendiosa.

Visando fazer com que as técnicas de refatoração de software sejam adotadas definitivamente como prática no desenvolvimento de software, especialistas da área publicam na literatura catálogos que sugerem técnicas de refatoração que podem envolver encapsulamento, associações, chamada de métodos, composição de métodos, entre outras.

Fowler (1999) apresenta 91 técnicas de refatoração classificadas em 17 categorias (conforme apresentado no ANEXO A) e Kerievsky (2004) por sua vez apresenta 24 técnicas com ênfase em padrões de projeto. O Quadro 1 apresenta

uma categoria denominada de Herança (*Inheritance*) que é utilizada como exemplo na explicação da técnica *Extract Superclass* (Extrair Superclasse).

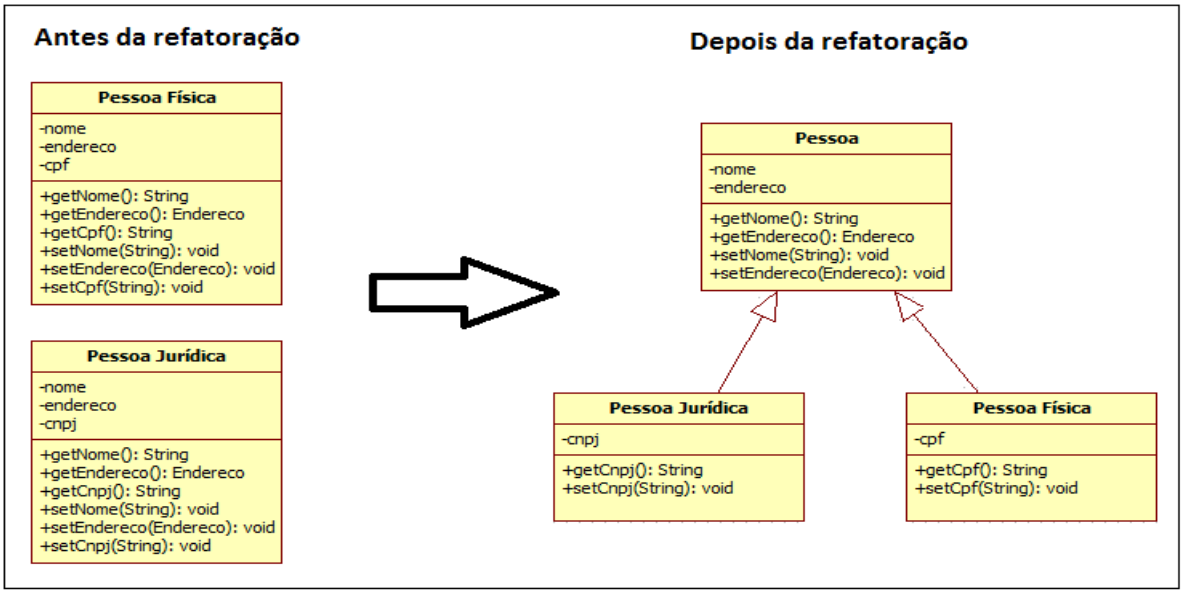
Quadro 1 - Categoria e técnicas de refatoração definidas por Fowler

Inheritance	*Collapse Hierarchy* *Encapsulate Downcast* *Extract Interface* *Extract Module* *Extract Subclass* *Extract Superclass* *Form Template Method* *Introduce Null Object* *Pull Up Constructor Body* *Pull Up Field* *Pull Up Method* *Pull Down Field* *Pull Down Method* *Replace Abstract Superclass with Module* *Replace Conditional with Polymorphism* *Replace Delegation with Hierarchy* *Replace Delegation with Inheritance*

Fonte: Adaptado de Fowler (1999)

A categoria *Inheritance* é composta por 17 (dezessete) técnicas. Entre as técnicas dessa categoria um exemplo é a *Extract Superclass* (Extrair Superclasse) que sugere a extração das características em comum entre as classes para uma nova classe (uma classe-pai). Essa classe-pai servirá como uma classe geral, possibilitando a herança de seus métodos e atributos por classes que atribuirão suas especificidades. A Figura 1 apresenta o resultado da aplicação da técnica de *Extract Superclass.*

Figura 1 - Exemplo de aplicação da técnica Extract Superclass



Fonte: Autoria própria

A Figura 1 apresenta um exemplo da aplicação da técnica de *Extract superclass* em um modelo que permite reduzir a redundância de informações, o que torna o modelo mais legível, de manutenção menos dispendiosa, além de torná-lo apto a receber funcionalidades e informações em manutenções futuras.

2.2 REFATORAÇÃO BASEADA EM PADRÕES DE PROJETO

A refatoração utilizando padrões de projeto é uma abordagem promissora para a melhoria do projeto durante as atividades de desenvolvimento (MURAKI; SAEKI, 2001).

Os padrões de projetos são divididos em três categorias: criacionais, estruturais e comportamentais que podem ser encontrados no livro *Design Patterns: Elements of Reusable Object-Oriented Software* da *Gang of Four* que lista os vinte e três padrões (GAMMA *et al*.,1995).

Um dos benefícios da utilização dos padrões é garantir que o software será desenvolvido utilizando as técnicas de refatoração que mantém o código menos sujeito a falhas e *bad smells* que se referem à códigos com sintomas de má concepção ou implementação de escolhas (FOWLER, 1999). O Quadro 2 mostra a relação entre alguns padrões de projetos com algumas técnicas de refatoração.

Quadro 2 - Exemplos da relação entre padrões e a refatoração

Cheiro (*Smell*)	**Refatorações**
Conditional Complexity	*Move Embellishment to Decorator (73)* *Replace Conditional Calculations with Strategy (50)* *Replace State-Altering Conditionals with (154)*
Combinatorial Explosion	*Replace Conditional Searches with specification (99)*
Duplicated Code	*Chain Constructors (19)* *Introduce Polymorphic Creation with Factory Method (43)*
Inapropriate Intimacy	*Encapsulate Composite with Builder (64)*
Indecent Exposure	*Encapsulate Classes with Creation Methods (27)*
Large Class	*Extract Creation Class (34)*
Long Method	*Compose Method (119)* *Move Accumulation Collecting Parameter (94)*
Long Parameter List	*Replace Conditional Calculations with Strategy (50)*

Fonte: Adaptado de Kerievsky (2002)

Por meio dos exemplos do Quadro 2 se pode notar a forte relação entre os padrões de projeto e a refatoração, para cada *smell* é sugerida uma técnica de refatoração que por sua vez pode estar associada a um padrão de projeto.

2.3 MÉTODOS DE REFATORAÇÃO

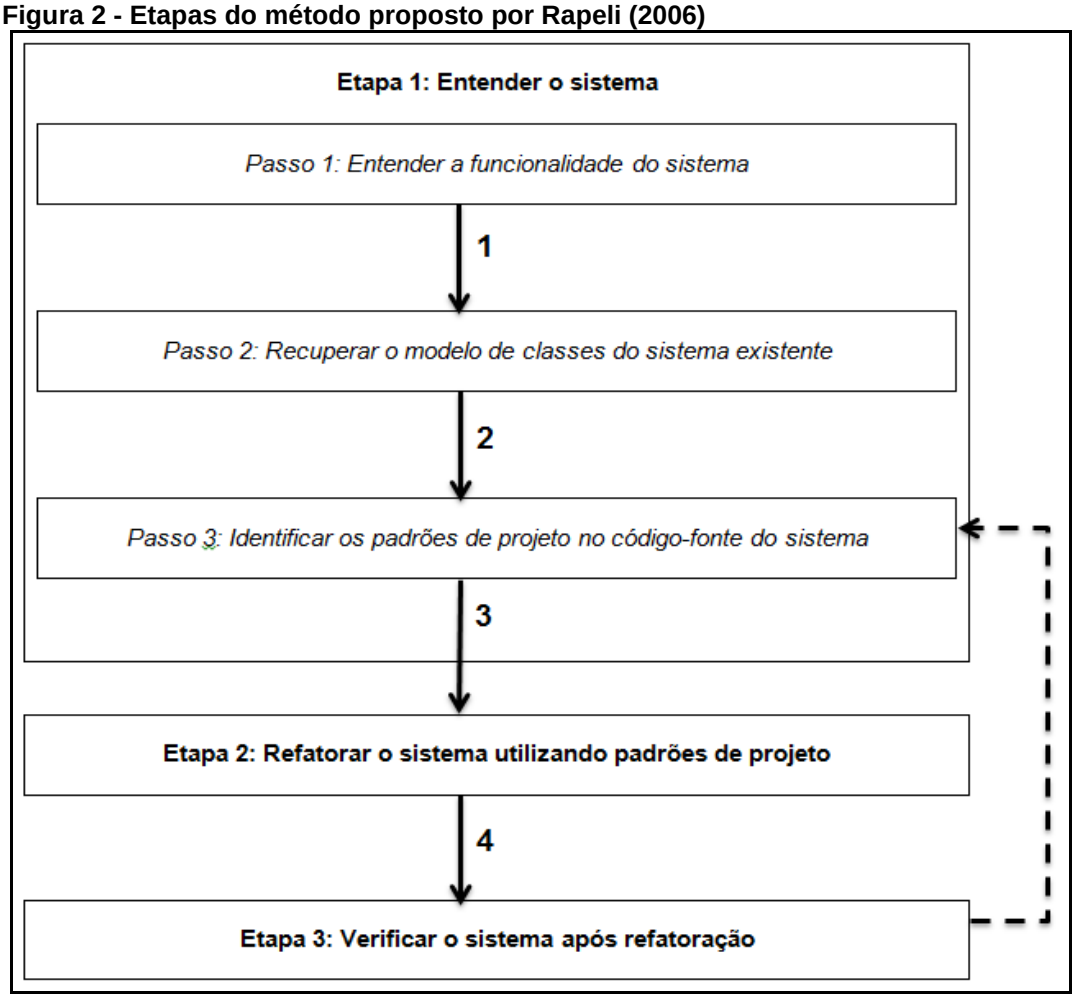
Esta seção apresenta alguns métodos de refatoração presentes na literatura e que serão utilizados como fonte de estudo para a realização deste trabalho. A seção 2.3.1 apresenta o método de refatoração baseado em padrões de projeto de Rapeli (2006). A seção 2.3.2 descreve o método proposto por Mens e Tourwé (2004). A seção 2.3.3 relata o método voltado para frameworks de domínio de Barros (2015) que será o método utilizado neste trabalho para refatoração da camada de persistência do FrameMK.

2.3.1 Método Baseado em Padrões de Projeto de Rapeli

A dissertação de Rapeli (2006) foi elaborada com o foco voltado para sistemas construídos em linguagem Java utilizando padrões de projeto. O trabalho foca na refatoração de sistemas orientados a objetos, tendo sido elaborado com base em estudos de caso de sistemas de informação. O trabalho serve como auxílio para quem optar por refatorar sistemas com essas características.

O método de refatoração proposto por Rapeli (2006) propõe utilizar os benefícios dos padrões de projetos para reorganizar e melhorar a estrutura do código-fonte existente além de, obter e atualizar a documentação do projeto. O método possui três etapas bem definidas que conduzem de forma organizada, a identificação de padrões que possam ser aplicados visando melhorar a estrutura do projeto. A Figura 2 apresenta as etapas e passos que compõem a aplicação do método.

Figura 2 - Etapas do método proposto por Rapeli (2006)



Fonte: Rapeli (2006, p. 26)

A primeira etapa proposta *Entender o Sistema* tem como objetivo tornar o sistema familiar ao engenheiro de software, caso ele ainda não tenha conhecimento de como o mesmo encontra-se codificado e estruturado. Como apresentado na Figura 3 esta etapa é composta por três passos: no primeiro passo (*Entender a funcionalidade do sistema)* o escopo e a arquitetura do sistema são identificados, no segundo passo (*Recuperar o modelo de classes do sistema existente)* o modelo de classes do sistema é recuperado e no terceiro passo *(Identificar os padrões de projeto no código-fonte do sistema)* são utilizadas as informações obtidas nos passos anteriores para identificar os padrões de projetos que possam ser utilizados no processo de refatoração do sistema.

Para melhor entendimento desta etapa, os passos são apresentados no seguinte formato: identificação do passo, representação da ação que será executada

no passo, descrição da justificativa da ação executada e objetivo esperado e a descrição do que deve ser feito para que o objetivo seja alcançado.

O primeiro passo *Entender a funcionalidade do Sistema* busca entender o funcionamento do sistema, para isso algumas informações do sistema são necessárias. Rapeli (2006) organiza as informações coletadas em uma tabela que segue o *layout* apresentado no Quadro 3.

Quadro 3 - Layout dos dados a serem obtidos ao executar o sistema

Número da interação	Interação do usuário com o sistema (Entrada)	Resposta do sistema (Saída)		
		Na tela do usuário	No console do programador	Arquivos gerados

Fonte: Rapeli (2006, p.41)

Os dados que devem ser coletados são:

- Número da interação: são números sequenciais;
- Interação do usuário com o sistema (Entrada): são obtidas todas as interações realizadas pelo usuário no momento em que o sistema é executado;
- Resposta do sistema (saída): após a interação do usuário uma lista contendo as saídas geradas é fornecida.

O objetivo do segundo passo é *Recuperar o modelo de classes do sistema existente,* sendo que, o engenheiro de software pode encontrar duas situações que variam de acordo com cada sistema:

- Modelo de classes já existe: nesse caso o modelo deverá ser avaliado para verificar se as informações nele contidas estão em acordo com o código-fonte do sistema;
- Modelo de classes não existe: neste caso o modelo de classes deverá ser criado a partir do código-fonte do sistema, fica a critério do engenheiro optar por construir o modelo manualmente ou utilizar uma ferramenta que recupere o modelo automaticamente.

O objetivo do terceiro passo é *Identificar os padrões de projeto no código-fonte do sistema*. Os dois passos executados anteriormente facilitam a identificação dos padrões aplicáveis ao sistema, pois o engenheiro terá conhecimento das

funcionalidades do sistema além de já conhecer a sua estrutura após ter recuperado o modelo de classes do sistema existente.

Assim, da mesma forma que se utiliza os *bad smells* como indícios de códigos que podem ser refatorados, Rapeli (2006) representa os indícios de que um padrão pode ser utilizado por meio de um quadro que contém o nome de cada um dos padrões, a categoria que pertence e uma breve descrição das situações em que o padrão pode ser empregado, além de disponibilizar trechos de código que exemplificam esses indícios. O *layout* do quadro proposto por Rapeli (2006) é apresentado de forma resumida no Quadro 4.

Quadro 4 - Exemplo de indícios da aplicabilidade dos padrões no sistema

<table>
<tr><th>Padrão</th><th>Indícios</th></tr>
<tr><td colspan="2">Categoria: Criação</td></tr>
<tr><td>Abstract Factory</td><td rowspan="2">O código deve apresentar um ponto em que há a possibilidade de instanciação de uma ou mais classes. As possíveis classes a serem instanciadas devem ser subclasses da mesma superclasse, implementar a mesma interface ou serem dependentes umas das outras. Exemplo:

<pre>if(condicao1){

    ClasseA classeA = new ClasseA();
    ClasseB classeB = new ClasseB();

}else if(condicao2){
    ClasseC classeC = new ClasseC();
}
ClasseD classeD = new ClasseD();
ClasseE classeE = new ClasseE();</pre>
Utilizar o padrão Abstract Factory quando for necessária a delegação para a instanciação de objeto(s) e o padrão Factory Method quando for necessária a herança para a instanciação de objeto(s).</td></tr>
<tr><td>Factory Method</td></tr>
</table>

Fonte: Rapeli (2006, p.42)

A segunda etapa Refatorar o sistema utilizando padrões de projeto consiste em refatorar o sistema utilizando os padrões identificados na primeira etapa. A execução da etapa é dividida em duas atividades que são executadas paralelamente: o modelo de classes é atualizado com os padrões identificados e é realizado a implementação do software utilizando estes padrões.

