UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUAN BUKOWITZ BELUZZO

ABORDAGEM PARA AVALIAR E DETECTAR PONTOS DE INSERÇÃO E APLICAÇÃO DE PADRÕES DE PROJETO EM CÓDIGO-FONTE

DISSERTAÇÃO

PONTA GROSSA

2018

LUAN BUKOWITZ BELUZZO

ABORDAGEM PARA AVALIAR E DETECTAR PONTOS DE INSERÇÃO E APLICAÇÃO DE PADRÕES DE PROJETO EM CÓDIGO-FONTE

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Ciência da Computação do programa de Pós-Graduação em Ciência da Computação da Universidade Tecnológica Federal do Paraná - Campus Ponta Grossa. Área de Concentração: Sistemas e Métodos de Computação**.**

Orientadora: Prof. Dra. Simone Nasser Matos

PONTA GROSSA

2018

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Câmpus Ponta Grossa
n.57/18

B453 Beluzzo, Luan Bukowitz

Abordagem para avaliar e detectar pontos de inserção e aplicação de padrões de projeto em código-fonte. / Luan Bukowitz Beluzzo. 2018.
100 f.; il. 30 cm

Orientadora: Prof. Dra. Simone Nasser Matos

Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação. Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2018.

1. Software - Refatoração. 2. Padrões de software. 3. Programação orientada a objetos (Computação). 4. Projeto auxiliado por computador. I. Matos, Simone Nasser. II. Universidade Tecnológica Federal do Paraná. III. Título.

CDD 004

Elson Heraldo Ribeiro Junior. CRB-9/1413. 20/11/2018.

	Ministério da Educação **UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ** Diretoria de Pesquisa e Pós-Graduação Programa de Pós-Graduação em Ciência da Computação	

FOLHA DE APROVAÇÃO

Título de Dissertação Nº 5/2018

ABORDAGEM PARA AVALIAR E DETECTAR PONTOS DE INSERÇÃO E APLICAÇÃO DE PADRÕES DE PROJETO EM CÓDIGO-FONTE

Por

Luan Bukowitz Beluzzo

Esta dissertação foi apresentada às **14 horas** de **23 outubro 2018**, na sala da **DIRPPG**, como requisito parcial para a obtenção do título de MESTRE EM CIÊNCIA DA COMPUTAÇÃO, Programa de Pós-Graduação em Ciência da Computação. O candidato foi arguido pela Banca Examinadora, composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Profª. Drª. Simone do Rocio Senger de Souza (USP)

Profª. Drª. Helyane Bronoski Borges (UTFPR)

Profª. Drª. Simone Nasser Matos (UTFPR)
Orientador e presidente da banca

Visto da Coordenadora:

Profª. Drª. Sheila Morais de Almeida
Coordenadora do PPGCC
UTFPR - Câmpus Ponta Grossa

- A Folha de Aprovação assinada encontra-se na Coordenação do Programa -

RESUMO

BELUZZO, L. B**. Abordagem para avaliar e detectar pontos de inserção e aplicação de padrões de projeto em código-fonte.** 2018. 100 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2018.

A refatoração tem a finalidade de melhorar o código-fonte em relação aos requisitos de qualidade como: manutenibilidade, flexibilidade, legibilidade, entre outros. Dentre os trabalhos encontrados na literatura voltados a refatoração de software baseada em padrões de projetos foram analisados os que abordam métodos e ferramentas. Constatou-se que estes trabalhos aplicam somente um processo de refatoração construído pelos próprios autores, as ferramentas possuem pouca ou nenhuma interação com o usuário e não existe a preocupação de se avaliar antecipadamente os benefícios de se aplicar um determinado padrão no código-fonte. Por isto, este trabalho criou uma abordagem para detecção de pontos de inserção e aplicação de padrões de projeto que reúne em um mesmo ambiente os métodos da literatura, além de fornecer informações sobre os benefícios de se usar um determinado padrão antes de sua aplicação efetiva. Estas informações são obtidas por meio de métricas de software relacionadas aos atributos de qualidade como a manutenibilidade, confiabilidade e reusabilidade. A abordagem proposta foi inicialmente avaliada aplicando cenários de testes providos pelos métodos da literatura e posteriormente foram usados cinquenta projetos *open-source* encontrados na web para testes. Como resultado, verificou-se que a abordagem é capaz de retornar candidatos a refatoração de mais de um método da refatoração, além de apresentar ao usuário uma avaliação do candidato a refatoração baseada em métricas e atributos de qualidade.

Palavras-chave: Refatoração. Padrões de Projeto. Métricas.

ABSTRACT

BELUZZO, Luan Bukowitz**. An approach for evaluating, detecting design patterns insertion spots and applying them in source.** 2018. 100 p. Dissertation (Master in Computer Science) - Federal University of Technology - Paraná, Ponta Grossa, 2018.

Refactoring processes are executed in order to improve a given source code in terms of quality aspects such as maintainability, flexibility, readability, and others. Several methods and tools were proposed in the literature to perform source code refactorings. It was noticed In the literature that the researches only apply their own refactoring process, the tools developed by them present just a few interactions with the user or no interaction at all and, there is no concern of assessing the benefits of applying a given design pattern in the source code. That is why this work has created an approach for detecting design patterns insertion spots and applying them. This approach puts some refactoring methods of the literature in the same environment, besides providing assessments of the real benefits of inserting a given pattern before actually applying it. The assessment of the source code is created based on software metrics related to quality attributes like maintainability, reliability, and reusability. The evaluation of the approach was initially performed by applying test scenarios described in literature methods, after that, fifty open-source projects were retrieved from the web in order to test the proposed approach. As a result, the approach was able to suggest refactoring candidates of different refactoring methods, it also presents the assessment of the refactoring candidate to the user, in terms of metrics and quality attributes.

Keywords: Refactoring. Design Patterns. Metrics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As refatorações visam criar uma melhor estrutura de um projeto sem alterar suas saídas. Fowler *et al*. (1999) mencionam que os pontos de refatoração podem aparecer ao longo do tempo de desenvolvimento de um sistema, independentemente de quão bem o código foi inicialmente projetado e arquitetado.

Um código-fonte de baixa qualidade exige um maior tempo de manutenibilidade e torna sua leitura mais difícil aos desenvolvedores responsáveis por sua alteração. Por isso, a adoção de refatorações podem tornar o código mais legível e de fácil manutenção a outros desenvolvedores (FOWLER *et al*. 1999; KERIEVSKY, 2008).

As refatorações podem ser realizadas usando técnicas (FOWLER *et al*., 1999) ou baseada em padrões de projeto (KERIEVSKY; 2008). A refatoração baseada em padrões de projeto representam estruturas que permitem melhorar o sistema em termos de atributos de qualidade tais como reusabilidade, flexibilidade, manutenibilidade (Li *et al*., 2007; KERIEVSKY, 2008). Padrões de projeto encapsulam a descrição abstrata e estruturada de uma solução satisfatória para um problema que ocorre repetidamente dentro de um contexto, dado um conjunto de forças ou restrições que atuam sobre ele (GAMMA *et al*., 1994). Por isso, os padrões visam melhorar e formalizar a documentação de experiência do conhecimento, registrando de maneira estruturada as soluções que pessoas mais experientes encontraram ao se defrontar com problemas repetitivos.

Ge e Murphy-Hill (2008) destacam que independentemente da qualificação do profissional que realiza as refatorações, ele pode aplicá-las incorretamente. A utilização de uma ferramenta de refatoração pode auxiliar o desenvolvedor de software, apresentando mensagens de auxílio a refatoração, de forma que o processo passa a ser executado de forma mais rápida.

Ducasse, Rieger e Golomigni (1999), Murphy-Hill e Black (2008) e Ge e Murphy-Hill (2008) criaram ferramentas automatizadas que aplicam técnicas de refatoração. A utilização de ferramentas de refatoração permite diminuir a probabilidade de erro humano em processos de refatoração manuais (GE; DUBOSE; MURPHY-HILL, 2012). Um outro ponto a ser destacado é que estas refatorações não se limitam a alterações pontuais de pequena granularidade, mas já propõem grandes

mudanças na estrutura do código-fonte com a inserção de padrões de projeto (FOWLER *et al*., 1999).

Para se ter uma abrangência maior dos trabalhos já desenvolvidos sobre o assunto de detecção de pontos de inserção de padrões de projeto foi proposto neste trabalho a realização de um mapeamento sistemático abrangendo os anos de 1997 a 2017. O método de mapeamento adotado usou como base os métodos de Kitchenham e Charters (2007) e Pagani, Kovaleski e Resende (2015).

A partir deste mapeamento, trabalhos foram analisados sobre o assunto, sendo que alguns se baseiam: em precursores como ponto inicial a refatoração (CINNÉIDE; NIXON, 1999); na obtenção de pontos de inserção de padrões de projeto a construção de fatos e regras Prolog (JEON; LEE; BAE, 2002); na inserção de determinados padrões como: *Strategy* (CHRISTOPOULOU et al., 2012), *Factory Method* e *Strategy* (WEI *et al*., 2014), NULL *Object* (GAITANI *et al*., 2015), *Template Method* (ZAFEIRIS *et al*., 2017), entre outros. Constatou-se que os trabalhos não abordam interação com usuários durante o processo de refatoração; incorporação de diversos métodos de refatoração da literatura em um único ambiente; uma estrutura extensível a aplicação de mais métodos; e a medição dos efeitos refatoração antes de aplicá-la efetivamente.