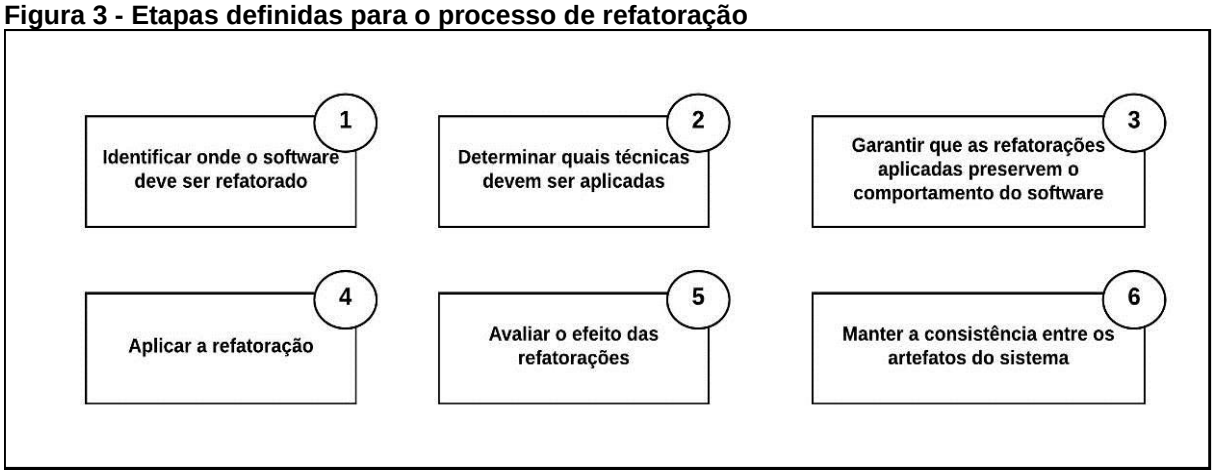
Na terceira etapa o objetivo é Verificar o sistema após a refatoração, analisando se as funcionalidades do sistema continuam inalteradas o que garante que o sistema não irá apresentar alterações inesperadas no seu comportamento externo. A execução desta etapa é composta por um único passo, podendo ser utilizado as mesmas interações realizadas no primeiro passo da etapa 1 (entender a funcionalidade do sistema) para analisar se as saídas do sistema continuam inalteradas.

2.3.2 MÉTODO DE MENS E TOURWÉ

Mens e Tourwé (2004) buscaram na literatura um vasto referencial teórico que, tornou possível comparar e discutir abordagens distintas relacionadas à refatoração de software. Durante a análise das abordagens foram definidos alguns critérios essenciais para se analisar: as atividades de refatoração suportadas, as técnicas específicas e a forma com que são usadas para suportar tais atividades, os tipos de artefatos de software que estão sendo refatorados, a importância da análise das consequências ao se utilizar uma determinada ferramenta de refatoração e o efeito causado pela refatoração no processo de software.

Com base no estudo das diferentes abordagens encontradas referente ao tema, foi elaborado um processo de refatoração composto por seis etapas: Identificar onde o software deve ser refatorado, determinar quais técnicas devem ser aplicadas, garantir que as refatorações aplicadas preservem o comportamento do software, aplicar a refatoração, avaliar o efeito das refatorações e manter a consciência entre os artefatos do sistema. A Figura 3 apresenta uma adaptação das etapas definidas pelos autores do método.

Figura 3 - Etapas definidas para o processo de refatoração



Fonte: Adaptado de Mens e Tourwé (2004)

A primeira etapa do processo consiste em identificar onde as refatorações devem ser aplicadas, neste momento é determinado o nível de abstração apropriado para a aplicação. O nível de abstração determinado definirá se as refatorações serão aplicadas diretamente no código-fonte do programa ou em artefatos mais abstratos do software, tais como: modelos de projetos ou documentos de requisitos.

Na segunda etapa são determinadas quais técnicas de refatoração devem ser aplicadas nos locais identificados na etapa anterior. Geralmente estas duas etapas são combinadas com o auxílio de ferramentas que detectam os *bad smells* e sugerem as técnicas apropriadas para solucionar o problema.

A terceira etapa do processo permite preservar o comportamento do software. Para garantir isto, Mens e Tourwé (2004) implementam um disciplinado conjunto de testes baseado na definição proposta por Opdyke (1992) que afirma que, para o mesmo conjunto de entrada de valores, o conjunto resultante na saída deve ser o mesmo antes e depois da refatoração.

A quarta etapa foca na aplicação das técnicas encontradas nos locais adequados.

Na quinta etapa do processo são avaliados os efeitos causados pela refatoração. Este procedimento é realizado por meio das avaliações de atributos qualitativos do software, como robustez, extensibilidade, reusabilidade e desempenho. Para que seja possível analisar estes atributos é necessário que se avalie cada refatoração aplicada de acordo com o propósito particular. Deve-se considerar que cada refatoração pode apresentar distintas melhorias na estrutura do software como tamanho, complexidade, acoplamento, coesão, etc. Por outro lado,

deve-se considerar o desempenho do software como uma característica importante que pode ser afetada pela refatoração.

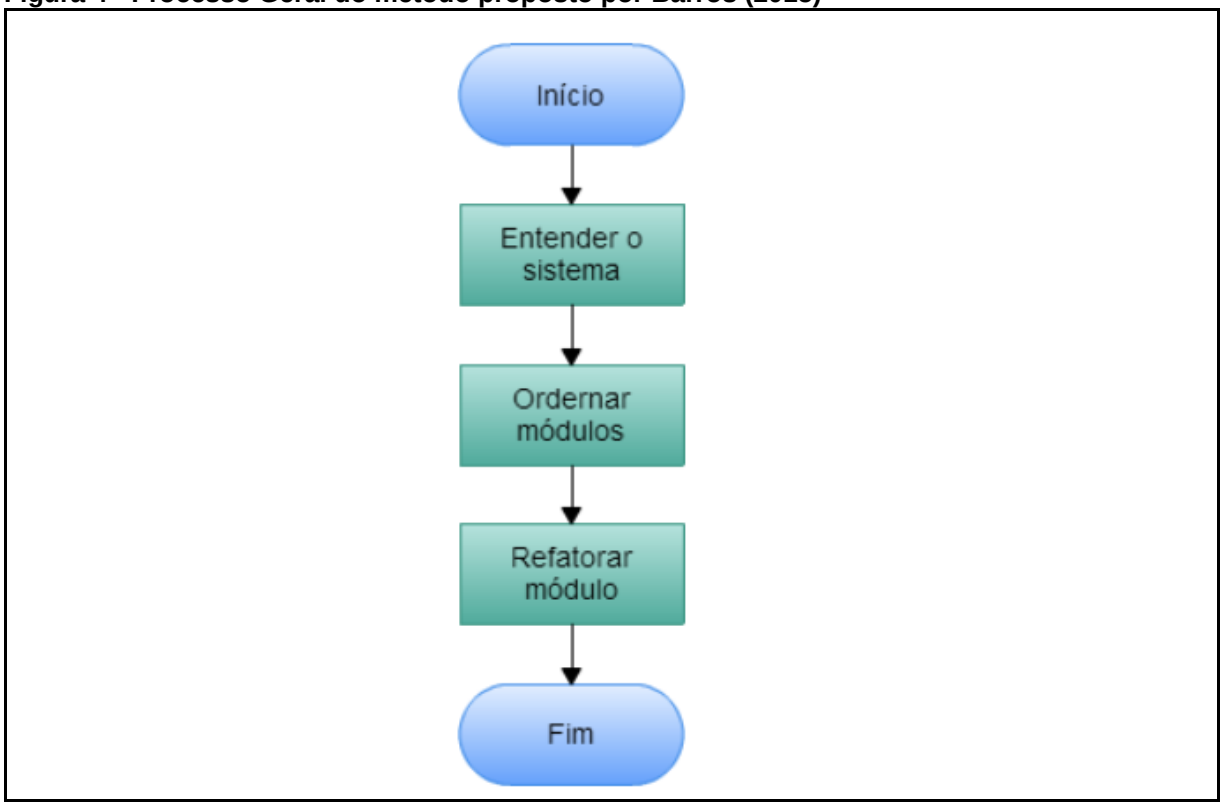
A sexta e última etapa do processo visa manter a consistência entre todos os artefatos do software, tais como o código-fonte, documentação, modelos, especificação de requisitos, testes, etc.

2.3.3 MÉTODO DE BARROS

Barros (2015) apresenta um método de refatoração com o foco voltado para frameworks de domínio que utiliza como base os métodos citados na seção 2.3.1 e 2.3.2. O método proposto é dividido em três etapas: Entender o sistema, Ordenar módulos e Refatorar módulo. A Figura 4 apresenta o fluxograma geral do método.

O diferencial do método proposto por Barros (2015) é o seu foco em frameworks de domínio, além de tratar características como metapadrões, inversão de controle, padrões de projeto que são fundamentais na arquitetura dos frameworks e a automatização da refatoração utilizando ferramentas que auxiliam a análise do código.

Figura 4 - Processo Geral do método proposto por Barros (2015)



Fonte: Barros (2015, p.33)

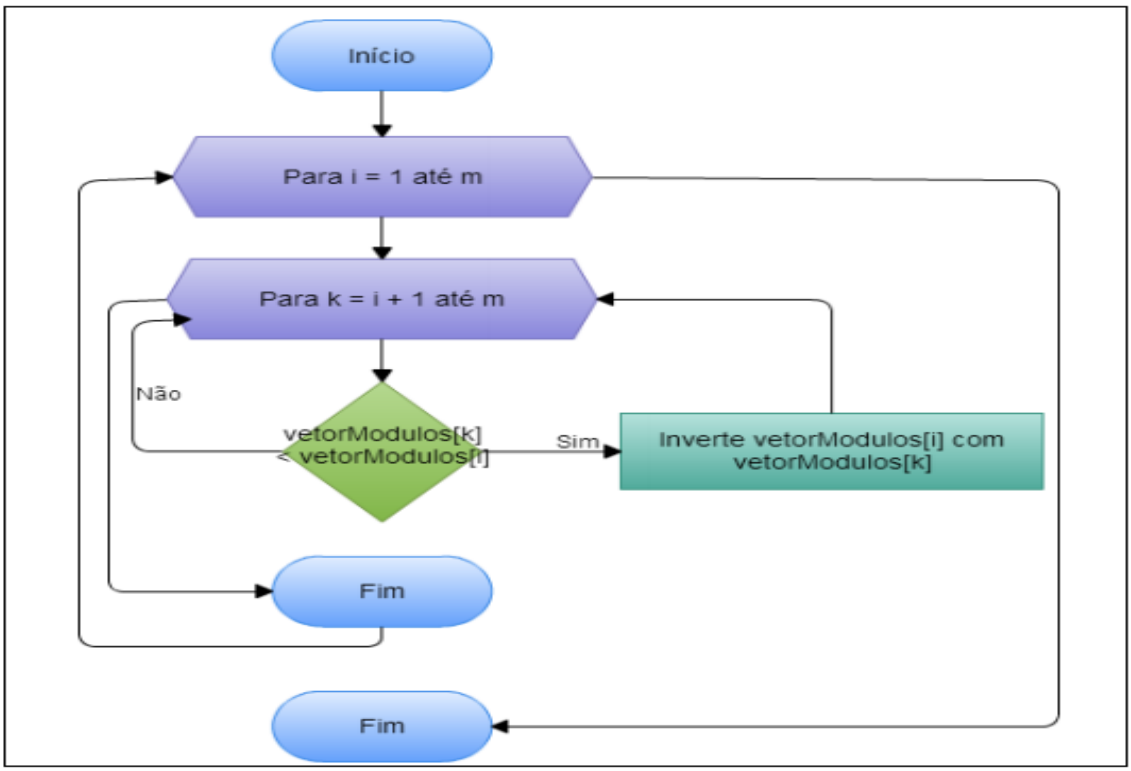
Este método permite ao engenheiro de software Entender o Sistema e o objetivo da primeira etapa, assim como no método proposto por Rapeli (2006) a primeira etapa também é composta por três passos: Entender o sistema, Ordenar módulos e Refatorar módulo.

O primeiro passo Entender o sistema é composto de três novos passos:

- Utilizar o sistema: onde o engenheiro tem o primeiro contato com sistema que será refatorado;
- Verificar módulos: onde serão identificados os módulos do sistema;
- Gerar diagrama de classe dos módulos: será gerado o modelo de classes dos módulos identificados no sistema, permitindo visualizar o funcionamento e o relacionamento entre as classes do sistema.

A segunda etapa definida no processo de refatoração é *Ordenar Módulo*, onde a escolha pelo módulo que será refatorado é baseada no algoritmo de ordenação *Insertion Sort* com objetivo de ordenar os módulos em ordem crescente pela quantidade de *bad smells*. O fluxograma da Figura 5 exibe a forma como o módulo que será refatorado é escolhido.

Figura 5 - Escolha do módulo que será refatorado



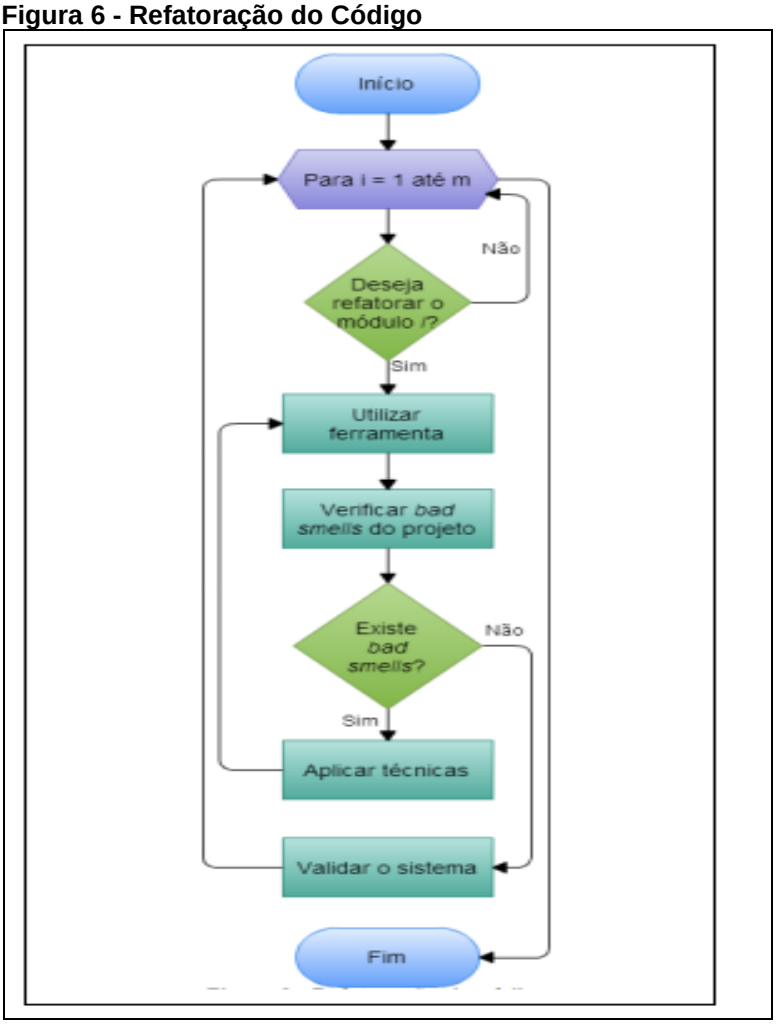
Fonte: Barros (2015, p.36)

A análise dos *bad smells* de cada módulo é feito com o auxílio de ferramentas de análise, tais como: *SonarQube (2008), FindBugs (2003), CheckStyle (2001)* e *CodePro AnalytiX (2001).*

Ao finalizar a execução das interações presentes no fluxograma da Figura 5, todos os módulos do sistema estarão avaliados gerando um vetor com os módulos ordenados do menor para o maior, levando em consideração a quantidade de *bad smells* encontrados em cada módulo. A opção de iniciar pelos módulos que apresentam menor quantidade de *bad smells* tende a tornar a aplicação da refatoração mais fácil para desenvolvedores menos experientes.

A última etapa do método proposto é Refatorar Módulo onde por fim a refatoração do código-fonte é realizada. Barros (2015) apresenta a refatoração do módulo por meio do fluxograma da Figura 6.

Figura 6 - Refatoração do Código



Fonte: Barros (2015)

O fluxograma da Figura 6 apresenta o processo que envolve a refatoração do sistema, no qual é realizado a interação entre os módulos já ordenados permitindo o desenvolvedor optar refatorar o módulo ou seguir para o próximo. Quando o desenvolvedor optar por refatorar o módulo será feito a utilização da ferramenta de análise, focando no módulo em questão.

Após concluir a análise do módulo do projeto, são identificados os *bad smells,* caso eles existam, o passo seguinte será voltado para aplicação das técnicas de refatoração focadas em frameworks. A escolha das técnicas fica a critério do desenvolvedor que deverá selecioná-las conforme analisa o código do projeto.

O método de Barros (2015) apresenta ainda a refatoração com os seguintes metapadrões: *Unification, 1:1 Connection, 1:N Connection, 1:1 Recursive Unification, 1:N Recursive Unification, 1:1 Recursive Connection, 1:N Recursive Connection* além da refatoração baseada em *Inversão de controle,* um de seus diferenciais em relação aos citados anteriormente.

2.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou a importância do processo de refatoração que permite criar programas mais flexíveis, de fácil manutenção e reusáveis. O processo de refatoração pode ser realizado por técnicas como as propostas por Fowler (1999) ou baseada em padrões de projeto (GAMMA *et al*, 1995).

Alguns métodos foram propostos para auxiliar no processo de refatoração como o de Mens e Tourwé (2004), Rapeli (2006) e Barros (2015). O método de Barros (2015) é voltado para refatoração de frameworks e abrange características dos dois outros autores e por isto será utilizada neste trabalho.

3 FRAMEWORK DE FORMAÇÃO DO PREÇO DE VENDA

Este capítulo tem como objetivo apontar alguns conceitos essenciais relacionados à utilização de frameworks, bem como, apresentar o framework estudado neste trabalho. A seção 3.1 apresenta uma introdução conceitual aos desafios encontrados pelos desenvolvedores quando precisam implementar seus softwares com qualidade, bem como os conceitos sobre frameworks. A seção 3.2 descreve o framework utilizado como caso de estudo deste trabalho, o FrameMK e apresenta sua estrutura, dando ênfase a camada de persistência, na qual o método de refatoração foi aplicado. Por fim, na seção 3.3 são feitas as considerações finais do capítulo.

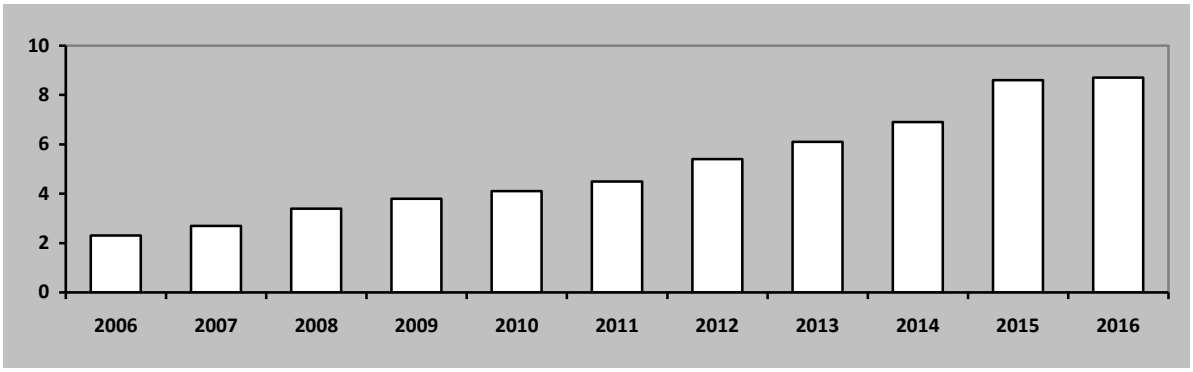
3.1 FRAMEWORKS: UMA VISÃO GERAL

De acordo com os dados apresentados pela Associação Brasileira das Empresas de Software (ABES), ao longo dos últimos anos o desenvolvimento de softwares tem apresentado um crescimento exponencial dentro do mercado brasileiro, isso acontece muito em função da constante evolução tecnológica que atinge as empresas dos mais variados seguimentos.

Esta nova realidade vivenciada dentro das empresas acaba tornando-as cada vez mais informatizadas, para manter um determinado empreendimento competitivo, administradores optam frequentemente pela contratação de softwares gestores para tornar o gerenciamento interno de suas empresas menos dispendioso, além de softwares exclusivamente focados em prover serviços aos seus clientes. Estas necessidades são os principais fatores que impulsionaram o mercado voltado para o desenvolvimento de software.

O Gráfico 1 ilustra o crescimento do mercado de software entre os anos de 2006 a 2016, as informações apresentadas foram coletadas pela ABES.

Gráfico 1 - Evolução dos indicadores do mercado brasileiro de software (US$ BILHÕES)



Fonte: ABES 2017

Os dados apresentados pela ABES evidenciam um crescimento considerável em relação ao investimento em softwares por parte das empresas, por outro lado, com o crescimento dos investimentos consequentemente se aumentam as exigências em relação aos softwares desenvolvidos.

Para que tais softwares sejam aceitos no mercado precisam atender a um conjunto de requisitos de qualidade. Garantir a qualidade de um software é custoso e, com o incessante surgimento de novas tecnologias acaba por se tornar uma tarefa ainda mais difícil.

Os frameworks surgem como uma ferramenta composta por definições e abordagens de autores conceituados na área da engenharia de software. A subseção 3.1.1 apresenta algumas definições de framework encontradas na literatura. Na subseção 3.1.2 relata as diferentes dimensões em que estes frameworks podem ser classificados. Por fim, na subseção 3.1.3 analisa-se as vantagens e desvantagens encontradas pelos desenvolvedores ao optar pelo uso de um framework como apoio no desenvolvimento de softwares.

3.1.1 Definições

Desde o surgimento das primeiras abordagens em relação aos frameworks, muitas definições foram propostas para o tema e, ao longo dos anos essas definições foram amadurecendo possibilitando que se tenha acesso a uma gama de pesquisas relacionadas ao assunto.

Existem definições clássicas de frameworks, como a proposta por Johnson and Foote (1988) em que um framework é definido como sendo um conjunto de

classes que definem um projeto abstrato de soluções para uma família de problemas relacionados. Mais tarde, essa definição foi complementada por Gamma *et al*. (1994), no qual o conceito de framework é estendido como sendo um conjunto de classes tanto abstratas quanto concretas, que cooperam entre si para produzir um projeto reusável para uma categoria específica de software. Uma outra definição foi apresentada por Willemann e Ibarra (2007, p. 41) que define um framework como sendo:

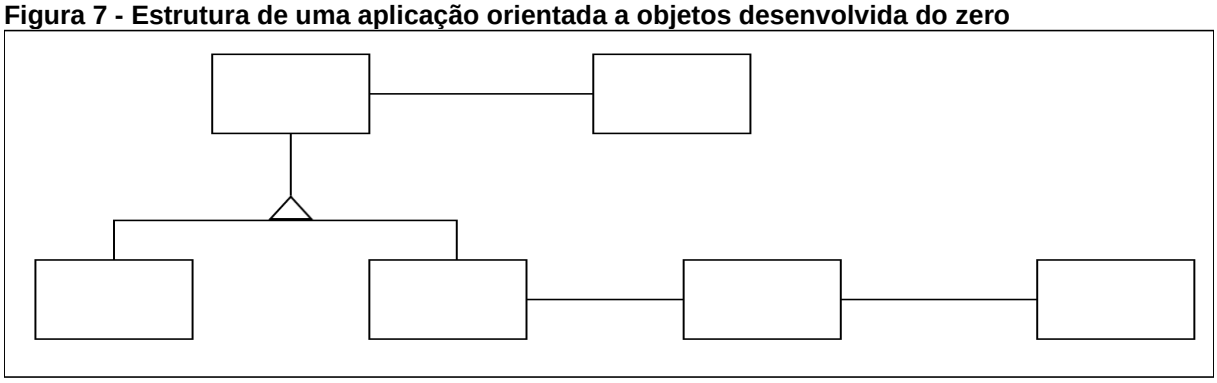
> Um framework ou arcabouço é uma estrutura de suporte definida em que outro projeto de software pode ser organizado e desenvolvido, quando se analisa o conceito no âmbito do desenvolvimento de software. Um framework pode incluir programas de suporte, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software (WILLEMANN; IBARRA, 2007, p. 41).

De uma forma geral, um framework pode ser definido conceitualmente como sendo um arcabouço de funcionalidades e artefatos organizados dentro de uma arquitetura condizente com o seu domínio de aplicação.

As principais definições de frameworks encontradas na literatura enaltecem uma de suas principais características, o reuso. Esta característica é fundamental para se justificar o porquê de se utilizar um framework como apoio no desenvolvimento de um determinado software.

As linguagens orientadas a objetos por padrão tendem a induzir o desenvolvedor a reusar as classes definidas no projeto, na maioria dos casos existem as bibliotecas que proporcionam a reutilização de forma explícita. Os frameworks surgem como uma extensão aperfeiçoada ao conceito das bibliotecas. As Figuras 7, 8 e 9 apresentam a estrutura de um software codificado sem auxílio de frameworks ou bibliotecas externas, um software desenvolvido com o auxílio de bibliotecas e um software com base na utilização de um framework, respectivamente.

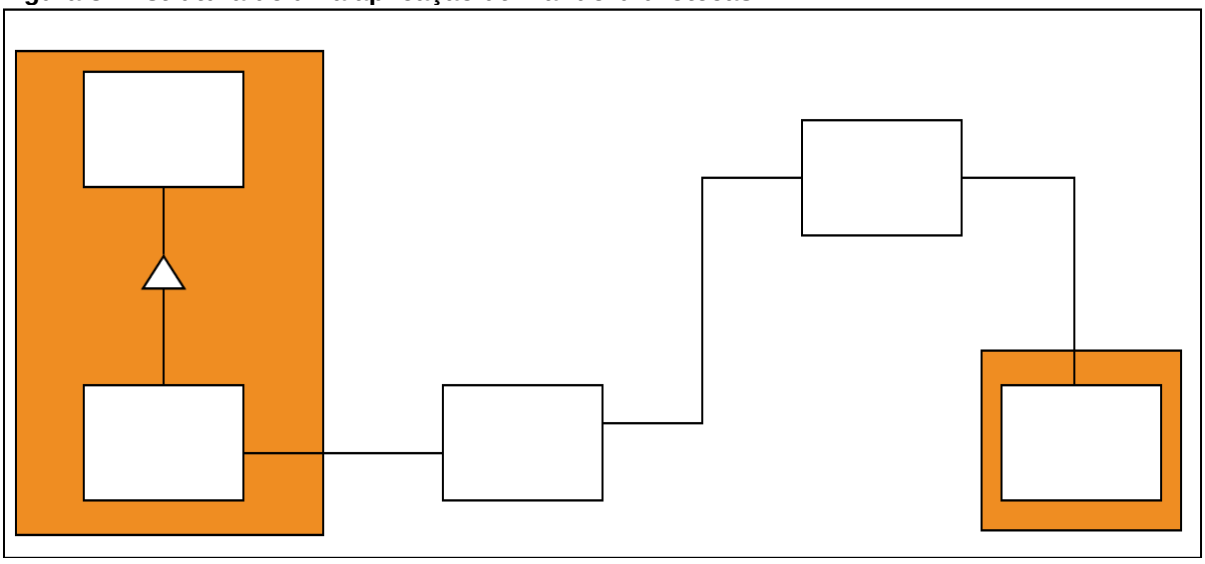
Figura 7 - Estrutura de uma aplicação orientada a objetos desenvolvida do zero



Fonte: Autoria própria

A estrutura apresentada na Figura 7 representa uma aplicação simples, que foi desenvolvida utilizando os conceitos definidos pela orientação a objetos, tais como herança e associação entre instâncias de classes. A Figura 8 apresenta uma alternativa para o desenvolvimento de sistemas utilizando bibliotecas.

Figura 8 - Estrutura de uma aplicação utilizando bibliotecas



Fonte: Autoria própria

O uso de bibliotecas como auxílio no desenvolvimento de software é uma prática constante entre os desenvolvedores e em diferentes paradigmas de programação.

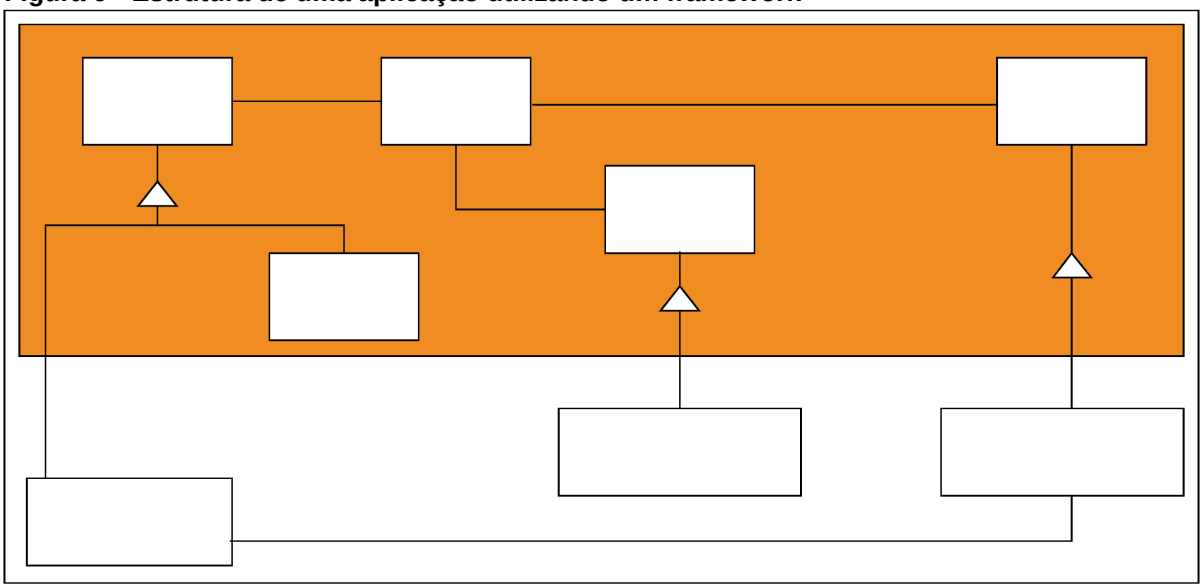
Na orientação a objetos o uso de bibliotecas proporciona um nível ainda mais alto de reuso, reduzindo consideravelmente o acoplamento na aplicação. A grande vantagem do uso de bibliotecas é reutilizar uma mesma solução em diferentes projetos evitando o retrabalho. Ao utilizar uma biblioteca o desenvolvedor

deve verificar se a mesma é nativa da linguagem utilizada no projeto, caso não seja deverá ficar atento e escolher uma versão que atenda a linguagem de programação utilizada e seguir as orientações de uso definidas pelo criador da mesma, tais como instalação e implementação dentro da aplicação. Um exemplo essencial para a linguagem Java é o pacote JDK (*Java SE Development Kit*) fornecido pela *Oracle* que é composto por um compilador e diversas bibliotecas que possuem classes com funções e métodos essenciais para o desenvolvimento de sistemas em Java.