Esse trabalho criou uma abordagem que contempla em um ambiente múltiplos métodos de detecção e inserção de padrões de projeto que permite realizar uma avaliação prévia (antes da aplicação do padrão) a refatoração do projeto; uma avaliação em relação a atributos de qualidade como manutenibilidade, confiabilidade e reusabilidade, obtidos por meio de métricas de software. O processo de aceitação ou não da aplicação do padrão fica sob responsabilidade do usuário, pois a abordagem permite sua participação no processo de refatoração.

A abordagem foi subdividida em 4 (quatro) módulos: *Client App, Intermediary Service, Detection Methods Service e Metrics Service*. Cada um dos módulos apresenta funções definidas de interação com o usuário, promoção da interação entre os módulos e detecção de pontos de inserção.

1.1 JUSTIFICATIVA

O propósito de refatorar é prover uma boa estrutura interna para o produto de software sem mudar seu comportamento. Algumas das possíveis mudanças são fragmentações de métodos, reestruturação de hierarquias de classes, redefinição de nomes de métodos e classes, etc. Adicionalmente podem ser inseridos novos casos de testes ou funcionalidades porque foram omitidos ou esquecidos (FOWLER *et al*., 1999).

Um processo de refatoração pode ser necessário para cenários independentemente de como um produto de software está sendo arquitetado ou modelado. Este processo permite eliminar os *bad smells* e melhorar sua inteligibilidade para facilitar as modificações futuras (FOWLER *et al*., 1999).

Ferramentas de refatoração são utilizadas no desenvolvimento de software, dentre estas, muitas são pouco apreciadas por engenheiros de software pois acabam alterando seu fluxo de trabalho e apresentam refatorações ditas como incorretas por alguns desenvolvedores. Estas ferramentas têm como objetivos principais em sua maioria, pequenas refatorações já definidas na literatura e não a implementação de padrões que provêm uma maior manutenibilidade e flexibilidade no código.

Portanto, o uso de ferramentas que auxiliem no processo de refatoração com maior participação do usuário, em que se tem integrado em um único ambiente métodos já consolidados para realização de refatoração baseada em padrões e avaliação da qualidade, se torna algo relevante na produção de produtos de software de maior qualidade.

1.2 OBJETIVOS

Considerando a importância do tema sobre refatoração de software e a falta de: i) um ambiente centralizado de métodos de refatorações baseados em padrões de projeto, ii) um processo de aplicação das refatorações que avalie antecipadamente sua contribuição para o código-fonte e iii) maior participação do usuário no processo de refatoração, foi estabelecido o objetivo geral desta pesquisa.

O objetivo geral é criar uma abordagem para detecção de pontos de inserção e aplicação de padrões de projeto em código-fonte orientado a objetos usando

interação com o usuário, métricas de software e métodos da literatura capaz de identificar os possíveis padrões que podem ser usados.

Para o cumprimento do objetivo geral, os seguintes objetivos específicos foram definidos: i) realizar um mapeamento sistemático sobre o assunto de métodos ou ferramentas de detecção de pontos de inserção e inserção de padrões de projeto; ii) criar um protótipo baseado na abordagem proposta, e por fim, iii) avaliar a abordagem proposta a partir de cenários de testes.

1.3 METODOLOGIA DE PESQUISA

Esta proposta teve como objetivo criar uma abordagem de refatoração de software com a aplicação de padrões de projeto e avaliação do código-fonte. Como nenhum mapeamento sistemático ou revisão haviam sido elaborados até o momento da pesquisa sobre o tema, foi realizado um mapeamento sistemático sobre métodos e ferramentas voltados a detecção de pontos de inserção de padrões de projeto em códigos-fonte. O mapeamento permitiu identificar assuntos que ainda não haviam sido propostos e testados em relação ao tema.

A abordagem proposta foi implementada para que fosse possível a realização de sua avaliação. Foram realizadas duas avaliações: a primeira foi executada por meio da obtenção de candidatos a refatoração em um cenário simples de aplicação usando como referências os métodos da literatura voltados a detecção e inserção de padrões de projeto e a segunda avaliação foi executada aplicando a refatoração em múltiplos projetos *open-source* disponíveis na web.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho está organizado em seis capítulos. O Capítulo 2 traz um apanhado geral sobre o tema de refatoração e apresenta refatorações voltadas a padrões de projeto, abordagens e tipos de refatoração.

O Capítulo 3 detalha o estado da arte que foi realizado por um mapeamento sistemático sobre métodos ou ferramentas de refatoração orientadas a objeto capazes de detectar pontos de inserção de padrões de projeto em código-fonte.

O Capítulo 4 descreve a abordagem proposta relatando seu funcionamento interno. Além disso, apresenta os pontos de extensão da abordagem em termos de métodos de refatoração e avaliações de atributos de qualidade baseadas em métricas de software.

O Capítulo 5 desceve a ferramenta que implementa as funcionalidades da abordagem proposta e as tecnologias utilizadas para sua criação. Aborda também os cenários de testes usados para a avaliação da abordagem proposta e a análise dos resultados.

Por fim, o Capítulo 6 apresenta as considerações finais deste trabalho, bem como os trabalhos futuros que podem dar continuidade a esta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

O processo de refatoração é importante porque garante uma qualidade maior no software que é produzido. Fowler *et al*. (1999) mencionam que os pontos de refatoração podem aparecer ao longo do tempo de desenvolvimento de um sistema, independentemente de quão bem o código foi inicialmente projetado e arquitetado.

O processo de refatoração permite: i) modularizar classes com excessos de responsabilidades; ii) encontrar código duplicado, com linhas similares ou exatamente iguais; iii) falicitar a inserção de novos requisitos ao sistema que serão modelados e implementados; iv) encontrar nomes de classes e métodos que não são significativos; entre outros (MARTIN, 2009).

A refatoração pode ser conduzida por meio de técnicas ou baseada em padrões de projeto. Em relação as técnicas, Fowler *et al*. (1999) criou um catálogo separando-as em grupos: *Composing Methods*, *Moving Features Between Objects*, *Organizing Data*, *Simplifying Conditional Expressions*, *Making Method Calls Simpler*, *Dealing with Generalization* e *Big Refactorings*. A refatoração baseada em padrões de projeto é fundamentada nos padrões do Gamma *et al*. (1994) e existem métodos na literatura tais como os de Mel Ó Cinnèide e Nixon (1999), Kerievsky (2008) e de Ouni *et al*. (2017), que estabelecem um processo de detecção de pontos de inserção e inserção de padrões. Pesquisas mostram que as refatorações baseadas em de padrões de projeto aumentam a reusabilidade, manutenibilidade e legibilidade, por isto foi foco deste trabalho.

A Seção 2.1 descreve a importância de refatorações tanto em códigos legados quanto aqueles ainda em desenvolvimento. Apresenta como desenvolvedores tem se utilizado de refatorações semi-automatizadas e descreve algumas abordagens para extração de dados e manipulações do código-fonte comuns da literatura. A abordagem proposta usa métricas de software para avaliar atributos de qualidade, tal como manutenibilidade, antes que o padrão de projeto seja efetivamente aplicado ao projeto. Os conceitos sobre métricas estão descritos na Seção 2.2. As considerações finais deste capítulo são relatadas na Seção 2.3.

2.1 REFATORAÇÃO DE SOFTWARE

O fato de simplificar o processo de desenvolvimento de determinado código-fonte já foi motivo para que as refatorações fossem realizadas, mas, outros cenários são apresentados tais como: i) realizar a inserção de um novo código prevendo sua evolução de forma facilitada; ii) melhorar a legibilidade do código; iii) manter um código antigo de forma que não diminua o rendimento dos programadores envolvidos. Isto ocorre, por exemplo, em cenários onde o programador é encarregado de fazer alterações em códigos legados (KERIEVSKY, 2008).

Com base nas técnicas previamente apresentadas por Fowler *et al*. (1999), é observado que as refatorações, em diversas situações, compõem um processo que deve tomar atenção de seu executor, para que, uma vez que aplicada, não resulte na inserção de mais erros no código refatorado. Em vários cenários os processos de refatoração costumam a ser extensos, o que significa que estes podem ser fragmentados em refatorações menores, isto faz com que a complexidade de uma única refatoração seja elevada.

O processo de refatoração pode ser realizado de forma manual ou automatizado. Refatorações manuais são frequentemente realizadas, porém, ao invés de auxiliarem no processo de desenvolvimento de software podem introduzir mais erros ao software. Isso ocorre porque desenvolvedores não estão tecnicamente preparados o suficiente para realizarem as refatorações (GE; DUBOSE; MURPHY-HILL, 2012).

Com o objetivo de aprimorar o processo de refatoração, algumas ferramentas foram criadas para implementação de diferentes técnicas. Ducasse, Rieger e Golomingi (1999) propuseram uma ferramenta para resolver problemas de duplicação de código em contextos orientados a objeto.

Murphy-Hill e Black (2008) apresentam três ferramentas que têm por objetivo aplicar técnicas de refatoração específicas. A primeira é baseada em seleções feitas pelo programador; a segunda representa a classe como se fosse um retângulo e todos seus métodos e declarações internas se apresentam como retângulos internos aninhados; a terceira é baseada em anotações, ela desenha retângulos ao redor do código, flechas direcionais e também colore algumas áreas para dar contraste nas partes do código que devem ser alteradas, diferenciando-as das demais.