A estrutura apresentada na Figura 8 representa uma aplicação simples desenvolvida com o uso de bibliotecas, as partes em branco representam as classes desenvolvidas internamente na aplicação e a parte em colorida representa as bibliotecas utilizadas como auxílio no desenvolvimento. Analisando a Figura 9 pode-se observar uma redução de códigos desenvolvidos internamente quando se comparado com a estrutura de aplicação desenvolvida do zero, apresentada na Figura 8, isso reduz o tempo gasto pelo desenvolvedor que pode reutilizar soluções já validadas pela comunidade de desenvolvedores (KERIEVSKY, 2008).

Geralmente as bibliotecas são utilizadas de forma isolada, ou seja, para cada necessidade encontrada ao longo do desenvolvimento pode-se utilizar uma determinada biblioteca que atende especificamente aquele problema, porém em um projeto de grande porte pode ser necessário o uso de uma quantidade excessiva de bibliotecas e isso pode dificultar o trabalho do desenvolvedor. Uma solução para este problema é unir várias bibliotecas comuns a um determinado domínio de aplicação em uma espécie de arcabouço que pudesse ser utilizado nos projetos, os frameworks surgiram como uma solução que emprega esses conceitos, a Figura 10 exemplifica como se constitui a estrutura de uma aplicação desenvolvida utilizando um framework (FAYAD; SCHMIDT; JOHNSON, 1999).

Figura 9 - Estrutura de uma aplicação utilizando um framework



Fonte: Autoria própria

A Figura 9 apresenta a estrutura básica de uma aplicação desenvolvida utilizando um framework, as partes coloridas representam o framework e a parte em branco representa as especificidades implementadas pelo desenvolvedor do software. Ao utilizar um framework o desenvolvedor encontrará uma estrutura já definida para a aplicação, essa estrutura fornece a implementação genérica para um domínio específico, a partir disso, o desenvolvedor pode utilizar os recursos disponibilizados pelo framework e focar seus esforços nas especificidades da sua aplicação. O exemplo da Figura 9 ilustra de forma superficial o modo como a aplicação fica organizada e possibilita demonstrar a redução de esforços gastos no desenvolvimento do software (FAYAD; SCHMIDT; JOHNSON, 1999).

A principal diferença entre frameworks e bibliotecas esta na forma em que ambas fornecem as classes e (ou) artefatos, enquanto a reutilização por meio das bibliotecas acontece de forma isolada (conforme apresentado na Figura 8) ficando na responsabilidade do desenvolvedor definir as interligações de acordo com suas necessidades. Já os frameworks fornecem uma estrutura completa de classes organizadas dentro de uma arquitetura particular, cabendo ao desenvolvedor se preocupar apenas com as especificidades da sua aplicação (SILVA, 2000).

É importante ressaltar que o uso de um framework não impede que o desenvolvedor utilize bibliotecas em sua aplicação, porém geralmente os frameworks fornecem os principais recursos para o desenvolvimento de um software

de acordo com seu domínio e, sendo assim, é importante sempre que se houver a necessidade de implementar uma nova funcionalidade consultar primeiramente se o framework já não a possui implementada para uso. Caso não encontre, pode optar por utilizar uma biblioteca ou mesmo desenvolver a funcionalidade. É importante destacar também que antes de optar pela utilização de um determinado framework é essencial realizar um estudo detalhado em relação às especificidades do software que será desenvolvido.

As próximas subseções abordam algumas das características que permitem categorizar um framework e possibilitar que o desenvolvedor opte pelo melhor que se adapta as necessidades do software em desenvolvimento e por fim, analisam-se algumas vantagens e as desvantagens do seu uso.

3.1.2 Classificação de frameworks

Existem várias características que permitem classificar um framework, para este trabalho foram selecionadas algumas que são consideradas importantes no momento em que o desenvolvedor precisa escolher o framework a ser utilizado para apoiar o desenvolvimento de seu software, neste caso são discutidas as classificações em relação ao reuso do framework e as diferentes dimensões que o caracterizam.

A classificação em relação ao processo de reuso dos frameworks refere-se a como ele é utilizado, por meio da sua instanciação ou extensão e Yassin e Fayad (2000) o classifica da seguinte forma:

- Caixa-branca: No reuso caixa-branca o framework é utilizado como base e permite que seu conjunto de classes seja implementado pelas subclasses da aplicação. Neste caso, o desenvolvedor pode adicionar as funcionalidades específicas da aplicação implementando pontos flexíveis da superclasse do framework. Neste tipo de reuso é necessário que o desenvolvedor possua um conhecimento aprofundado do funcionamento do framework utilizado para que consiga gerar as subclasses, além de exigir esforço para se desenvolver o código.
- Caixa-preta: No reuso caixa-preta o framework é utilizado como base e fornece ao desenvolvedor apenas uma interface, onde os pontos

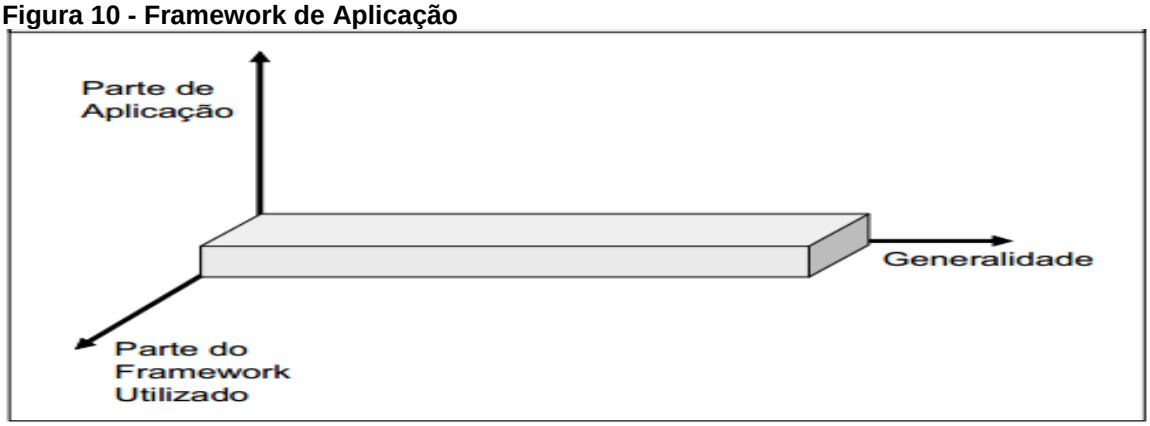
flexíveis são formados por uma composição de objetos. Neste caso, o desenvolvedor não precisa ter conhecimento em relação a implementação interna das classes do framework, basta conhecer a interface disponibilizada. Isso torna o reuso mais fácil por parte do desenvolvedor, porém este tipo de reuso é menos flexível quando comparado ao reuso do tipo caixa-branca.

- Caixa-cinza: Este último é considerado como sendo híbrido, tendo em vista que no caixa-cinza o reuso pode ser feito por meio de herança e ligações dinâmicas (assim como no reuso do tipo caixa-branca) e também por meio da utilização das interfaces disponibilizadas pelo framework base (assim como no reuso do tipo caixa-preta). O objetivo deste tipo de reuso é contornar as dificuldades apresentadas pelos tipos anteriormente citados, tornando o reuso mais flexível e ao mesmo tempo reduz a complexidade durante a extensão.

Os frameworks ainda podem ser caracterizados de acordo com diferentes dimensões, onde um pertence a um propósito distinto, ou seja, onde ele é aplicado. Um framework pode pertencer a uma das seguintes dimensões: aplicação (infra-estrutura), domínio ou suporte (CARVALHO SOBRAL, 2015).

Os frameworks de aplicação são compostos por um conjunto de funcionalidades que podem ser aplicados em diferentes domínios. O objetivo deste tipo de framework é prover uma estrutura para a solução das necessidades da aplicação e não a solução propriamente dita, os frameworks utilizados no desenvolvimento de interfaces (GUI) de sistemas é um exemplo. A Figura 10 apresenta o exemplo da estrutura de um framework de aplicação.

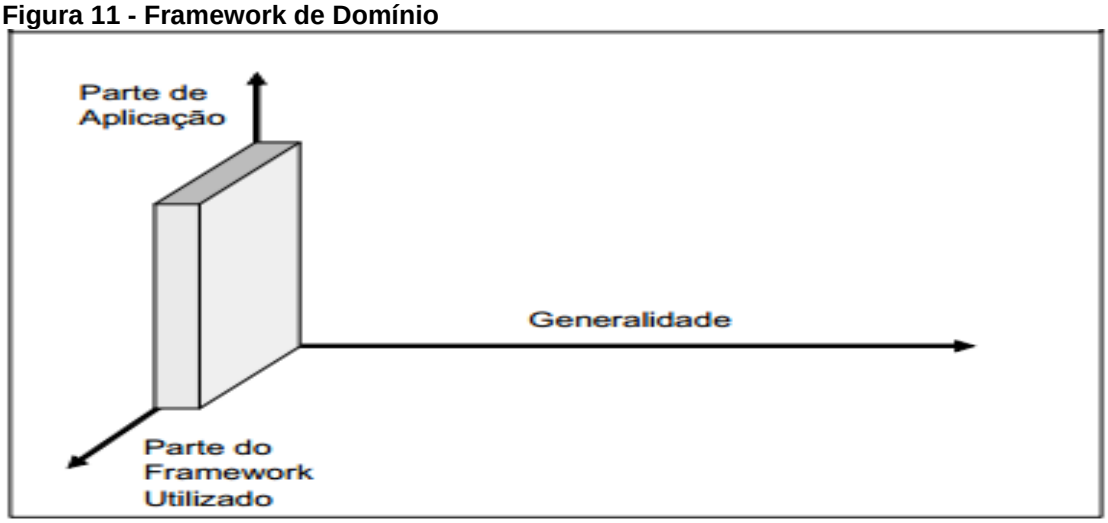
Figura 10 - Framework de Aplicação



Fonte: Adaptado de Willemann e Ibarra (2007)

Os frameworks de domínio englobam um conjunto de funcionalidades aplicáveis a um domínio específico, sendo que o domínio pode ser entendido como a sua área de atuação, como exemplo, frameworks para uma determinada linguagem de programação, redes, construção de interfaces (GUI) ou na manipulação de banco de dados (BD), como exemplo o Hibernate (WILLEMAN; IBARRA, 2007). A Figura 11 apresenta o exemplo de estrutura de um framework de domínio.

Figura 11 - Framework de Domínio



Fonte: Adaptado de Willemann e Ibarra (2007)

Os frameworks de suporte são menos encontrados em relação aos citados anteriormente, assim como os frameworks de aplicação eles são independentes de domínio e seu objetivo é fornecer serviços em nível de sistema operacional, como o acesso a arquivos e dispositivos (WILLEMAN; IBARRA, 2007).

O framework utilizado como base para este trabalho se enquadra na segunda dimensão apresentada como de domínio.

3.1.3 Vantagens e Desvantagens

Ao optar pela utilização de um framework o desenvolvedor deixa de se preocupar exclusivamente com a aplicação em desenvolvimento e volta as atenções para as aplicações futuras que seguem uma estrutura padrão (SOMMERVILLE, 2008). Dentre os benefícios da utilização dos frameworks destacam-se (FOWLER, 1999):

- A redução da codificação por parte do desenvolvedor para aplicações análogas.
- A redução da codificação do processo de desenvolvimento torna-se menos dispendioso e as entregas aos clientes mais rápidas.
- Os softwares desenvolvidos tornam-se mais modulares, simplificando a sua extensão e integração com outras aplicações.
- O processo de manutenção do software se torna menos complexo, visto que, o framework fornece uma estrutura de classes padronizadas para o software, permitindo que a manutenção se limite as classes específicas da aplicação desenvolvida.
- Torna o software longevo, pois aumenta a sua capacidade de evolução.

A utilização dos frameworks também pode apresentar desvantagens dependendo da situação, podendo se destacar (FOWLER, 1999):

- Inicialmente a utilização de um framework pode consumir muito tempo devido a sua curva de aprendizado, dependendo do caso, torna-se inviável.
- Alguns erros podem gerar mais dificuldade de se identificar a origem, se acontece dentro da estrutura do framework ou na da aplicação.
- A escolha de um framework sem analisar o domínio ao qual se aplica, pode resultar em um uso inconsistente, difícil de manter e que acaba não aproveitando a maioria das suas vantagens.

3.2 FRAMEWORK DE FORMAÇÃO DE PREÇO DE VENDA (FRAMEMK)

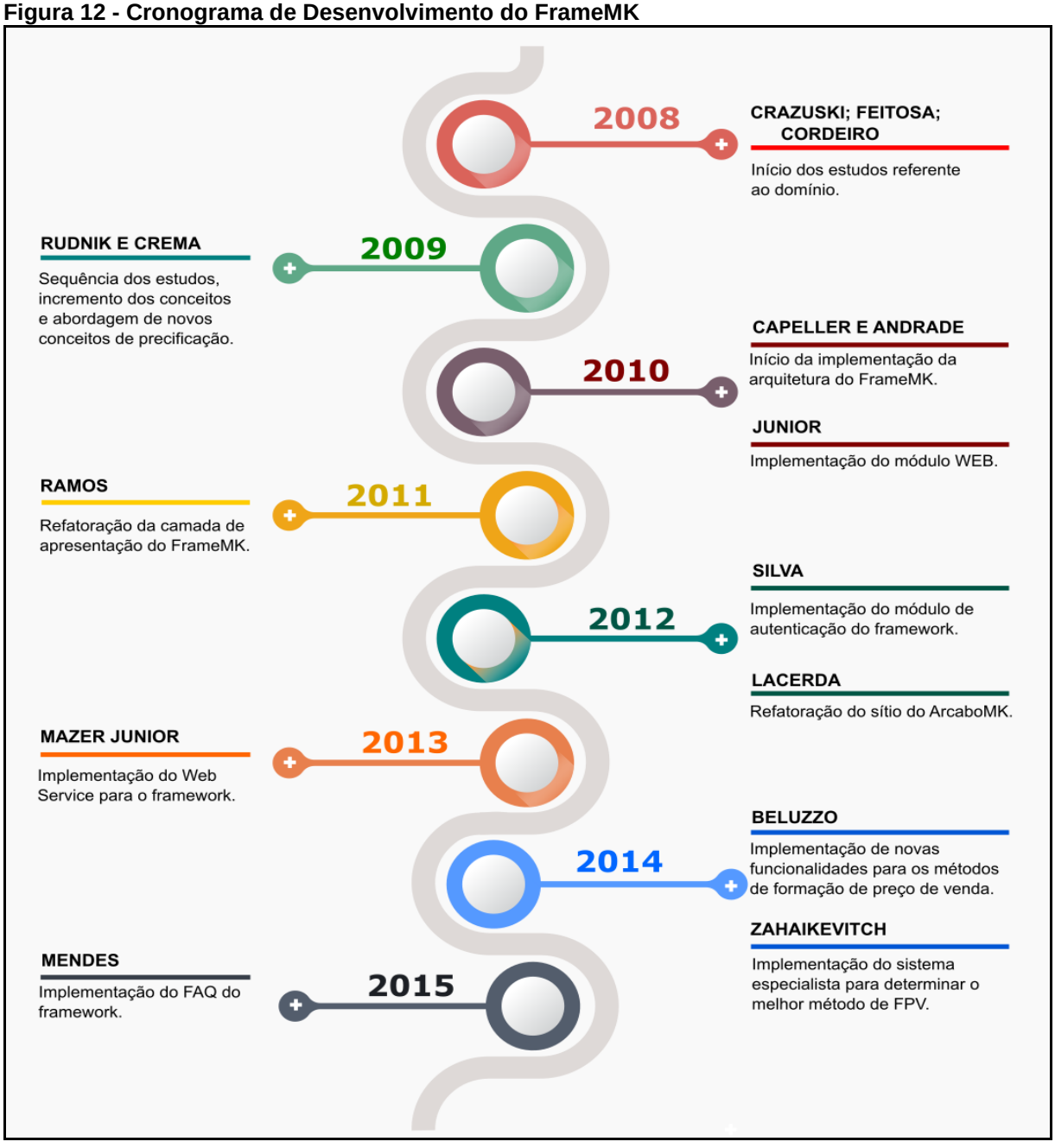
O FrameMK é um projeto desenvolvido pela Linha de Pesquisa de Engenharia de Software (LPES) do Grupo de Pesquisa em Sistemas de Informação (GPSI) da Universidade Tecnológica Federal do Paraná.

O projeto surgiu com o objetivo de propor um modelo de arquitetura de um framework de domínio na área de formação de preço de venda (MAZER JUNIOR, 2013). Inicialmente notou-se a carência de aplicativos voltados para a área de cálculos do preço de venda de produtos e serviços, esta necessidade motivou o desenvolvimento do projeto que tem como característica importante a

implementação de vários métodos de formação de preço de venda (FPV), o método do SEBRAE, Custo Pleno.

Ao longo dos anos professores e acadêmicos desenvolveram trabalhos que contribuíram para a evolução do FrameMK. A Figura 12 apresenta uma linha cronológica que exibe as fases que conduziram o projeto até o seu estado atual. Logo após, estas informações são complementadas, detalhando cada trabalho desenvolvido durante o período.

Figura 12 - Cronograma de Desenvolvimento do FrameMK



Fonte: Autoria própria

Crazuski, Feitosa e Cordeiro (2008) iniciaram o estudo do domínio, coletando informações em relação aos pontos de estabilidade e flexibilidade utilizando o processo dirigido por responsabilidades. Neste trabalho foi realizado a modelagem dos três métodos de FPV que compõem o framework.

Os estudos prosseguiram em 2009, o trabalho proposto por Rudnik e Crema (2009) estudou o domínio e complementou a definição dos conceitos sobre a FPV e apresentou novos conceitos em relação ao lucro, dois novos métodos foram implementados, sendo o método do Custeamento Marginal e o do Retorno Sobre o Capital (ROIC).

Em 2010, Capeller e Andrade (2010) deram início a construção da arquitetura do FrameMK, a proposta utilizava três métodos de FPV, Custo Pleno, Custo Baseado em Atividade e SEBRAE que já haviam sidos modelados em trabalhos anteriores. No mesmo ano, Junior (2010) desenvolveu um trabalho voltado para WEB que auxiliou na implementação do método tendo como base decisões de empresas concorrentes.

Em 2011, o FrameMK passou a operar em plataforma WEB, o trabalho apresentado por Ramos (2011) baseou-se em trabalhos anteriores para refatorar a camada de apresentação do framework.

Em 2012, Silva (2012) implementou o módulo de autenticação do FrameMK e o controle de acesso dos usuários. O método desenvolvido identifica aspectos na fase de análise usando uma matriz de adjacência. Estes aspectos são identificados a partir das similaridades com requisitos não funcionais e sendo assim, o analista não precisa ter profundo conhecimento sobre aspectos. Ainda este ano, Lacerda (2012) refatorou o sítio ArcaboMK, através deste trabalho foi possível implementar a estrutura que armazena os trabalhos realizados pelo GPSI e acadêmicos envolvidos no projeto.

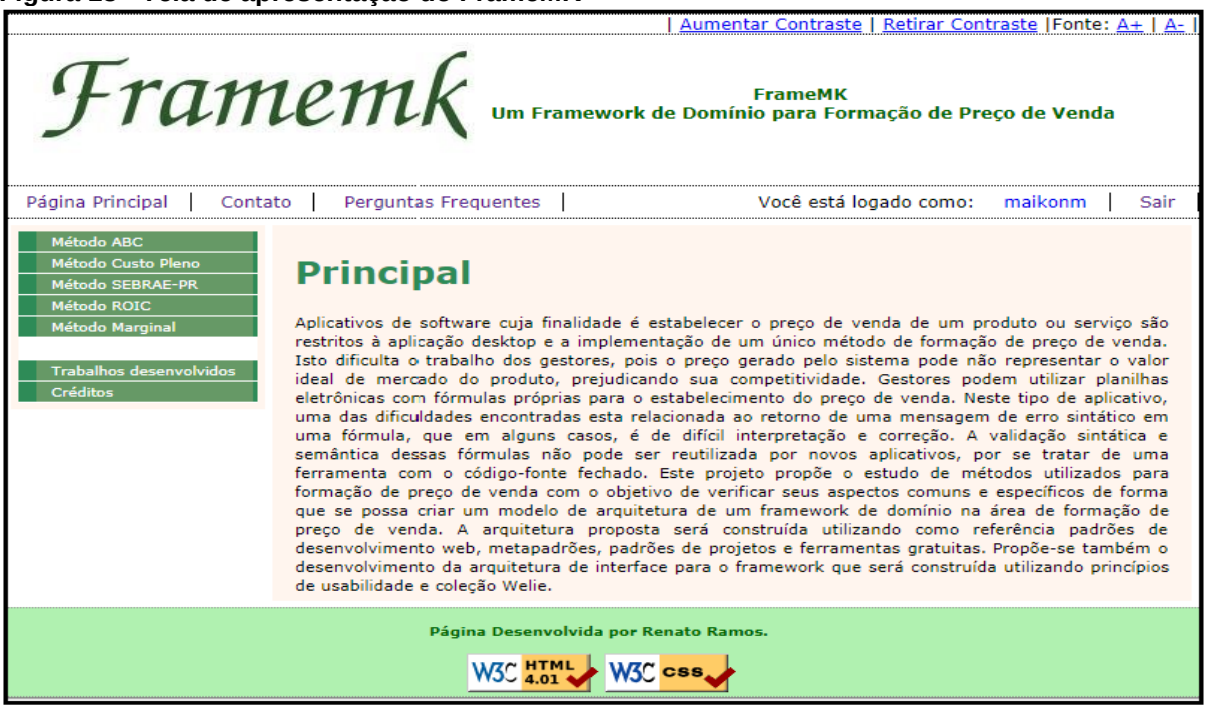
Em 2013, Mazer Junior (2013) implementou o *web service* que é utilizado no FrameMK. Em 2014, Barbosa e Beluzzo (2014) estenderam o FrameMK incluindo novas funcionalidade para o método ABC além de modelar os métodos Marginal e ROIC. Ainda neste ano, Zahaikevitch (2014) propôs em seu trabalho um sistema especialista, capaz de determinar o melhor método de FPV através da análise das características das empresas.

Em 2015, Mendes (2015) realizou um estudo sobre Linhas de Produto de Software (LPS) e escolheu três produtos no domínio de *Frequently Asked Questions*

(FAQ) como estudo: o jornal O Globo, as lojas Americanas e o FrameMK. Após a realização deste estudo foi possível implementar o FAQ do FrameMK.

A Figura 13 apresenta a tela inicial do FrameMK, após o usuário realizar o cadastro e se autenticar no sistema serão apresentadas as informações básicas juntamente com os métodos de FPV que podem ser escolhidos para a realização dos cálculos.

Figura 13 - Tela de apresentação do FrameMK

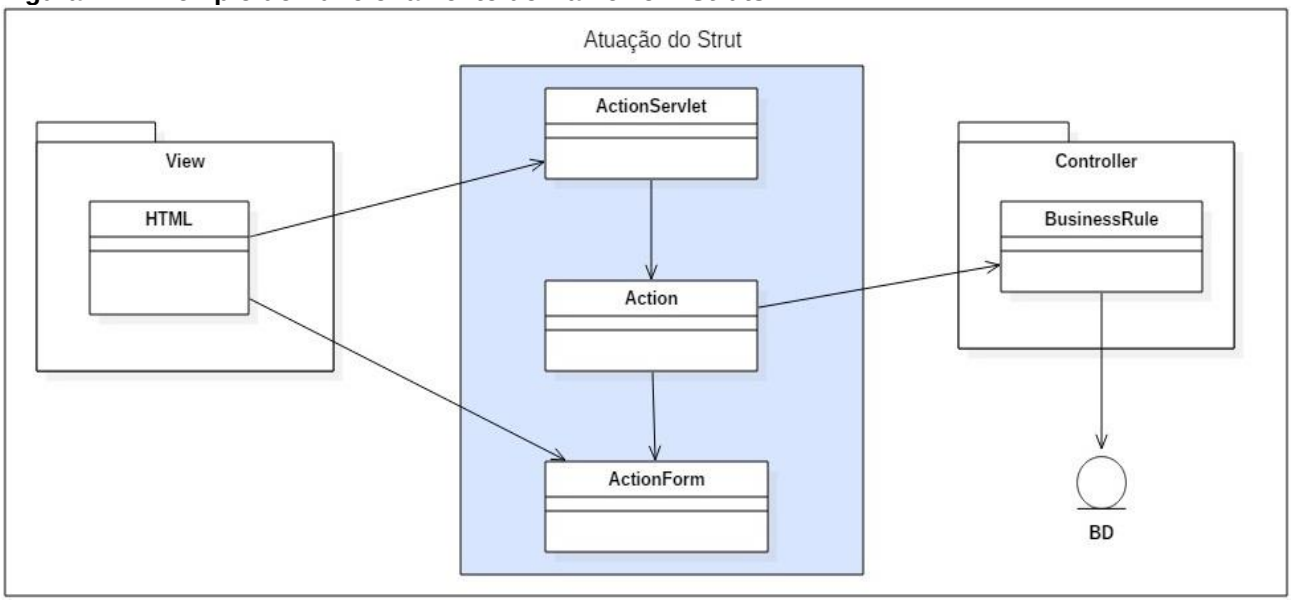


Fonte: Autoria própria

3.2.1 Arquitetura FRAMEMK

O FrameMK foi desenvolvido utilizando a linguagem de programação Java e a sua estrutura foi projetada utilizando o framework de aplicação *Struts*. O *Struts* fornece uma estrutura inicial para a aplicação, suas principais responsabilidades são a comunicação e integração entre as camadas de visualização da aplicação através da implementação de um controlador. A Figura 14 apresenta como o *Struts* atua dentro da aplicação.

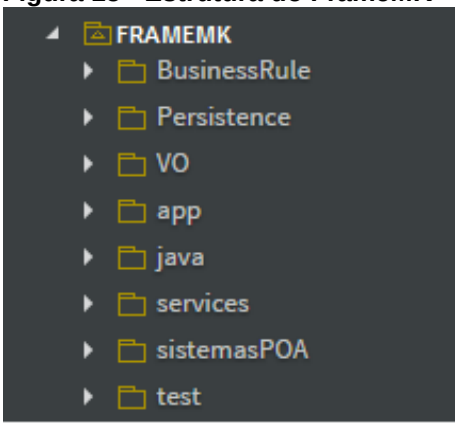
Figura 14 - Exemplo do Funcionamento do framework Struts



Fonte: Autoria própria

Os trabalhos desenvolvidos pelos acadêmicos foram incrementando a estrutura do FrameMK ao longo dos anos, o framework possui uma estrutura bem definida tanto para a aplicação *desktop* quanto para a sua versão Web. A estrutura do FrameMK é apresentada na Figura 15.

Figura 15 - Estrutura do FrameMK



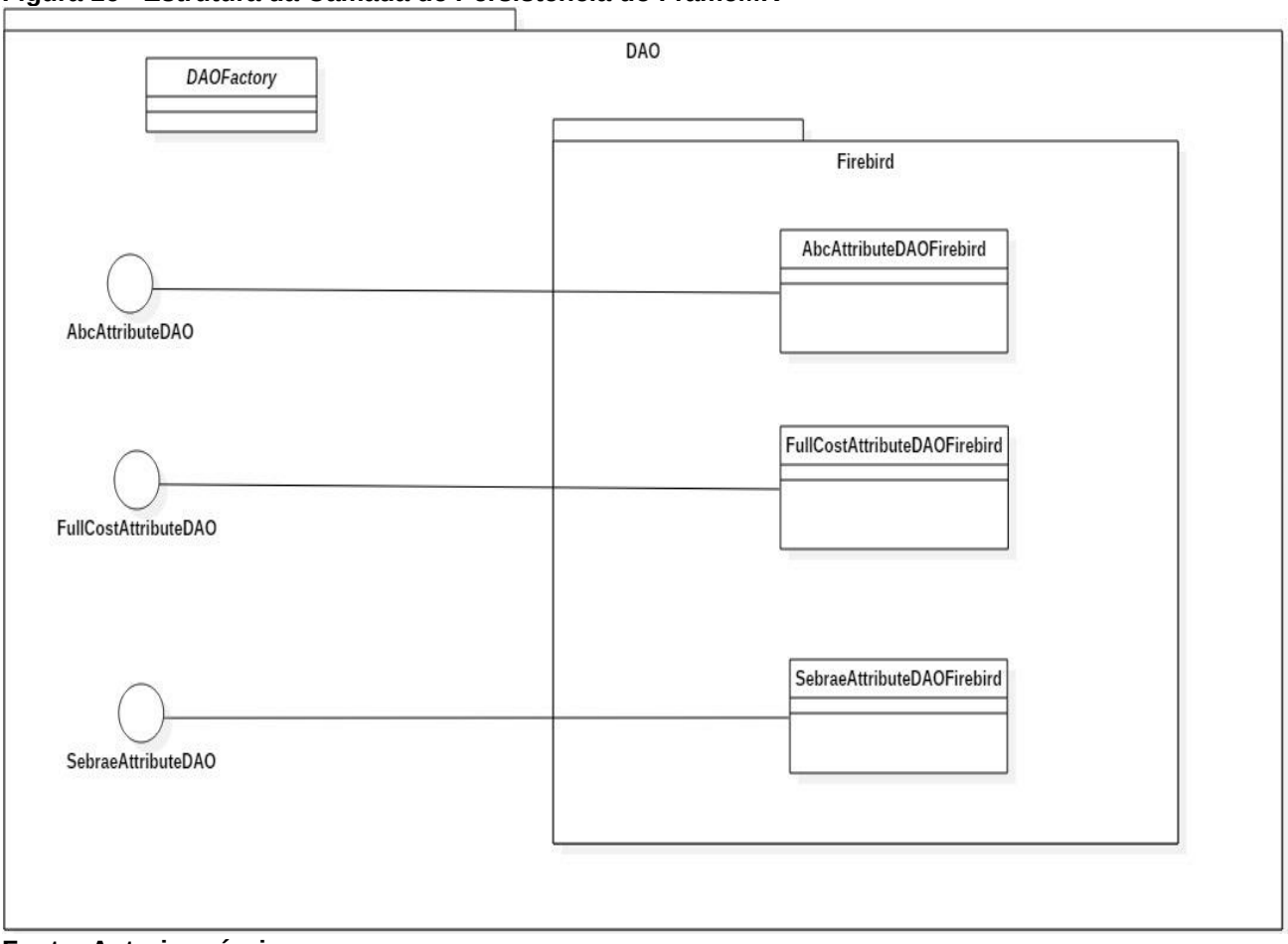
Fonte: Autoria própria

O pacote *BusinessRule* contém a lógica da aplicação, as classes implementadas neste pacote executam as funcionalidades definidas em cada método de FPV. No pacote *VO* os objetos que compõem os métodos são definidos, o pacote *app* contém as classes responsáveis pela interface da aplicação. O pacote *services* abrange os serviços referentes ao *web-service* da aplicação. O pacote

sistemasPOA contém o módulo de *login* utilizado no FrameMK que é baseado em aspectos, por meio dele é feito o controle de usuários da aplicação. O pacote de *test* possui as classes de testes da *Web service* do sistema. Por fim, o pacote *Persistence* contempla as classes responsáveis pela conexão e a comunicação com o banco de dados e armazena a camada que será refatorada utilizando o método de Barros (2015), definido no capítulo anterior.