Um passo importante em refatorações automatizadas é que algumas técnicas encontradas na literatura são dependentes entre si e por isto é recomendado checar se as refatorações efetuadas pela técnica de refatoração *A* terão efeitos na técnica *B* (ROBERTS, 1999). Murphy-Hill e Black (2008) destacam o uso da técnica *Extract Method* no seu trabalho, devido a seu impacto em outras técnicas, sendo considerado um tipo de técnica base para outras.

Em pesquisas realizadas por Ge e Murphy-Hill (2014), utilizando de oito desenvolvedores de software de diversos níveis profissionais, foi observado que a utilização de uma ferramenta (*GostFactor*) auxilia o processo de refatoração e fez com que os erros no processo reduzissem em 23,3%.

A conclusão dos estudos mostra que ferramentas de refatoração podem efetivamente reduzir erros humanos em processos de refatoração e sugerem que, com a devida adaptação a sua ferramenta, pode ocasionar também, em uma maior agilidade no processo.

2.1.1 Processo de Refatoração Baseado em Padrões de Projeto

Padrões de projeto podem ser descartados na concepção inicial do projeto por não serem conhecidos ou valorizados pelos engenheiros de software. Apesar disso, alguns padrões se fazem necessários devido a evolução natural do software através de novos requisitos. Eles são úteis quando anexados no processo de refatoração, adicionando mais flexibilidade ao código dado (CINNÉIDE, 2000).

Alguns exemplos de padrões de projeto são: *abstract factory*, *factory method*, *builder*, *adapter*, *flyweight*, *proxy*, *observer*, *visitor* e *template method* (GAMMA *et al*., 1994). Diversos autores propuseram a refatoração aplicando padrões de projeto como Fowler *et al*. (1999) e Kerievsky (2008), em que seus trabalhos são base para outros pesquisadores. Zafeiris *et al*. (2017) propõe a aplicação do padrão *Template*, verificando situações em que uma invocação super (em uma hierarquia de classes) é feita em determinado método de uma classe. Após a refatoração é criado um método *template* na classe pai e são feitas chamadas a métodos *hook* no método da classe filha que antes continha a chamada super.

Visando a aplicação dos padrões *Factory* e *Strategy*, Wei *et al*. (2014) apresenta uma abordagem que faz avaliações de expressões condicionais. O processo de verificação de candidatos a refatoração busca métodos que contém

expressões condicionais complexas, ou seja, condicionais que verificam o tipo de um parâmetro ou atributo para que nos vários ramos de expressões condicionais sejam apresentados comportamentos diferenciados. Candidatos a aplicação do padrão *Factory* são criados quando o parâmetro do método é utilizado para gerar novas entidades de retorno. Os candidatos a aplicação do *Strategy* são criados quando não existe uma classe de retorno no método o qual a expressão condicional está inserida.

Cinnéide e Nixon (1999) apresentam um método de refatoração baseado em *minipatterns* e *minitransformations*, onde *minipatterns* são como “pequenos passos” em direção a determinado padrão e *minitransformations* são as refatorações necessárias para que um determinado “pequeno passo” seja alcançado. Por exemplo, dado um *minipattern*: “a classe pai deve ser abstrata”, a *minitransformation*: “refatorar a determinada classe pai para abstrata” é aplicada. Este tipo de *minipattern* e *minitransformation* apresenta uma situação (genérica) aplicável a diversos tipos de padrões de projetos, estes podem ser aplicados em sequência de acordo com viabilidade de aplicação para o padrão em questão.

Os métodos de refatoração como Cinnéide e Nixon (1999), Wei *et al*. (2014), Zafeiris *et al*. (2017), entre outros, utilizam de abordagens de extração e tipos de refatoração para a aplicação dos padrões de projeto. Estes assuntos são abordados nas próximas seções.

2.1.1.1Abordagens de extração de código-fonte

A refatoração baseada em padrões de projeto pode utilizar várias abordagens para extração de código-fonte tais como: JavaCC (JAVACC, 2018), SOOT (VALEE-RAI *et al*., 1999), Fatos Prolog (CLOCKSIN; MELISH, 2003), Compilador Eclipse IDE (ECLIPSE IDE, 2018) e Árvores de Sintaxe Abstrata (JONES, 2003).

JavaCC (JAVACC, 2018) é um analisador de código-fonte que permite a inclusão de código em uma classe preexistente, através de definições de funções customizadas. Permite também a extração de informações do código a partir de símbolos (*tokens*) definidos pelo desenvolvedor.

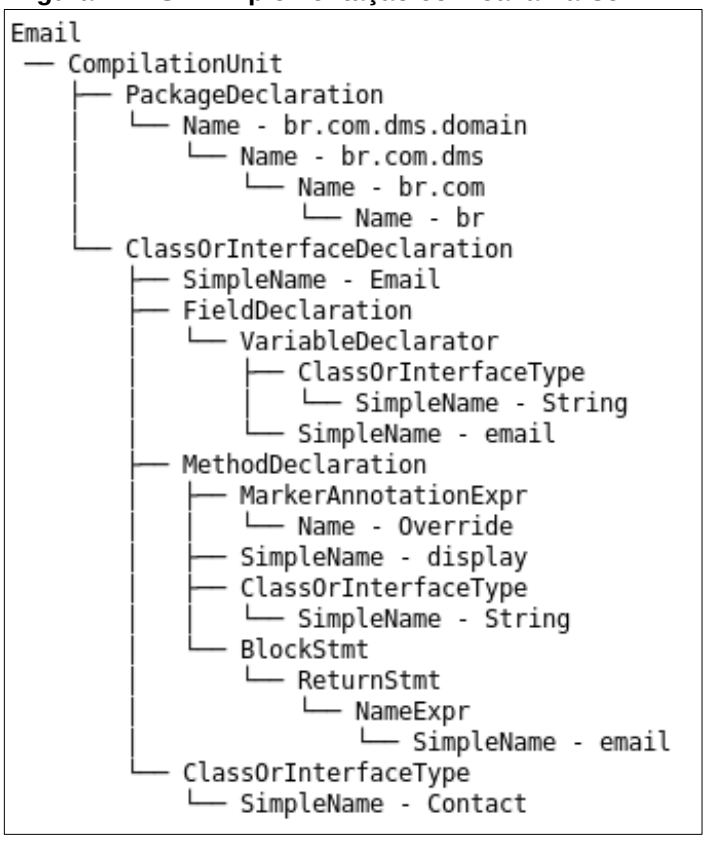
SOOT (VALLE-RAI *et al*., 2010) é um framework de otimização que pode ser usado para obter informações sobre classes, métodos, atributos e até mesmo, relacionamentos entre classes. Fatos Prolog (CLOCKSIN; MELISH, 2003) são um

conjunto de dados que correspondem a informações do código-fonte que pode ser (similarmente a um banco de dados) usado para buscas com objetivo de responder questões de interesse do manipulador de dados.

Compilador Eclipse IDE provê tabelas não persistentes relacionadas as classes, métodos, campos e outras estruturas de um código-fonte. Estas tabelas são geradas quando o código-fonte é compilado (ECLIPSE IDE, 2018).

Árvores de Sintaxe Abstrata (AST) são apresentadas em detalhes porque dentre as abordagens de extração estudadas, são as mais utilizadas nos métodos de refatoração (JONES, 2003). Esta abordagem representa o código-fonte usando uma estrutura de dados em árvore na qual cada elemento de uma classe é um nó. Estas estruturas omitem outras partes do código-fonte como vírgulas, ponto e vígulas e outros. Uma classe que tenha dois atributos e um método por exemplo, terá seu tipo de classe como raiz da árvore e os seus atributos e método como nós filhos desta arvore, conforme apresenta a Figura 1.

Figura 1 - *AST* - Implementação com Java Parser



Fonte: Autoria Própria

A classe Email na Figura 1 ilustra um exemplo de como é a representação de uma classe por meio de Árvores de Sintaxe Abstrata (JONES, 2003). Com esta implementação é possível ver as características de uma AST e manipular sua estrutura interna. Cada nó da estutura de árvore tem uma classe pai comum, a saber, *Node*. Nota-se o vínculo de um nó pai com seus filhos como acontece com *ClassOrInterfaceDeclaration* que tem os nós filho: *SimpleName*, *FieldName*, *MethodDeclaration* e *ClassInterfaceType*.

SimpleName é o nó que contém o nome do nó pai, no caso como se trata da declaração da classe, o nome é *Email*. O nó *FieldDeclaration* mostra um campo da classe *E-mail*, a qual não está inicializada pois entre os nós da sua hierarquia tem-se somente o nome da variável (*SimpleName*) e o tipo que foi declarado (*ClassOrInterfaceType*). *MethodDeclaration* é representado como o único método da classe *Email*, a partir dele se vê mais detalhes de seus nós filho, além de apresentarem o nome do método (*SimpleName*) e o seu tipo de retorno (*ClassOrInterfaceType*). Também há o nó que envolve a declaração do corpo do método (*BlockStmt*) e o *ClassOrInterfaceType* referente a *ClassOrInterfaceDeclaration*, o qual é um nó específico para declaração do tipo da classe.