A Figura 16 apresenta a estrutura do pacote de persistência, para esta representação foram selecionadas as classes abstratas responsáveis pela definição dos atributos dos métodos para FPV, *AbcAttributeDAO*, *FullCostAttributeDAO*, *SebraeAttributeDAO*. A estrutura apresentada é detalhada na sequência.

Figura 16 - Estrutura da Camada de Persistência do FrameMK



Fonte: Autoria própria

A classe abstrata *DAOFactory* contém os métodos (abstratos) *getABCAttributeDAO(), getSebraeAttributeDAO(), getFullCostAttributeDAO(),* estes retornam os atributos pertencentes a cada um dos três métodos de FPV utilizados no FrameMK. Ainda dentro do pacote DAO são apresentadas as interfaces que

determinam os cabeçalhos referentes às funções que definem os atributos para os métodos de FPV. O pacote *Firebird* contempla as classes que interagem diretamente com o banco de dados, estas implementam os métodos das interfaces *AbcAttributeDAO, FullCostAttributeDAO e SebraeAttributeDAO* que determinam os atributos para os métodos de FPV, ABC, Custo Pleno e Sebrae, respectivamente.

3.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo abordou as vantagens do uso de frameworks no desenvolvimento de software. Foram apresentados os tipos de reuso (caixa branca, caixa preto ou híbrido) e sua classificação (suporte, aplicação ou de domínio). Este trabalho utiliza como estudo de caso para refatoração um framework de domínio denominado de FrameMK.

O FrameMK é um projeto que está sendo desenvolvido por vários alunos de graduação e mestrado do Grupo de Pesquisa em Sistemas de Informação da Universidade Tecnológica Federal do Paraná. Como existe rotatividade de pessoas no projeto, houve a necessidade de se refatorar o código da camada persistência. O processo de refatoração desta camada é descrito no próximo capítulo.

4 REFATORAÇÃO DA CAMADA DE PERSISTÊNCIA DO FRAMEMK

Este capítulo tem como objetivo apresentar as adaptações propostas ao método de Barros (2015) e posteriormente, as refatorações aplicadas na camada de persistência do FrameMK utilizando o método proposto. A seção 4.1 apresenta o fluxograma original proposto por Barros (2015), complementado pelo fluxograma adaptado para este trabalho. A seção 4.2 descreve as classes da camada de persistência do framework, em seguida apresentam-se os quadros com as técnicas aplicadas. A seção 4.3 relata as considerações finais do capítulo.

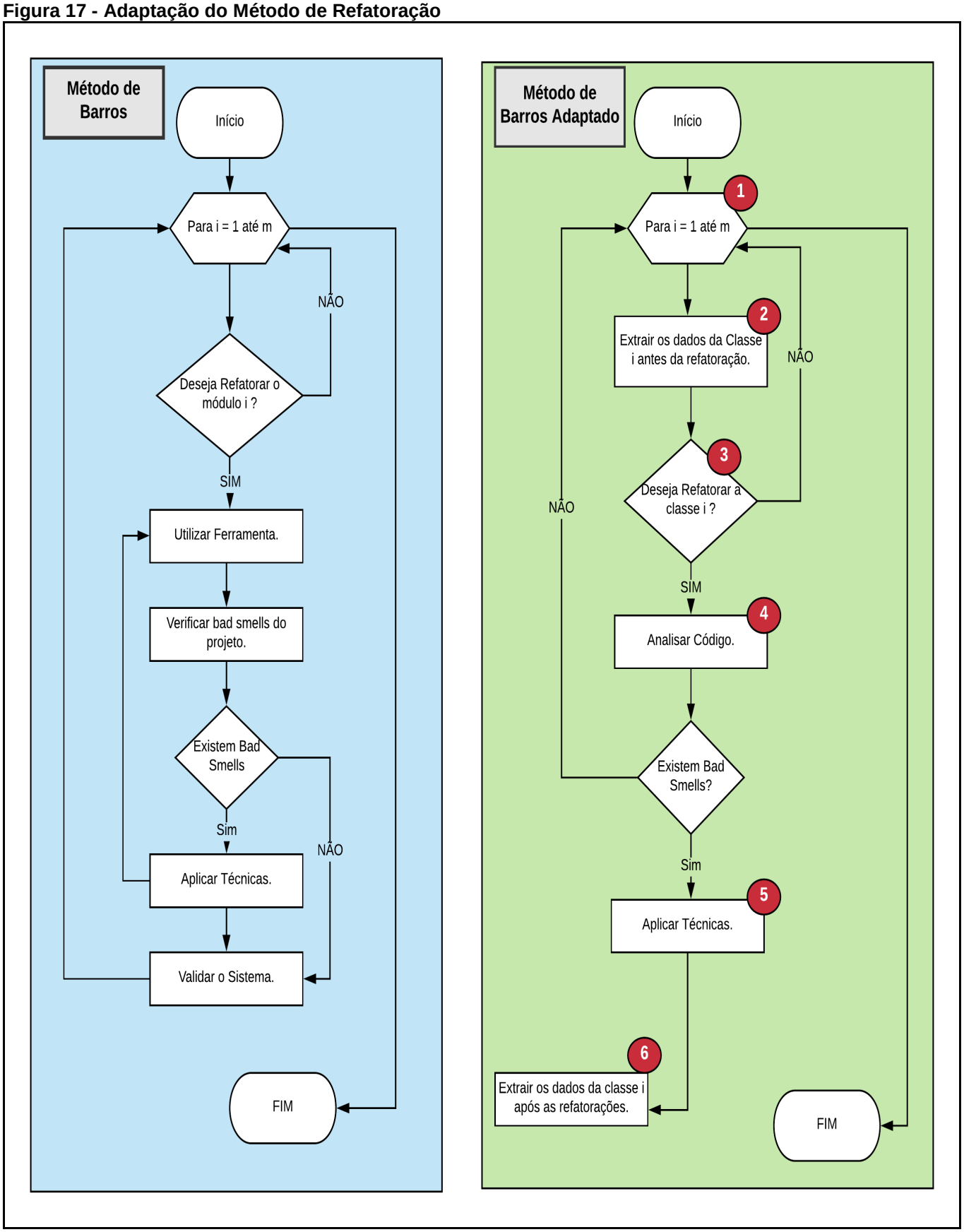
4.1 PROCESSO PROPOSTO PARA A APLICAÇÃO DAS TÉCNICAS

Por meio dos estudos realizados sobre os métodos de refatoração, descritos no Capítulo 2, verificou-se que o método proposto por Barros (2015) é mais adequado para o caso de teste usado neste trabalho pelos seguintes fatores:

- Por ser um método iterativo, podendo percorrer as camadas ou classes de um software.
- O método é aplicável em frameworks de domínio, característica específica do caso de teste deste trabalho.
- O método de Barros (2015) foi desenvolvido utilizando o FrameMK como caso de testes.
- As etapas do método proposto por Barros (2015) se mostraram mais flexíveis ao ser comparado ao proposto por Rapeli (2006) e Mens e Tourwé (2004).

Os ajustes realizados no fluxograma do método são apresentados na Figura 17 e detalhados na sequência.

Figura 17 - Adaptação do Método de Refatoração



Fonte: Autoria própria

O método de refatoração escolhido foi justificado por suas características que permitem otimizar o processo de refatoração para o FrameMK, contudo, alguns ajustes foram necessários visando adaptar o processo de refatoração proposto pelo método de Barros (2015), estes ajustes foram enumerados na Figura 17 e seguem descritos:

1) Barros (2015) propõe em seu método a iteração em todas as camadas do framework, o método adaptado itera entre as classes da camada de persistência do framework.
2) O método adaptado apresenta um novo passo, “Extrair os dados da classe i antes da refatoração” que funciona da seguinte forma: ao percorrer a camada de persistência do framework realiza-se a extração do número de métodos e linhas de código de cada uma das classes que compõem a camada antes de se aplicar as refatorações. Essas informações são gravadas em uma tabela com o objetivo de apresentar um comparativo desses valores após a aplicação das técnicas.
3) O método original verifica se um módulo será refatorado, a adaptação proposta verifica se a classe deve receber as refatorações.
4) O passo “Analisar Código” substitui os passos “Utilizar Ferramenta” e “Verificar *bad smells* do projeto” do método original. Como este trabalho foca na refatoração de uma camada específica do framework, foi optado por não utilizar uma ferramenta para a detecção de *bad smells* no código. Cada classe da camada foi analisada manualmente para se detectar as necessidades de refatorações, para que tornar isso possível foi necessário estudar as 91 técnicas catalogadas por Fowler (1999) e as 24 propostas por Kerievsky (2004).
5) A diferença do passo “Aplicar Técnicas” encontra-se na metodologia utilizada para se identificar as necessidades das refatorações, visto que o processo adaptado age baseando-se no conhecimento obtido com o estudo dos catálogos.
6) O passo “Extrair os dados da classe i após a refatoração” complementa o passo apresentado no item 2 dessa lista, extraindo os

dados de cada classe após as refatorações, completando a tabela utilizada para quantificar o número de métodos e linhas de código das classes durante o processo de refatoração.

A aplicação do método adaptado no estudo de caso é descrita na próxima seção.

4.2 REFATORAÇÃO DA CAMADA DE PERSISTÊNCIA DO FRAMEMK

A camada de persistência do FrameMK possui 12 classes, cada classe tem como responsabilidade implementar as funcionalidades de um dos métodos de FPV ou são voltadas para as configurações referentes a conexão com o banco de dados, conforme apresentado no Quadro 5.

Quadro 5 - Classes implementadas na camada de persistência

Classe	Implementação
AbcActivityDAOFirebird	Método ABC
AbcAttributeDAOFirebird	Método ABC
AbcLogValueDAOFirebird	Método ABC
AbcProductDAOFirebird	Método ABC
AbcProductionLineDAOFirebird	Método ABC
FBDAOFactory	Conexão com Banco de Dados
FullCostAttributeDAOFirebird	Método Full Cost
FullCostItemDAOFirebird	Método Full Cost
FullCostLogValueDAOFirebird	Método Full Cost
ProductDAOFirebird	Gerenciamento de Produtos
SebraeAttributeDAOFirebird	Método SEBRAE
SebraeLogValueDAOFirebird	Método SEBRAE

Fonte: Autoria própria

As técnicas aplicadas na camada de persistência foram separadas de acordo com as categorias definidas no catálogo de refatorações proposto Fowler (FOWLER, 1999). Cada categoria é apresentada em um quadro, contendo as técnicas aplicadas e as classes onde foram aplicadas. Cada quadro é complementado com informações que justificam o uso de cada técnica, além de apresentar trechos dos códigos refatorados.

Quadro 6 - Categoria Encapsulation: Técnicas Aplicadas

Categoria: Encapsulation	**Técnicas:** Encapsulate Field, Hide Method.
Classes Refatoradas	
AbcActivityDAOFirebird, AbcAttributeDAOFirebird, AbcProductDAOFirebird, AbcProductionLineDAOFirebird, FullCostAttributeDAOFirebird, FullCostItemDAOFirebird, ProductDAOFirebird e SebraeAttributeDAOFirebird.	

Fonte: Autoria própria

O catálogo de técnicas proposto por Fowler (1999) ressalta a importância de encapsular os dados do seu projeto quando se trabalha com orientação a objetos, um exemplo de encapsulamento citado é a utilização de *getters* e *setters*. O Quadro 5 apresenta algumas das classes em que foram aplicadas técnicas da categoria *Encapsulation*.

A Figura 18 apresenta a técnica *Encapsulate Field* utilizada para encapsular campos. A aplicação da técnica consiste em proteger a visibilidade do atributo “sql” e posteriormente implementar os métodos assessores “getSql” e “setSql”, garantindo que o atributo fique acessível internamente na classe e por meio de objetos da classe onde o mesmo se encontra, utilizando os métodos acessores.

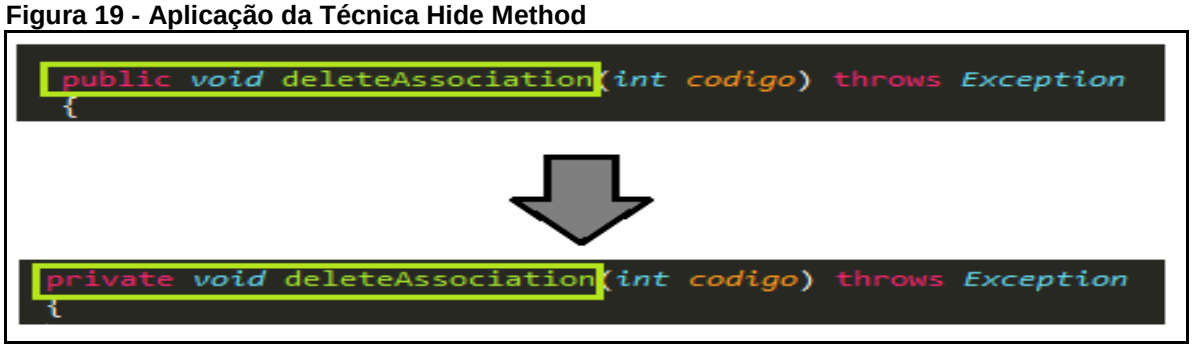
Figura 18 - Aplicação da Técnica Encapsulate Field

```
private static StringBuilder sql = new StringBuilder();

public static StringBuilder getSql() {
    return sql;
}

public static void setSql(StringBuilder aSql) {
    sql = aSql;
}
```

Fonte: Autoria própria

A Figura 19 apresenta a aplicação da técnica *Hide Method*, que também pertence à categoria *Encapsulation*. A técnica consiste em alterar a forma como o método “deleteAssociation” é apresentado, tendo em vista que o mesmo é de uso exclusivo da classe.

Figura 19 - Aplicação da Técnica Hide Method



Fonte: Autoria própria

Fowler (1999) apresenta em seu catálogo uma série de técnicas pertencentes a categoria *Class Extraction* que visam melhorar o código desacoplando classes, interfaces, módulos e até mesmo conjunto de parâmetros. Uma dessas técnicas foi utilizada para refatorar algumas classes do framework, conforme descrito no Quadro 7.

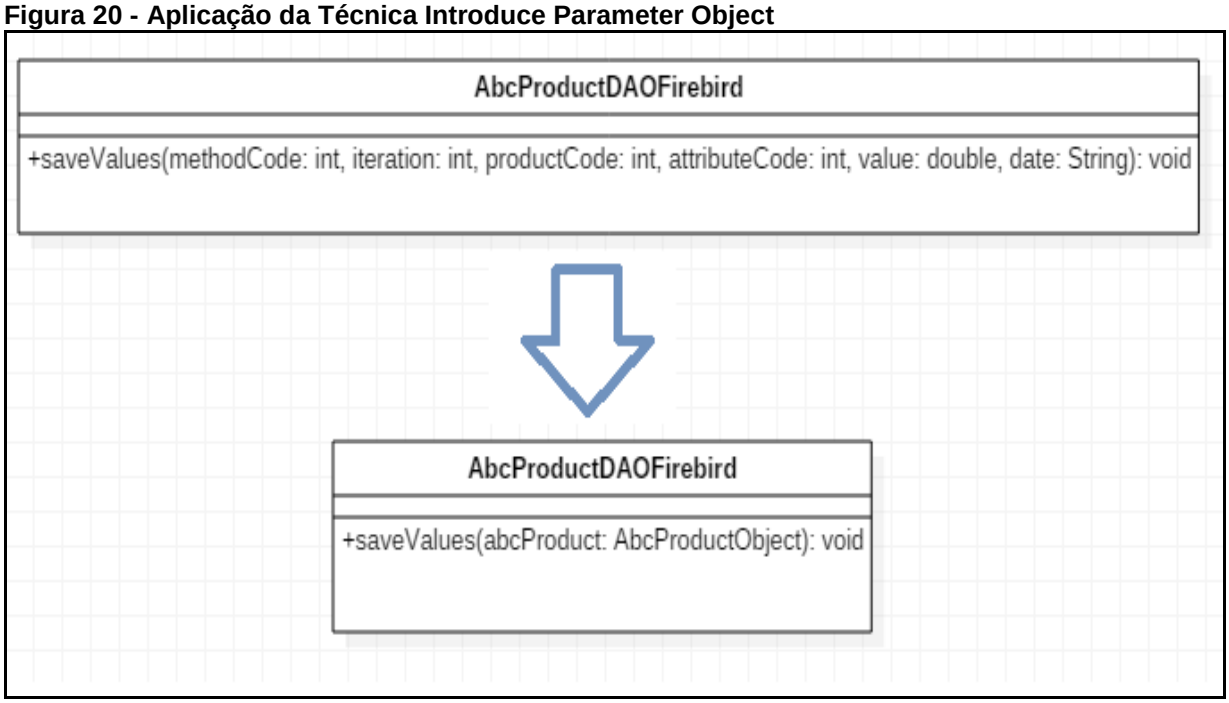
Quadro 7 - Categoria Class Extraction: Técnicas Aplicadas

Categoria: Class Extraction	**Técnicas:** Introduce Parameter Object
Classes Refatoradas	
AbcProductDAOFirebird, FullCostItemDAOFirebird e FullCostLogValueDAOFirebird.	

Fonte: Autoria própria

A Figura 20 apresenta a aplicação de uma das técnicas propostas por Fowler, pertencente à categoria *Class Extraction*. A aplicação da técnica proporcionou redefinir os escopos das classes de forma mais “amigável”, a definição de um objeto parametrizável possibilitou o reuso em todos os métodos da classe, fazendo com que a passagem de parâmetros acontecesse de forma mais “limpa” visualmente.

Figura 20 - Aplicação da Técnica Introduce Parameter Object



Fonte: Autoria própria

O Quadro 8 apresenta a categoria *Local Variables*, as técnicas apresentadas por Fowler nesta categoria atuam na remoção de declarações de variáveis desnecessárias, o autor sugere analisar o código buscando verificar as variáveis declaradas localmente nos métodos.

Quadro 8 - Categoria Local Variables: Técnicas Aplicadas

Categoria: Local Variables	**Técnicas:** Extract Variable
Classes Refatoradas	
AbcActivityDAOFirebird, AbcAttributeDAOFirebird, AbcLogValueDAOFirebird, AbcProductDAOFirebird, AbcProductionLineDAOFirebird, FullCostAttributeDAOFirebird, FullCostItemDAOFirebird, ProductDAOFirebird e SebraeAttributeDAOFirebird.	

Fonte: Autoria própria

Conforme apresentado na Figura 21, as classes de persistência continham uma grande quantidade de variáveis e objetos declarados internamente em seus métodos, principalmente objetos voltados para conexão com o banco de dados.

Figura 21 - Código com variáveis internas nos métodos

```
public ResultSet getProduct(int codeAttribute) throws SQLException{


    Connection con = FBDAOFactory.connect();

    Statement stm = con.createStatement();

    ResultSet rs = stm
            .executeQuery("select produto_codigo, valor " +
                    " from atributo_produto " +
                    " where atributo_codigo = "
                    + codeAttribute);

    return rs;
}
```

Fonte: Autoria própria

A Figura 22 apresenta a aplicação da técnica *Extract Variable,* conforme apresentado na Figura 21, as classes apresentam excesso de declarações desnecessárias, a técnica foi aplicada em diversas classes conforme apresentado no Quadro 8.

Figura 22 - Aplicação da Técnica Extract Variable

```
protected static Connection con;
protected static Statement stm;
protected static ResultSet rs;

public ResultSet getProduct(int codeAttribute) throws SQLException{

    con = FBDAOFactory.connect();

    stm = con.createStatement();

    rs = stm
            .executeQuery("select produto_codigo, valor " +
                    " from atributo_produto " +
                    " where atributo_codigo = "
                    + codeAttribute);

    return rs;
}
```

Fonte: Autoria própria

A categoria *Vendor Libraries* catalogada por Fowler (1999) apresenta técnicas voltadas para o reuso, utilizando métodos externos ou mesmo bibliotecas

terceiras que atendem a necessidade do desenvolvedor. O Quadro 9 exibe quais classes utilizam a técnica *Vendor Libraries*.

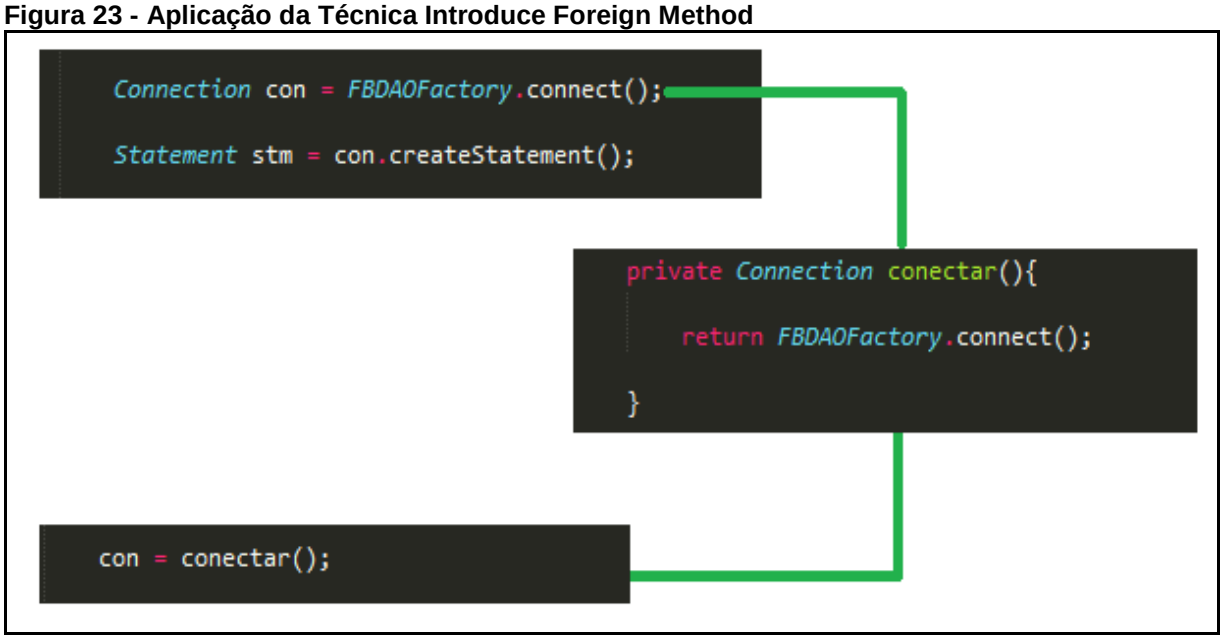
Quadro 9 - Categoria Vendor Libraries: Técnicas Aplicadas

Categoria: Vendor Libraries	**Técnicas:** Introduce Foreign Method
Classes Refatoradas	
AbcActivityDAOFirebird, AbcAttributeDAOFirebird, AbcLogValueDAOFirebird, AbcProductDAOFirebird, AbcProductionLineDAOFirebird, FullCostAttributeDAOFirebird, FullCostItemDAOFirebird, ProductDAOFirebird e SebraeAttributeDAOFirebird.	

Fonte: Autoria própria

A Figura 23 apresenta um exemplo na classe “AbcActivityDAOFirebird”, a necessidade de estabelecer conexão com o banco de dados acaba fazendo com que seja acionado o objeto de conexão “FBDAOFactory” e, posteriormente habilitando a conexão por meio de um de seus métodos. Isso ocorre inúmeras vezes na camada de persistência do framework, a utilização da técnica *Introduce Foreign Method* moveu essa chamada para um método externo, tornando-o acessível para todos os demais métodos da classe, de forma simplificada.

Figura 23 - Aplicação da Técnica Introduce Foreign Method



Fonte: Autoria própria

As técnicas apresentadas na categoria *Method Calls* agem diretamente nos métodos. Fowler (1999) define técnicas para gerenciar parâmetros, alterar

parametrizações e visibilidade. O Quadro 10 ilustra as classes do FrameMK em que foi aplicada o método *Method Calls*.

Quadro 10 - Categoria Method Calls: Técnicas Aplicadas

Categoria: Method Calls	**Técnicas:** Separate Query from Modifier
Classes Refatoradas	
AbcActivityDAOFirebird.	

Fonte: Autoria própria

A Figura 24 apresenta um método que realiza duas operações distintas no banco de dados, isso obriga o desenvolvedor realizar uma verificação por meio de um atributo passado como parâmetro, para saber qual operação está sendo requisitada.

Figura 24 - Exemplo de Código que mistura os métodos e será refatorado

```
private void addOrEditActivity(Integer codigo, String descricao, float custo) throws Exception{

    try {

        con = conectar();
        int index = 1;

        if (codigo == null){

            cs = con.prepareCall("{call SP_CAD_ATIVIDADE_ABC(?,?)}");

        }else{

            cs = con.prepareCall("{call SP_EDIT_ATIVIDADE_ABC(?,?,?)}");
            cs.setInt(index++, codigo);
        }

        cs.setString(index++, descricao);
        cs.setFloat(index++, custo);
        cs.executeUpdate();

        FBDAOFactory.disconnect();

    } catch (Exception e){

        e.printStackTrace();
        throw (new Exception(e));
    }
}
```

Fonte: Autoria própria

A Figura 25 apresenta uma alternativa de implementação do método ilustrado na Figura 24, foi aplicada a técnica *Separate Query from Modifier*, dividindo

o método em dois, cada um implementando uma única funcionalidade. Ainda neste método foi aplicada a técnica *Introduce Parameter Object*, já exemplificado anteriormente.

Figura 25 - Aplicação da Técnica Separate Query from Modifier

```
private void addActivity(ActivityObject activity) throws Exception{

    con = conectar();
    int index = 1;

    try {

        cs = con.prepareCall("{call SP_CAD_ATIVIDADE_ABC(?,?)}");

    } catch (Exception e) {

        e.printStackTrace();
        throw (new Exception(e));
    }
    cs.setString(index++, activity.getDescricao());
    cs.setFloat(index++, activity.getCusto());
    cs.executeUpdate();

    FBDAOFactory.disconnect();

}
private void editActivity(ActivityObject activity) throws Exception{

    con = conectar();
    int index = 1;

    try {

        cs = con.prepareCall("{call SP_EDIT_ATIVIDADE_ABC(?,?,?)}");

    } catch (Exception e) {

        e.printStackTrace();
        throw (new Exception(e));
    }
    cs.setInt(index++, activity.getCodigo());
    cs.setString(index++, activity.getDescricao());
    cs.setFloat(index++, activity.getCusto());
    cs.executeUpdate();

    FBDAOFactory.disconnect();
}
```

Fonte: Autoria própria

Além das técnicas catalogadas por Fowler (1999), outras técnicas voltadas para as boas práticas de desenvolvimento foram aplicadas no código sempre que

necessário, o sítio *javapractices.com* foi utilizado como referência nessa etapa. As classes da camada apresentavam comentários em excesso espalhados pelo código conforme apresentado na Figura 26, em muitos casos se tratavam de códigos de testes que foram comentados e por fim acabaram permanecendo desnecessariamente, gerando linhas de código e "poluindo" visualmente os métodos.

Figura 26 - Excesso de comentários espalhados pelo código

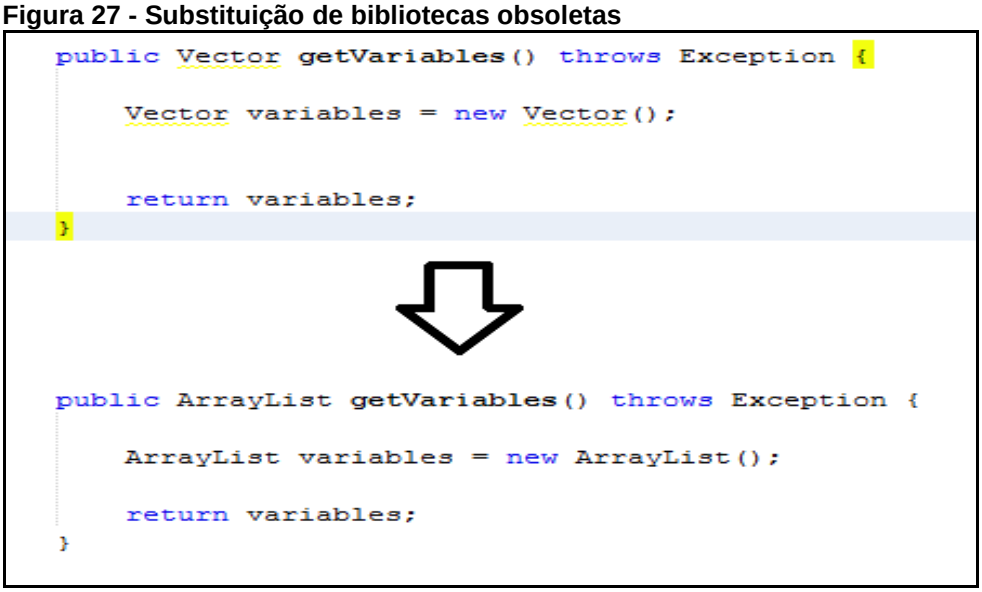
```
        /*
         * JOptionPane.showMessageDialog(null, "Cadastrado com Sucesso!",
         * "Cadastro", JOptionPane.INFORMATION_MESSAGE);
         */

    } catch (Exception e) {

        /*
         * JOptionPane.showMessageDialog(null, e.getMessage());
         * JOptionPane.showMessageDialog(null,
         * "Ocorreu um erro ao tentar efetuar o cadastro!", "Cadastro",
         * JOptionPane.ERROR_MESSAGE);
         */
        e.printStackTrace();
        throw (new Exception(e));
    }
```

Fonte: Autoria própria

Durante a análise do código foram encontradas várias bibliotecas e funções obsoletas na camada de persistência do framework, isso é inevitável com o passar dos anos.