2.1.1.2 Tipos de Refatoração

Além das abordagens de extração apresentadas na seção anterior, outro aspecto encontrado em trabalhos da literatura é referente aos tipos de se realizar a refatoração, a saber, *minitransformations*, *reflective refactorings*, refatorações baseadas em papéis e as baseadas em *Intent Aspects.*

Minitransformations possibilitam que o código alvo da refatoração se aproxime gradativamente ao padrão desejado por meio de pequenas modificações (CINNÈIDE, 2001). A Figura 2 apresenta a *Abstraction Minitransformation*, que considera a possiblidade de um processo de refatoração criar uma nova interface e faz com que a Classe *c* a implemente.

Figura 2 - *Abstraction Minitransformation*

```
Abstraction(Class c, String newName){
        Interface inf = abstractClass(c, newName);
        addInterface(inf);
        addImplementsLink(c, inf);
}
```

Fonte: Cinnèide (2001)

Reflective Refactorings é um meio de alteração do código-fonte provido pelas *Java Development Tools* (ECLIPSE JDT, 2018), na qual classes, métodos podem ser extraídos ou obtidos em tempo de execução e alterados por meio de códigos java de refatoração (Figura 3).

Figura 3 - *Reflective Refactorings* - Criação de padrão Visitor

```
1  // member of RMethod class
2  void RClass makeVisitor(String visitorClassName)
3    throws RException
4  {
5    RPackage pkg = this.getPackage();
6    RClass vc = pkg.newClass(visitorClassName);
7    RField singleton = vc.addSingleton();
8
9    RMethodList methodList = this.getRelatives();
10   RParameter newPara =
11     methodList.addParameter(vc, singleton);
12
13   RMethod delegate = null;
14   for(RMethod m : methodList) {
15     if(!m.isMovable())
16       continue;
17     delegate = m.moveAndDelegate(newPara);
18     m.rename("visit");
19    }
20
21   RMethodList delegateList =
22     delegate.getRelatives();
23   delegateList.rename("accept");
24
25   return vc;
26 }
```

Fonte: Kim, Batory e Dig (2014)

Como é observado na Figura 3, o código-alvo da refatoração é facilmente alterado por meio da aplicação do padrão Visitor. Criam-se uma classe (linha 6), métodos da classe Visitor (linhas 9 a 18) e a alteração dos métodos a serem visitados (linhas 21 a 23).

As refatorações baseadas em papéis consideram que os padrões (candidatos a inserção) tem um conjunto de papéis que os definem (MENS, TOURWÉ, 2001). Estes papéis podem ser classes, métodos, relações entre classes, como apresentado na Figura 4.

Figura 4 - Refatorações baseadas em papéis - Definição de padrão

```
pattern(abstractFactory,
  [abstractFactory, concreteFactory, genericProduct,
  abstractProduct, concreteProduct, abstractRelation,
  concreteRelation, abstractFactoryMethod,
  concreteFactoryMethod]).
```

Fonte: Mens e Tourwé (2001)

Após isso, é possível solicitar a criação de um novo padrão a partir da especificação de seus membros e os papéis executados por cada um dos membros definidos. Na Figura 5 observa-se que cada definição dos membros do padrão é informado o nome e o papel desempenhado por aquele novo elemento, neste exemplo, a criação do padrão *Abstract Factory*, têm: 1 (uma) classe *abstract factory* (*Look*), 2 (duas) *factories* concretas (*MSLook* e *MacLook*) que são encarregadas de construir botões *MSLook* ou *MacButton* e janelas *MsWindow* e *MacWindow*.

Figura 5 - Refatorações baseadas em papéis - Criação de *Abstract Factory*

```
patternInstance(AF1, abstractFactory).
role(AF1, abstractFactory, Look).
role(AF1, concreteFactory, MSLook).
role(AF1, concreteFactory, MacLook).
role(AF1, genericProduct, Widget).
role(AF1, abstractProduct, Window).
role(AF1, abstractProduct, Button).
role(AF1, concreteProduct, [MSWindow, Window]).
role(AF1, concreteProduct, [MSButton, Button]).
role(AF1, concreteProduct, [MacWindow, Window]).
role(AF1, concreteProduct, [MacButton, Button]).
role(AF1, abstractRelation, [Look, Window]).
role(AF1, abstractRelation, [Look, Button]).
role(AF1, concreteRelation, [MSLook, MSWindow]).
role(AF1, concreteRelation, [MacLook, MacWindow]).
role(AF1, concreteRelation, [MSLook, MSButton]).
role(AF1, concreteRelation, [MacLook, MacButton]).
role(AF1, abstractFactoryMethod, [newWindow, Look, Window]).
role(AF1, abstractFactoryMethod, [newButton, Look, Button]).
role(AF1, concreteFactoryMethod, [newWindow, MSLook]).
role(AF1, concreteFactoryMethod, [newWindow, MacLook]).
role(AF1, concreteFactoryMethod, [newButton, MSLook]).
role(AF1, concreteFactoryMethod, [newButton, MacLook]).
```

Fonte: Mens e Tourwé (2001)

As refatorações baseadas em *Intent Aspects* contém os *Intent Aspects* que são pontos na estrutura de determinado software que são passíveis de aplicação de

um padrão (RAM, RAJESH, 2004). Ram e Rajesh (2004) fazem inicialmente a definição textual do *Intent Aspect* para determinado padrão e após isso, fazem a definição valendo de cláusulas em Prolog.

A Figura 6 ilustra a definição textual do padrão *Singleton* em que a classe sob avaliação deve ter seus membros estáticos públicos e deve ter somente uma instância em toda a aplicação.

Figura 6 - Detecção de classe candidata *Singleton*

```
Singleton(?C) :-
   ( class(?C),
   forall( context(?Member, ?C),
   modifier(?member, 'static'),
   modifier(?member, 'public'));
   instances(?C, ?Number),?Number==1.
   ) .
```

Fonte: Mens e Tourwé (2001)

A definição da busca da classe candidata para aplicação do padrão *Singleton* (Figura 6) é feita com TyRuBa (TYRUBA, 2018) que é uma linguagem derivada de Prolog. As clausulas do *Intent Aspect* para o parão *Singleton,* seguem os seguintes passos: 1) obtém-se inicialmente as classes; 2) são verificados seus membros estáticos; 3) para cada membro estático é verificado se ele também tem acesso público; e finalmente, 4) verifica-se se há somente uma instância daquela classe na aplicação.

2.2 MÉTRICAS

Durante os estudos no tema de refatoração com padrões de projeto levantados para este trabalho, verificou-se que não existem ferramentas ou abordagens que façam a avaliação antecipada dos benefícios da aplicação de padrões de pojeto. A avaliação de código-fonte pode ser feita por métricas que possibilitam obter valores quantitativos.

Xenos et al. (2000) apresenta uma pesquisa apontando métricas em projetos orientados a objeto voltadas a diversos cenários: métricas de classes, de acoplamento, de métodos, de herança e de sistema. A qualidade de determinado

projeto orientado a objeto pode ser avaliada se apropriando das medidas extraídas do mesmo.

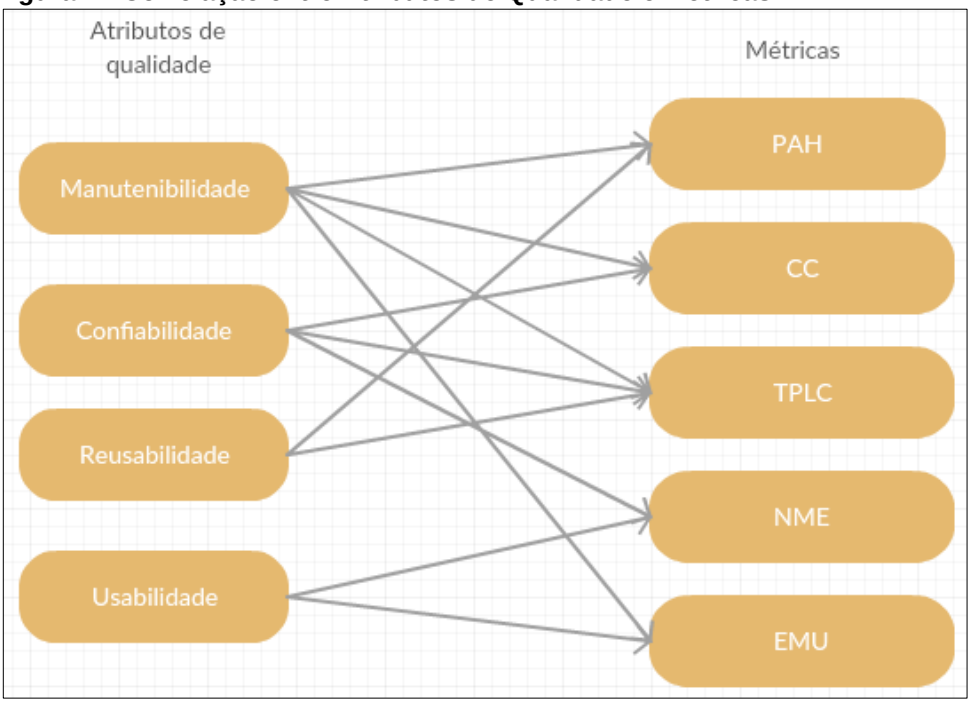
Outra forma de se verificar a qualidade de código-fonte, é através de atributos de qualidade. Olsina et al. (1999) separa-os em uma hierarquia de atributos tais como: Usabilidade, Funcionalidade, Confiabilidade, Eficiência e Manutenibilidade.

Considerando ambas as possiblidades, existem maneiras de correlacionar métricas de software e atributos de qualidade. Este tipo de correlação é útil pois apesar das métricas serem bons instrumentos de medição do código-fonte, são um recurso mais técnico para obtenção de dados. Termos como complexidade ciclomática e profundidade da árvore de herança podem ser recebidos com dificuldades por aqueles que os estudam, por isso, é interessante que sejam usados termos para se avaliar um determinado produto de software, como por exemplo, as nomenclaturas de atributos de qualidade.

Sommerville (2011) faz uma correlação entre atributos de qualidade com cinco métricas de software (Figura 7). Esta correlação tem por objetivo apontar quais métricas tem influência sobre determinado atributo de qualidade, por exemplo o atributo de qualidade Usabilidade é afetado quando há variações nas métricas de número de mensagens de erro (NME) e extensão do manual do usuário (EMU).

As demais métricas apresentadas por Sommervile (2011) em sua correlação são (Figura 7): profundidade da árvore de herança (PAH), complexidade ciclomática (CC), tamanho do programa em linhas de código (TPLC).

Figura 7 - Correlação entre Atributos de Qualidade e Métricas



Fonte: Adaptado de Somerville (2011)

É mais simples ao entendimento do usuário final compreender, por exemplo, que a manutenibilidade do seu código foi aprimorada ou que seu projeto está mais reusável do que analisar os impactos que seu código-fonte apresentou a partir de uma complexidade ciclomática maior ou menor, ou até mesmo entender as implicações de uma maior extensão do manual do usuário em seu projeto.

O resultado da avaliação pode ser mais significativo quando o usuário consegue a entender o código em termos de atributos de qualidade, pois a avaliação deixa de ser técnica e passa a ser mais amigável.

2.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo relatou a importância de um processo de refatoração semi-automatizado o que permite uma maior assertividade no desenvolvimento de suas tarefas. Foi apresentado um breve descritivo de métodos de refatorações, bem como abordagens de extração e tipos de refatoração que podem ser aplicados ao processo de refatoração.

O tema de métricas foi abordado porque os métodos presentes na literatura não realizam uma avaliação antecipada sobre os benefícios da aplicação dos padrões de projeto no código-fonte. As métricas podem ser correlacionadas a atributos de

qualidade para que o usuário possa ter uma avaliação mais intuitiva dos benefícios da aplicação de um determinado padrão.

O estado da arte sobre refatoração baseada em padrões de projeto é apresentado no Capítulo 3 e foi realizado por meio de um mapeamento sistemático.

3 ESTADO DA ARTE

Considerando a grande quantidade de estudos já realizados e aqueles em desenvolvimento sobre inserção de padrões de projeto, uma prática comum é a realização de revisões da literatura de forma mais sistematizada. Esta revisão contribui para avaliar a área de pesquisa em que se deseja propor uma solução ainda não avaliada.

Até o presente momento, nenhum mapeamento foi realizado sobre o tema de refatoração de software baseado em padrões de projeto. Portanto, o mapeamento proposto foi realizado no período de 1997 a 2017, utilizou nove (9) repositórios digitais e seis (6) questões que ajudaram a identificar uma solução ainda não avalidada sobre o tema de inserção/detecção de padrões de projeto em código-fonte. Inicialmente, 1149 trabalhos foram obtidos e após passarem pelo processo de filtragem, 26 foram selecionados para uma avaliação minuciosa.

Este Capítulo está organizado em três seções principais. A Seção 3.1 apresenta um breve descritivo da metodologia de pesquisa usada para a realização do mapeamento sistemático, bem como a descrição dos passos executados da metodologia. A Seção 3.2 mostra os resultados encontrados a partir da execução do mapeamento, respondendo a cada uma das perguntas levantadas no início do mapeamento. A Seção 3.3 se utiliza do mapeamento realizado para descrever os trabalhos relacionados. Por fim, a última Seção apresenta as considerações finais do capítulo.

3.1 METODOLOGIA DE PESQUISA

Dentre os métodos encontrados na literatura para realizar o mapeamento sistemático, destaca-se o desenvolvido por Kitchenham e Charters (2007). Devido a sua relevância, alguns autores já utilizaram este método para executar sua revisão, tais como: Dyba e Dingsór (2008), Garousi e Mäntylä (2016), Guinea, Nain e Le Traon (2016), Martins e Gorschek (2016). Martins e Gorschek (2016) apontam que o trabalho de Kitchenham e Charters (2007) é uma referência a aqueles que desejam realizar revisões e mapeamentos sistemáticos e relatam que esta abordagem cobre diversos passos para gerar uma pesquisa abrangente.

O método proposto por Kitchenham e Charters (2007), apesar de muito usado, não contempla uma classificação de leitura sobre os trabalhos selecionados. A classificação permite identificar os trabalhos mais relevantes no tema de pesquisa e auxilia o pesquisador em sua leitura. Esta classificação é contemplada pelo método proposto por Pagani, Kovaleski e Resende (2015).

A metodologia para realizar o mapeamento sistemático usado neste trabalho foi a junção dos procedimentos de Kitchenham e Charters (2007) e Pagani e Kovaleski e Resende (2015). O mapeamento foi realizado pelo autor deste trabalho juntamente com duas professoras da área de Engenharia de Software. Foram levantados trabalhos relacionados ao foco desta pesquisa de 1997 a 2017 no assunto de métodos ou ferramentas para detectar e inserir padrões de projeto em código-fonte.

A princípio, como previsto pelo método, foi definido o protocolo pelos autores visando trazer ao pesquisador uma ampla quantidade de informações sobre: autores dos trabalhos, relevância de cada trabalho e sua real contribuição para a comunidade. Também, neste protocolo foram definidas as questões de pesquisa, bases de busca e suas formas específicas de pesquisa, *strings* de busca, palavras-chave e filtros, apresentadas nas próximas seções.

3.1.1 Questões de Pesquisa

O objetivo central da pesquisa foi realizar um levantamento das principais ferramentas e métodos utilizados em refatorações que aplicam padrões de projeto em código-fonte. As seguintes perguntas foram elaboradas:

- Q1. Qual o número de citações de cada trabalho?
- Q2. Qual o grupo de Padrão de Projeto (Criacional, Estrutural, Comportamental) o método de inserção utiliza?
- Q3. Quais foram os pesquisadores que desenvolveram métodos para a detecção de pontos de inserção de Padrões de Projeto nos últimos vinte anos? Qual a quantidade de trabalhos de cada autor?
- Q4. Quais são as ferramentas relacionadas aos métodos encontrados na pergunta Q3?
- Q5. Qual a evolução por ano dos trabalhos selecionados?

- Q6. Existe uma relação (dependência) entre os trabalhos dos pesquisadores do assunto? Por exemplo, *Autor1* usa o método proposto por *Autor2*.

Além do objetivo principal de se obter o estado da arte do tema proposto, outros benefícios das perguntas criadas podem ser observados: i) levantar quais grupos padrões já foram alvos de pesquisas prevendo possíveis *gaps* na área; ii) verificar como as ferramentas tem sido construídas para o tema proposto a fim de identificar seu funcionamento e interações com o usuário; iii) elencar quais são os principais autores da área fazendo com que o processo de busca de trabalhos seja facilitado pelo autor correlato; e iv) elaborar critério de relevância de trabalhos por meio de suas citações.

3.1.2 Bases e Formas de Pesquisa

A extração de estudos primários foi executada em 7 (sete) bibliotecas digitais, um *journal* (*IBM Techical Journals*) e também na web (*Google Scholar*). Todas as pesquisas nestes repositórios foram feitas pela frase exata, abrangendo o período entre 1997 e 2017 e nas bibliotecas digitais foi usado a pesquisa avançada. Entretanto, cada repositório possui uma maneira específica de realizar sua pesquisa, o Quadro 1 apresenta cada um dos repositórios, seu tipo e configuração de pesquisa específica.

Quadro 1 - Repositórios - Filtros

Repositórios	Tipo	Configuração de Pesquisa
ACM	Biblioteca Digital	"Title" Opção "The ACM Guide for Computing Literature
Google Scholar	Web	"Title"
IBM Techinical Journals	*Journal*	"Title"
IEEE xplore	Biblioteca Digital	"Publication Title" "Abstract Title" or "Abstract"
Science Direct	Biblioteca Digital	"Abstract, Title and Keywords"
Scopus	Biblioteca Digital	"Article title, Abstract, Keywords"
Software Engineering Institute	Biblioteca Digital	"Title"
Springer	Biblioteca Digital	"With the exact frase"
Wiley	Biblioteca Digital	"Publication Titles" "Abstract Titles" or "Abstract"

Fonte: Autoria Própria

3.1.3 Strings de Busca e Palavras-chave

Vários termos de busca já haviam sido pensados e discutidos entre os pesquisadores, o que facilitou o processo de composição das *strings* de busca e palavras-chave. As palavras-chave foram: *Design Patterns in Refactoring, Introduction of Design Patterns, Introduce Design Patterns, Towards Design Patterns, Refactoring to a Design Pattern* e *Refactoring Based on Design Patterns*. Já as *strings* de busca utilizadas foram *Design Pattern AND Transformation e Automated Refactoing AND Design Patterns.*

Os termos estão em inglês, este foi um filtro pré-estabelecido para busca de trabalhos com objetivo de trazer trabalhos de diversas partes do mundo, uma vez que a língua inglêsa possibilita este tipo de busca por ser mundialmente conhecida.