As boas práticas de programação sugerem sempre atualizar as bibliotecas e funções que entram em desuso, buscando evitar problemas futuro com incompatibilidades na execução do software. A Figura 27 apresenta um exemplo aplicado no FrameMK.

Figura 27 - Substituição de bibliotecas obsoletas



Fonte: Autoria própria

A Tabela 1 apresenta os dados extraídos durante a refatoração, os dados são detalhados na sequência.

Tabela 1 – Apresentação dos totais referentes às linhas de códigos e métodos das classes

Camada de Persistência				
	Linhas de Código		Métodos	
Classes	Antes da Refatoração	Após a Refatoração	Antes da Refatoração	Após a refatoração
AbcActivityDAOFirebird	310	331	11	21
AbcAttributeDAOFirebird	303	348	10	19
AbcLogValueDAOFirebird	53	50	02	03
AbcProductDAOFirebird	288	320	08	15
AbcProductionLineDAOFirebird	216	260	07	16
FBDAOFactory	162	156	15	15
FullCostItemDAOFirebird	209	214	06	13
FullCostAttributeDAOFirebird	421	374	10	17
FullCostLogValueDAOFirebird	56	51	01	01
ProductDAOFirebird	216	260	07	16
SebraeAttributeDAOFirebird	443	401	11	18
SebraeLogValueDAOFirebird	52	47	01	01

Fonte: Autoria própria

A refatoração da camada de persistência do framework obteve como resultado a aplicação de técnicas nas 12 classes da camada, totalizando a aplicação de 6 técnicas diferentes de 5 categorias distintas, todas do catálogo proposto por Fowler (1999).

A Tabela 1 complementou um dos passos adicionados ao método de refatoração adaptado, servindo para armazenar os dados extraídos antes e depois da refatoração de cada classe.

Ao comparar os dados, percebe-se que todas as classes sofreram alterações. Analisando os números coletados se obtém os seguintes resultados: em relação ao número de códigos, 50% das classes apresentaram aumento de linhas e 50% reduziram a quantidade de linhas; em relação à quantidade de métodos os valores apresentam aumento de 75% e 25% das classes mantiveram a quantidade de métodos.

Os resultados são justificados pelos seguintes fatos: as técnicas *Encapsulate Field, Introduce Foreign Method, Separate Query from Modifier* tem como característica inserir novos métodos, gerando aumento de métodos e linhas de códigos na classe. Já as técnicas *Introduce Parameter Object, Extract Variable* permitiu a remoção de comentários desnecessários que caracterizam por remover listas extensas de atributos, reuso de variáveis e redução de linhas desnecessárias, respectivamente, gerando redução de linhas de códigos da classe.

4.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou inicialmente a adaptação realizada no método de Barros (2015), para que o mesmo atendesse as necessidades apresentadas como objetivos deste trabalho. Foram realizadas as refatorações seguindo o fluxo gerado pelo método adaptado.

Por fim, foram apresentados os resultados obtidos em forma de valores quantitativos coletados antes e após a aplicação do método de refatoração em cada classe da camada de persistência do FrameMK.

5 CONCLUSÃO

Este trabalho elaborou um estudo sobre a refatoração de software, buscando apresentar trabalhos distintos que ampliassem as possibilidades de se utilizar as técnicas de refatoração para promover melhorias em diferentes sistemas, visando buscar um método de refatoração que se adequasse a camada de persistência do FrameMK. O estudo inicial apresentou os conceitos gerais da refatoração de software, baseando-se em autores referências no assunto tal como (FOWLER, 1999).

Após a abordagem inicial do tema, buscou-se por métodos de refatoração que apresentassem diferentes cenários, em que os escolhidos foram: o método de Mens e Tourwé (2004) que tem como característica utilizar padrões de projetos no processo de refatoração; o método proposto por Rapeli (2006) em sua dissertação, que também utiliza padrões de projetos, porém é voltado para sistemas desenvolvidos na linguagem Java; o método de Barros (2015), que usa como base os dois métodos citados anteriormente, este método tem como diferencial ser voltado para frameworks de domínio, sendo este o principal motivo de ter sido escolhido como método a ser utilizado na refatoração do FrameMK.

Antes de iniciar o processo de refatoração do framework fez-se necessário estudar o FrameMK, alvo desse estudo. Foi apresentado a motivação do framework, seu processo de desenvolvimento, desde o início até os dias atuais, visto que se trata de um trabalho em andamento. Dedicou-se maior foco na estrutura do framework, especialmente na camada de persistência, por ser a utilizada na aplicação do método definido neste trabalho.

No processo de refatoração foi utilizado o método de Barros (2015), adaptando algumas etapas para que fosse possível utilizar as etapas já definidas, porém mantendo-o flexível para adicionar e/ou remover etapas sempre que necessário.

A adaptação do método fez-se necessária pelos seguintes aspectos: o método proposto tem como finalidade a refatoração de um software completo, neste trabalho a refatoração é proposta em uma única camada; Barros (2015) trata de características como metapadrões inversão de controle, que não se aplicam a abordagem deste trabalho; utiliza ferramentas para detecção de *bad smells.* Este

trabalho optou por estudar profundamente as técnicas de refatoração catalogadas por Fowler e Kerievsky e agi manualmente na camada, visto que por ser uma única camada o processo se tornou viável, encontrando pontos de refatoração que as ferramentas deixariam passar.

Os dados extraídos das classes durante o processo de refatoração possibilitaram uma análise do impacto da aplicação das técnicas em cada classe, chegando a conclusão de que a aplicação das técnicas gerou aumento de métodos, isso ocorreu devido as categorias de refatorações utilizadas durante o processo.

A refatoração da camada de persistência se mostrou satisfatória, tendo em vista o desenvolvimento do framework segue boas práticas de programação e aplica padrões de projetos, gerando um código já padronizado com poucos *bad smells*. Ainda assim, foram implementadas 6 (seis) técnicas diferentes na camada de persistência do framework, utilizado 5 (cinco) categorias distintas. Pelo fato do framework ter sido desenvolvido utilizando padrões de projetos, não foi possível implementar técnicas catalogadas por Kerievsky, porque todos os casos possíveis já haviam sido implementadas durante o desenvolvimento do mesmo.

5.1 TRABALHOS FUTUROS

Novos trabalhos podem ser realizados a partir desta pesquisa tais como:

- Refatorar as demais camadas do FrameMK utilizando métodos estudados neste trabalho.
- Criar uma ferramenta baseada nos catálogos de técnicas de refatoração apresentados neste trabalho.
- Estudar métricas voltadas para frameworks de domínio.
- Desenvolver uma ferramenta de testes automatizadas voltadas para frameworks de domínio.

REFERÊNCIAS

ABES (Associação Brasileira das Empresas de Software). **Mercado Brasileiro de Software:** Panorama e Tendências Estudo 2017 – Dados 2016. Disponível em: < http://www.abessoftware.com.br/dados-do-setor/estudo-2017--dados-2016>. Acesso em: 08 mai. 2018.

BARROS, V. P. A. **Um Método para Refatoração de Software Baseado em Frameworks de Domínio**. 2015. 96 f. Trabalho de Conclusão de Curso Superior de Bacharelado em Ciência da Computação – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

BOURGE, J.; BROWN, D. Rationale Support for Maintenance of Large Scale Systems. **Journal of Software Maintenance and Evolution: Research and Practice,** v.2, n. 1, 2002.

CARVALHO SOBRAL, A. J. F. **Frameworks Verticais vs Horizontais:** Gestão de Recursos Humanos com o OFBiz. 2015. 113 f. Dissertação (Mestrado) – Programa de Pós Graduação em Engenharia Informática, Instituto Superior de Engenharia do Porto, 2015.

FAYAD, M.; SCHMIDT, D.; JOHNSON, R. E. **Building Application Frameworks:** Object-Oriented Foundations of Framework Design. Nova Jersey: Wiley, 1999. 688 p.

FOWLER, M. **Refactoring**. Disponível em: <http://www.refactoring.com/>. Acesso em: 23 abr. 2015.

FOWLER, M. **Refactoring**: Improving the Design of Existing Code. Boston: Addison-wesley Professional, 1999. 464 p.

GAMMA, E.R.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software.** 1 ed, Estados Unidos: Addison-Wesley, 1994.

GRUPO de Pesquisa em Sistemas de Informação: GPSI. Disponível em: <http://gpes.utfpr.edu.br/gpes/>. Acesso em: 25 maio. 2015.

Java SE Development Kit 8. Disponível em:
< http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>. Acesso em: 03 jun. 2018.

R.E. Johnson;B. Foote. Designing Reusable Classes. **Journal of Object-Oriented Programming**, 1988.

KERIEVSKY, J. **Refatoração para padrões**. Porto Alegre: Bookman, 2008. 400p.

MAZER JUNIOR, A. **Métodos de Formação de Preço de Venda em Sistemas ERP por Intermédio de Arquitetura Orientada à Serviços do Framework FrameMK.** 2013. 115 f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2013.

MENDES, R. R. R. **Linha de Produto de Software: Um Estudo de Caso para o Desenvolvimento de um Sistema de FAQ**. 2015. 130 f. Trabalho de Conclusão de Curso Superior de Bacharelado em Ciência da Computação – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

MENS, T. TOURWÉ, Tom. A Survey of Software Refactoring. **IEEE Transactions on Software Engineering**, v. 30, n. 2, 2004.

PRESSMAN, R. **Software Engineering**, McGraw-Hill, 2005

Hibernate ORM. Disponível em: < http://hibernate.org/>. Acesso em: 04 jun. 2018.

OPDYKE, W. F. **Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks**. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

RAPELI, L. R. C. **Refatoração de sistema Java utilizando padrões de projeto:** um estudo de caso. 2006. 127 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de São Carlos, São Carlos, 2006.

RIBAS, J. H. **Desenvolvimento de classes de teste para a camada de persistência do framework de formação de preço de venda (FrameMK) usando JUnit**. 2014. 75 f. Trabalho de Conclusão de Curso Superior de Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2014.

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do SUL.

WILLEMAN, D. P; IBARRA, G. B. **Framework Java de Apoio ao Desenvolvimento de Aplicações Web com Banco de Dados, utilizando Struts, Tiles e Hibernate**. 2007. 141 f. Trabalho de Conclusão de Curso Superior de Bacharelado em Ciência da Computação – Universidade Federal de Santa Catarina. Florianópolis, 2007.

ANEXO A - TÉCNICAS DE REFATORAÇÃO CATALOGADAS POR FOWLER

As técnicas apresentadas abaixo são classificadas e apresentadas por Martin Fowler, as mesmas podem ser encontradas em seu livro de título *Refactoring*: *Improving the Design of Existing Code* ou por meio do seu sitio https://refactoring.com/catalog/.

ASSOCIATIONS
Change Bidirectional Association to Unidirectional, Change Reference to Value, Change Unidirectional Association to Bidirectional, Change Value to Reference, Duplicate Observed Data, Extract Class, Inline Class, Replace Delegation With Hierarchy, Replace Delegation with Inheritance, Replace Inheritance with Delegation and Replace Method with Method Object.
ENCAPSULATION
Encapsulate Collection, Encapsulate Downcast, Encapsulate Field, Hide Delegate, Hide Method, Preserve Whole Object, Remove Setting Method and Self Encapsulate Field.
GENERIC TYPES
Replace Array with Object, Replace Data Value with Object, Replace Hash with Object, Replace Magic Number with Symbolic Constanta and Replace Record with Data Class.
INTERFACES
Extract Interface, Replace Constructor with Factory Method.
CLASS EXTRACTION
Extract Class, Extract Interface, Extract Module, Extract Subclass, Extract Superclass and Introduce Parameter Object.
GOF Patterns
Form Template Method and Replace Type Code with State/Strategy.
LOCAL VARIABLES
Extract Variable, Inline Temp, Remove Assignments to Parameters, Replace Method with Method Object, Replace Temp with Chain, Replace Temp with Query and Split Temporary Variable.
VENDOR LIBRARIES
Introduce Foreign Method, Introduce Gateway and Introduce Local Extension.
ERRORS
Replace Error Code with Exception and Replace Exception with Test.
TYPE CODES
Replace Type Code with Class, Replace Type Code with Module Extension, Replace Type Code with Polymorphism, Replace Type Code with State/Strategy and Replace Type Code with Subclasses.
METHOD CALLS
Add Parameter, Encapsulate Downcast, Hide Method, Introduce Expression Builder, Introduce Gateway, Introduced Named Parameter, Introduce Parameter Object, Parameterize Method,

Preserve Whole Object, Remove Control Flag, Remove Named Parameter, Remove Parameter, Remove Setting Method, Rename Method, Replace Constructor with Factory Method, Replace Error Code with Exception, Replace Exception with Test, Replace Parameter with Explicit Methods, Replace Parameter with Method and Separate Query from Modifier.
ORGANIZING DATA
Change Bidirectional Association to Unidirectional, Change Reference to Value, Change Unidirectional Association to Bidirectional, Change Value to Reference, Duplicate Observed Data, Eagerly Initialized Attribute, Encapsulate Collection, Encapsulate Field, Lazily Initialized Attribute, Replace Array with Object, Replace Data Value with Object, Replace Magic Number with Symbolic Constant, Replace Record with Data Class, Replace Subclass with Fields, Replace Type Code with Class, Replace Type Code with State/Strategy, Replace Type Code with Subclasses and Self Encapsulate Field.
INHERITANCE
Collapse Hierarchy, Encapsulate Downcast, Extract Interface, Extract Module, Extract Subclass, Extract Superclass, Form Template Method, Introduce Null Object, Pull Up Constructor Body, Pull Up Field, Pull Up Method, Push Down Field, Push Down Method, Replace Abstract Superclass with Module, Replace Conditional with Polymorphism, Replace Delegation With Hierarchy, Replace Delegation with Inheritance, Replace Inheritance with Delegation, Replace Subclass with Fields and Replace Type Code with Subclasses.
CONDITIONALS
Consolidate Conditional Expression, Consolidate Duplicate Conditional Fragments, Decompose Conditional, Introduce Assertion, Introduce Null Object, Recompose Conditional, Remove Control Flag, Replace Conditional with Polymorphism, Replace Exception with Test and Replace Nested Conditional with Guard clauses.
MOVING FEATURES
Extract Class, Extract Module, Hide Delegate, Inline Class, Inline Module, Introduce Foreign Method, Introduce Local Extension, Move Field, Move Method, Remove Middle Man.
COMPOSING METHODS
Consolidate Conditional Expression, Decompose Conditional, Extract Method, Extract Surrounding Method, Extract Variable, Form Template Method, Inline Method, Inline Temp, Move Eval from Runtime to Parse Time, Remove Assignments to Parameters, Replace Loop with Collection Closure Method, Replace Method with Method Object, Replace temp with Query, Split Temporary Variable and Substitute Algorithm.
DEFINING METHODS
Dynamic Method Definition, Introduce Class Annotation, Isolate Dynamic Receptor, Remove Unused Default Parameter, Replace Dynamic Receptor with Dynamic Method Definition and Replace Method with Method Object.