3.1.4 Critérios de Inclusão e Exclusão

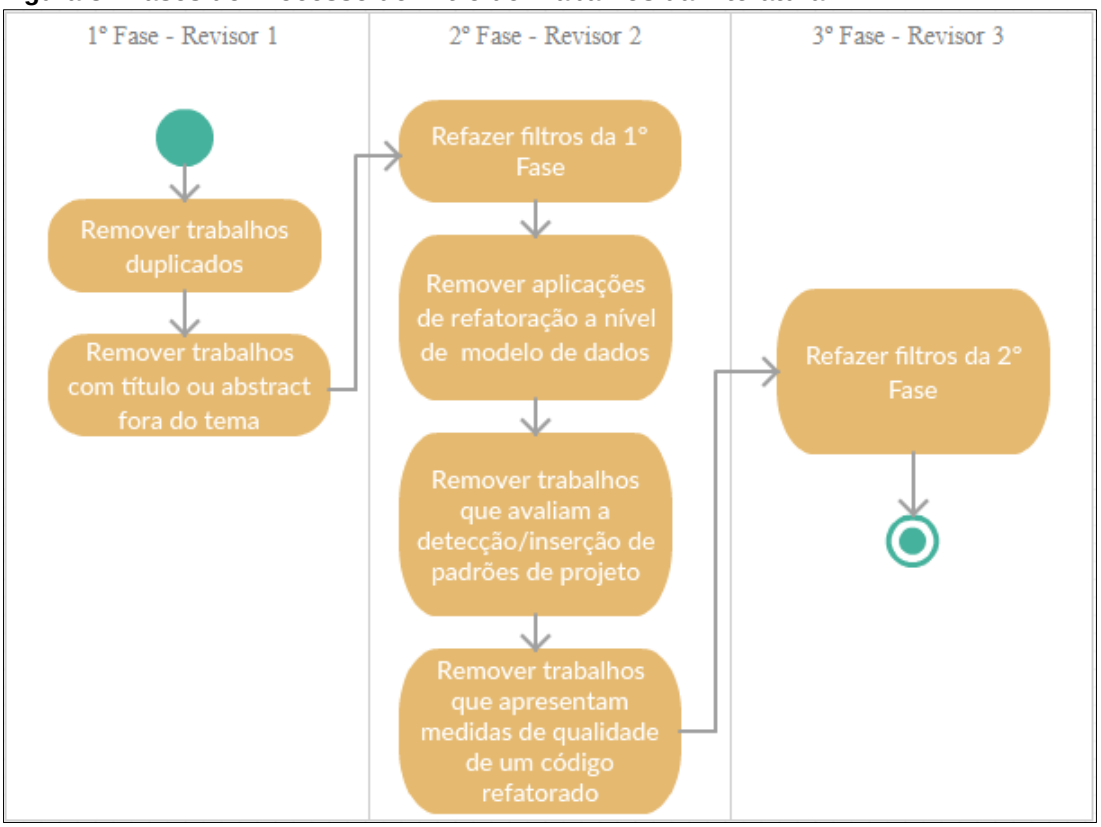
Os três pesquisadores que realizaram o mapeamento foram os avaliadores dos trabalhos primários coletados. Inicialmente foi previsto que os trabalhos válidos para a pesquisa seriam aqueles que apresentassem em seu Título, Palavras-Chave ou Abstract, informações relacionadas a: pontos inserção de padrões de projeto em código-fonte ou aplicação de padrões de projeto em código-fonte. Sendo estes dois tópicos o critério de inclusão desta pesquisa.

Posteriormente, foi executado um processo de filtragem que apresenta, dentro do contexto proposto, alguns critérios de exclusão na seleção de trabalhos. O processo de filtragem realizado pelos autores depois da pesquisa de trabalhos primários foi necessário porque haviam trabalhos obtidos por meio do seu *abstract*, *strings* de Busca ou palavras-chave. No caso das *strings* de Busca e palavras-chave haviam algumas genéricas, por exemplo, *Design Patterns AND Transformation* que trouxe trabalhos irrelevantes e que foram selecionados neste primeiro conjunto.

Este processo de filtragem foi desenvolvido em três fases, nos quais cada autor ficou responsável por uma das fases de filtragem (Figura 8). A primeira fase de filtragem removeu trabalhos duplicados, logo após, foi realizada a avaliação dos trabalhos para verificar se estes realmente se encaixavam na pesquisa. Isto foi realizado avaliando o título ou do *abstract* da pesquisa em questão.

A quantidade de trabalhos foi reduzida significativamente neste estágio e o motivo foi porque as *strings de Busca* e palavras-chave possuiam palavras mais genéricas. Constatou-se que várias delas não apresentavam nem mesmo o tema de refatoração tais como: *Introduction of Design Patterns, Towards Design Patterns*, etc. Apesar de não terem explicitamente o tema refatoração, outros termos similares como *Introduction* foram utilizados.

Figura 8 - Fases do Processo de Filtro de Trabalhos da Literatura



Fonte: Autoria Própria

Como havia uma grande quantidade de trabalhos no primeiro processo de filtragem, é possível que o executor do primeiro filtro tenha considerado estudos que não estavam dentro do foco do assunto especificado para o mapeamento. Por isto, a segunda fase de filtragem também seguiu as condições adotadas no primeiro filtro em termos de remoção de duplicados e análise de título e *abstract*.

Os filtros exclusivos da 2º Fase foram mais rigorosos, eliminando trabalhos relacionados a:

- Refatorações de modelos: trabalhos que estão relacionados à refatoração do diagrama de classe (modelo) e não em nível de código-fonte.
- Avaliações de detecção de pontos de inserção de padrões de projeto: trabalhos que abordam a aplicação de um método de refatoração para avaliar sua eficácia; mas não é implementada uma nova abordagem para o processo de refatoração.

A segunda fase de filtragem trata de um filtro no qual os trabalhos que divergiram de métodos/ferramentas/revisões sistemáticas de detecção de pontos de inserção de padrões de projeto foram excluídos. Isto reduziu a quantidade de trabalhos que estavam dentro do assunto do mapeamento.

A terceira fase de filtragem foi conduzida usando as mesmas restrições apresentadas na segunda fase, prevendo falhas do revisor da fase anterior.

A última forma de exclusão foi realizada para trabalhos de menor relevância. Estes foram obtidos após a execução de critérios definidos por Pagani e Kovaleski e Resende (2015).

3.2 RESULTADOS

Os trabalhos obtidos na primeira fase foram agrupados por suas *strings* de busca e palavras-chave juntamente com a quantidade de estudos encontrados. A Tabela 1 mostra o resultado deste agrupamento e a quantidade de trabalhos que foram selecionados considerando os filtros adotados.

Tabela 1 - Trabalhos Primários - Filtros

***strings* de busca e palavras-chave**	**Busca Primária**	**Processo de Exclusão**		
		1º Filtro	**2º Filtro**	**3º Filtro**
Desing Patterns in Refactoring	13	8	5	3
Introduction of Desing Patterns	49	8	5	1
Introduce Design Patterns	108	29	4	2
Towards Design Patterns	35	11	7	1
Refactoring to a Desired Pattern	0	0	0	0
Refactoring Based on Design Patterns	8	4	3	1
Design Patterns AND Transformation	908	165	24	9
Automated Refactoring AND Design Patterns	28	11	11	9
Total	1149	236	59	26

Fonte: Autoria Própria

Em relação aos trabalhos primários, foram identificados: seu número de citações, o qual é encontrado no *Google Scholar* (GOOGLE SCHOLAR, 2018); o fator de impacto, que está presente na lista de fatores de impacto do *SCIJournal* (ACM, 2018); e o ano de publicação (encontrado na referência do artigo) para gerar o *InOrdinatio* de cada um deles. O *InOrdinatio* é uma fórmula (1) estabelecida pelo método de Pagani e Kovaleski e Resende (2015) para classificar os artigos de acordo com sua relevância.

$$InOrdinatio = \left(\frac{IF}{1000}\right) + \alpha * [10 - (ResearchYear - PublishYear)] + (\Sigma\, Ci) \qquad (1)$$

onde: o *IF* é o Fator de Impacto do artigo baseado no ano de publicação do mesmo; o *α* é um valor a ser definido pelo autor da revisão, previsto como um peso variando de 1 a 10, o qual da um peso maior ao ano do trabalho no critério de classificação, caso seja definido com um valor mais próximo ou igual a 10; *ResearchYear* é o ano que a pesquisa foi realizada, neste caso, 2017; *PublishYear* é o ano em que o artigo foi publicado e por fim; é levantado o total de citações feitas ao artigo ($\Sigma\, Ci$).

O valor do *α* usado neste trabalho foi 2 (dois) para que os trabalhos mais antigos pudessem ter uma boa classificação independentemente do seu ano de publicação. Após o uso da fórmula para cada trabalho, os mesmos foram ordenados (*ranking*) e o resultado está apresentado na Tabela 2.

Tabela 2 - Trabalhos Selecionados - *Ranking InOrdination* (Fator de Impacto)

Ranking	Autor	Título	Fator de Impacto	Citações	Ano Publicação	InOrdination
4	Gaitani, M. A. G.; Zafeiris, V. E.; Diamantidis, N. A.; Giakoumakis, E. A.	*Automated refactoring to the Null Object design pattern*	1569	12	2015	29,57
5	Christopoulou, A.; Giakoumakis, E. A.; Zafeiris, V. E.; Soukara, V.	*Automated refactoring to the Strategy design pattern*	1328	14	2012	25,33
8	Zafeiris, V. E.; Poulias, S. H.; Diamantidis, N. A.; Giakoumakis, E. A.	*Automated refactoring of super-class method invocations to the Template Method design pattern*	1569	0	2017	21,57

Fonte: Autoria Própria

Neste primeiro *ranking* observa-se que do total de 26 trabalhos resultantes das fases de filtragem (Tabela 1), a grande maioria dos trabalhos não está presente na Tabela 2 por não terem um Fator de Impacto. A ausência deste valor ocorre porque não foram publicados em *Journals* ou porque o *Journal* não tem Fator de Impacto no ano de publicação daquele determinado estudo.

Um segundo *ranking* foi criado visando englobar os demais 23 (vinte e três) trabalhos restantes do processo de filtragem. Para tal, o cálculo do *InOrdinatio* foi adaptado, desta vez usando o índice H5 provido pelo *Google Scholar* (GOOGLE SCHOLAR, 2018) ao invés de usar o Fator de Impacto, conforme ilustra a fórmula (2).

$$InOrdinatio = \left(\frac{H5}{10}\right) + \alpha * [10 - (ResearchYear - PublishYear)] + (\Sigma\, Ci) \qquad (2)$$

Os trabalhos que se utilizaram desta nova fórmula, estão dispostos no segundo *ranking* (Tabela 3) e foram igualmente classificados conforme seu *InOrdinatio*.

Tabela 3 - Trabalhos Selecionados - *Ranking InOrdination* (H5)

Ranking	Autor	Título	Índice H5 (google metrics)	Citações	Ano Publicação	InOrdination
1	Cinnéide, M. Ó.	*Automated application of design patterns: a refactoring approach*	0	120	2001	108,0
2	Cinnéide, M. Ó.; Nixon, P.	*Methodology for the automated introduction of design patterns*	27	100	1999	86,7
3	Jensen, A.C.; Cheng, B.H.C.	*On the use of genetic programming for automated refactoring and the introduction of design patterns*	30	53	2010	62,0
4	Mens, T.; Tourwé, T.	*A declarative evolution framework for object-oriented design patterns*	27	53	2001	43,7
5	Jeon, S.U.; Lee, J.S.; Bae, D. H.	*An automated refactoring approach to design pattern-based program transformations in Java programs*	13	52	2002	43,3
6	Cinnéide, M.Ó.; Nixon, P.	*Automated Software Evolution Towards Design Patterns*	0	49	2001	37,0
7	Batory, D.; Tokuda, L.	Automated Software Evolution via Design Pattern Transformations	0	59	1995	35,0
8	Cinneide, M. Ó.	Automated refactoring to introduce design patterns	63	38	2000	30,3
9	Schulz, B.; Genssler, T.; Mohr, B.; Zimmer, W.	On the Computer Aided Introduction of Design Pattern into Object-Oriented Systems	0	47	1998	29,0
10	Zhao, C.; Kong. J.; Zhang. K.	*Design Pattern Evolution and Verification Using Graph Transformation*	0	26	2007	26,0

Tabela 3 - Trabalhos Selecionados - *Ranking InOrdination* (H5)

(continua)

Ranking	Autor	Título	Índice H5 (google metrics)	Citações	Ano Publicação	InOrdination
11	Ouni, A.; Kessentini, M.; Ó cinnéide, M.; Sahraoui, H.; Deb, K.; Inoue, K.	*MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells*	0	0	2017	20,0
12	Rajesh, J.; Janakiram, D.	*JIAD: A tool to infer design patterns in refactoring*	0	24	2004	18,0
13	Liu, W.; Hu, Z.; Liu, H.; Yang, L.	*Automated pattern-directed refactoring for complex conditional statements*	17	2	2014	17,7
14	Kim, J.; Batory, D.; Dig, D.	*Scripting Refactorings in Java to Introduce Design Patterns*	0	1	2014	15,0
15	Shimomura, T.; Ikeda, K.; Takahashi, M.	*An approach to GA-driven automatic refactoring based on design patterns*	0	5	2010	11,0
16	Shimomura, T.	*GA-driven Automatic Refactoring based on Design Patterns*	0	0	2012	10,0
17	Wheatman, M.; Liu, K.	*Automating software design pattern transformation*	17	1	2009	6,7
18	*Ram, D. Janaki; Rajesh, J.*	*Detecting Intent Aspects from Code to Apply Design Patterns in Refactoring: An Approach Towards a Refactoring Tool*	*0*	*6*	*2004*	*0,0*
19	*Cinnéide, M.Ó.; Nixon, P.*	*Program restructuring to introduce design patterns*	*24*	*13*	*1998*	*-2,6*

Tabela 3 - Trabalhos Selecionados - *Ranking InOrdination* (H5)

(conclusão)

Ranking	Autor	Título	Índice H5 (google metrics)	Citações	Ano Publicação	InOrdination
20	*Ziane, M.*	*A Transformational Viewpoint on Design Patterns*	*31*	*8*	*2000*	*-2,9*
21	*Lano, K.; Malik, N.*	*Mapping Procedural Patterns to Object-Oriented Design Patterns*	*0*	*4*	*1999*	*-12,0*
22	*Li, M.*	*Design Patterns in Software Refactoring: Theory and Practice*	*0*	*0*		
23	*Cinnéide, M.Ó.; Nixon, P.*	*Automated Refactoring Applied to the Gamma et al Design Patterns*	*0*	*0*		

Fonte: Autoria Própria

Esses *rankings* foram desenvolvidos para remover trabalhos que apresentam um valor de *InOrdinatio* menor que 0. Este tipo de ação (exclusão do mapeamento) é esperada quando os *rankings* são construídos, uma vez que o conjunto completo de trabalhos relevantes para o estudo foi levantado. Os trabalhos dos *rankings* foram lidos e tiveram seus dados extraídos para um formulário e os trabalhos fora de contexto foram excluídos.

Com exceção das primeiras exclusões baseadas no Título e *Abstract* dos trabalhos, as exclusões apresentadas no Quadro 2 foram baseadas no conteúdo interno do trabalho.

Quadro 2 - Exclusões - *Ranking InOrdination* (H5)

Ranking	**Motivo da Exclusão dos Trabalhos**
3,7,9 e 10	A inserção de padrões de projeto é feita em nível de modelo, não em nível de código-fonte.
15 e 16	É um método de avaliação de código-fonte dirigido a padrões de projeto.
17	É uma interpretação textual de padrões de projeto em nível de modelo.

Fonte: Autoria Própria

Vale mencionar que o critério de exclusão deste passo é o mesmo aplicado nos primeiros filtros, nada foi mudado nos critérios de inclusão ou exclusão. Outra observação é que somente o *Ranking* H5 teve trabalhos fora do contexto proposto,

sendo assim, a coluna *Ranking* presente no Quadro 2 é referente ao *Ranking* H5 da Tabela 4.

Portanto, ficou-se com 3 (três) trabalhos provenientes da Tabela 2 e 23 (vinte e três) resultantes da Tabela 3. Dos 23 trabalhos da Tabela 3, 12 foram excluídos (5 porque o valor do *InOrdinatio* foi menor que 0 (*Ranking* 19 ao 23) e 7 pelos motivos apresentados no Quadro 2). Com isso, 14 (quatorze) trabalhos foram selecionados para avaliação e foram lidos na integra. Constatou-se que, as referências bibliográficas contidas nos 14 (quatorze) artigos selecionados, existiam trabalhos relacionados ao tema do mapeamento sistemático. Portanto, foram selecionados e ranqueados conforme ilustra a Tabela 4.

Tabela 4 - Referências Bibliográficas Finais - *Ranking InOrdination* (H5)

Ranking	Autor	Título	H5	Citações	Ano Publicação	InOrdination
1	Kerievsky, J.	*Refactoring to Patterns*	*0*	*742*	*2008*	*741,0*
2	*Eden, A.H.; Gil, J.; Yehudai, A.*	*Precise specification and automatic application of design patterns*	*31*	*168*	*1997*	*151,1*
3	Hotta, K.; Higo, Y.; Kussumoto, S.	*Identifying, tailoring and suggesting form template method refactoring Opportunities with program dependence graph.*	*25*	*42*	*2012*	*54,5*
4	Jullerat, N.; Hirsbrunner, B.	*Toward an implementation of the "Form Template Method" Refactoring*	*13*	*29*	*2007*	*30,3*
5	Kim, J.; Batory, D.; Dig, D.	*Scripting parametric refactorings in java to retrofit design patterns*	*0*	*12*	*2015*	*28*
6	Ajouli, A.; Cohen, J.; Royer, J.-C.	*Transformations between composite and visitor implementations in Java*	*16*	*9*	*2013*	*22,6*
7	Ouni, A.; Kessentini, M.; Sahraoui, H. Cinnéide, M. Ò. Deb, K.; Inoue, K.A.	*A multi-objective refactoring approach to introduce design patterns and fix anti-patterns*	*0*	*1*	*2015*	*17*

Fonte: Autoria Própria

O conjunto completo de pesquisas analisadas neste trabalho combina os 14 (quatorze) trabalhos resultante dos *Rankings* com Fator de Impacto e H5 com o *Ranking* das Referências Bibliográficas (Tabela 4). A seleção final de trabalhos está presente na Quadro 3. Além disso, para cada trabalho foi dado um valor de Chave (S_n, onde *S* representa o trabalho selecionado e *n* é um número sequencial para sua identificação), que será utilizado posteriormente como uma referência simplificada do artigo.

Quadro 3 - Trabalhos Finais Selecionados

Chave	Autor	Título	Ano Publicação
S1	Gaitani, M. A. G.; Zafeiris, V. E.; Diamantidis, N. A.; Giakoumakis, E. A.	*Automated refactoring to the Null Object design pattern*	2015
S2	Christopoulou, A.; Giakoumakis, E. A.; Zafeiris, V. E.; Soukara, V.	*Automated refactoring to the Strategy design pattern*	2012
S3	Zafeiris, V. E.; Poulias, S. H.; Diamantidis, N. A.; Giakoumakis, E. A.	*Automated refactoring of super-class method invocations to the Template Method design pattern*	2017
S4	Ouni, A.; Kessentini, M.; Sahraoui, H. Cinnéide, M. Ò. Deb, K.; Inoue, K.A.	*A multi-objective refactoring approach to introduce design patterns and fix anti-patterns*	*2015*
S5	Cinnéide, M. Ó.	*Automated application of design patterns: a refactoring approach*	2001
S6	Cinnéide, M. Ó.; Nixon, P.	*A Methodology for the automated introduction of design patterns*	1999
S7	Mens, T.; Tourwé, T.	*A declarative evolution framework for object-oriented design patterns*	2001
S8	Jeon, S.U.; Lee, J.S.; Bae, D. H.	*An automated refactoring approach to design pattern-based program transformations in Java programs*	2002
S9	Cinnéide, M.Ó.; Nixon, P.	*Automated Software Evolution Towards Design Patterns*	2001
S10	Cinneide, M. Ó.	*Automated refactoring to introduce design patterns*	2000

Quadro 3 - Trabalhos Finais Selecionados

(continua)

Chave	Autor	Título	Ano Publicação
S11	Liu, W.; Hu, Z.; Liu, H.; Yang, L.	*Automated pattern-directed refactoring for complex conditional statements*	2014
S12	*Ram, D.J; Rajesh, J.*	*Detecting Intent Aspects from Code to Apply Design Patterns in Refactoring: An Approach Towards a Refactoring Tool*	*2004*
S13	Hotta, K.; Higo, Y.; Kussumoto, S.	*Identifying, tailoring and suggesting form template method refactoring Opportunities with program dependence graph.*	*2012*
S14	Rajesh, J.; Janakiram, D.	*JIAD: A tool to infer design patterns in refactoring*	2004
S15	Ouni, A.; Kessentini, M.; Ó Cinnéide, M.; Sahraoui, H.; Deb, K.; Inoue, K.	*MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells*	2017
S16	Eden, A.H.; Gil, J.; Yehudai, A.	*Precise specification and automatic application of design patterns*	*1997*
S17	Kerievsky, J.	*Refactoring to Patterns*	*2008*
S18	Kim, J.; Batory, D.; Dig, D.	*Scripting parametric refactorings in java to retrofit design patterns*	*2015*
S19	Kim, J.; Batory, D.; Dig, D.	*Scripting Refactorings in Java to Introduce Design Patterns*	2014
S20	Jullerat, N.; Hirsbrunner, B.	*Toward an implementation of the "Form Template Method" Refactoring*	*2007*
S21	Ajouli, A.; Cohen, J.; Royer, J.-C.	*Transformations between composite and visitor implementations in Java*	*2013*

Fonte: Autoria Própria

Os dados extraídos pelos estudos possibilitaram a criação de resultados por meio de tabelas e gráficos, os quais são detalhados na próxima seção. Já a Seção 3.3.2 apresenta algumas informações que precisam ser destacadas sobre o processo de mapeamento realizado.

3.2.1 Respostas as Questões

Os resultados da aplicação do mapeamento sistemático são apresentados com base em cada uma das perguntas que foram levantadas no início do processo. Os trabalhos foram referenciados por meio da sua chave (Quadro 3).

Embora cada estudo selecionado tenha seu conteúdo extraído através de um Formulário de Extração, este formulário previu a obtenção dos seguintes dados: título, autores, palavras-chave, ano de publicação, número de citações, padrões de projetos utilizados pelo método ou ferramenta, se é um método ou ferramenta, nome da ferramenta relacionada ao método (se existir), utiliza métodos de outros autores; e resumo (objetivo, funcionamento do método ou ferramenta, limitações e trabalhos futuros).

Estes formulários contendo as extrações não estão presentes neste documento, isto porque os seus respectivos dados estão distribuídos em cada resposta para as perguntas levantadas. Os resumos realizados são apresentados de forma condensada na Seção 3.3.

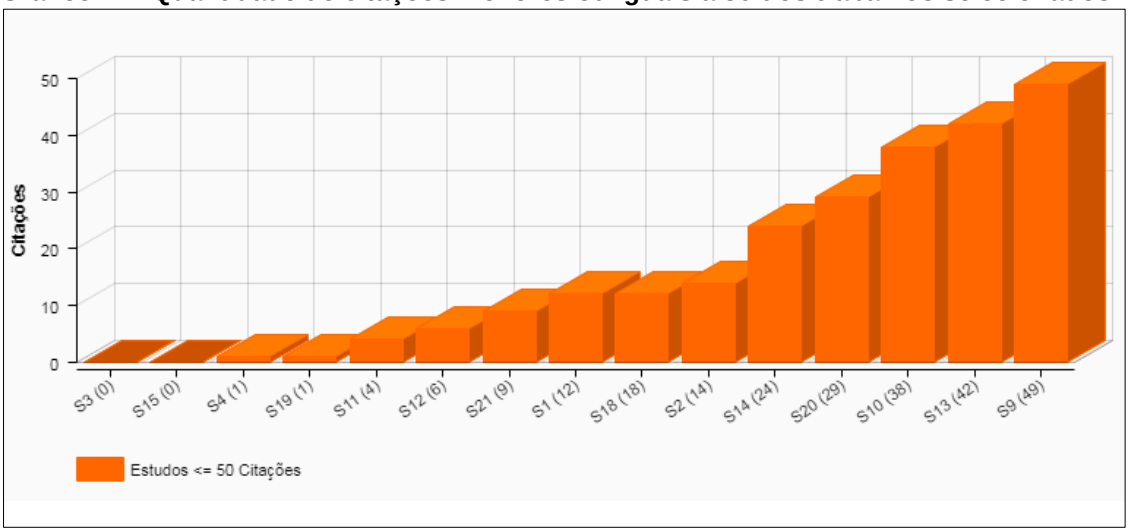
3.2.1.1Q1: Qual o número de citações de cada trabalho?

O método *InOrdinatio* utiliza o número de citações de cada trabalho para classificá-lo. Alguns trabalhos identificados no mapeamento proposto não apresentaram citações, mas, foram avaliados porque se observou que representam estudos atuais. Os trabalhos sem citações são:

- *Automated refactoring of super-class method invocations to the Template Method design pattern* (2017);
- *MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells* (2017).

Os trabalhos que apresentaram citações menor ou igual a 50 estão apresentados no Gráfico 1 e totalizaram 15 (quinze).

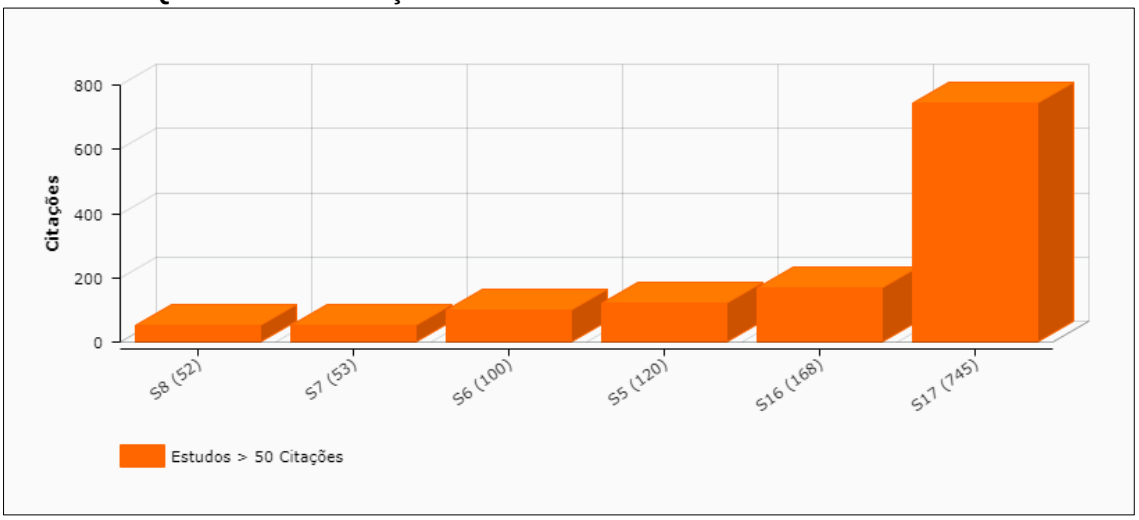
Gráfico 1 - Quantidade de citações menores ou iguais a 50 dos trabalhos selecionados



Fonte: Autoria Própria

O Gráfico 2 exibe os demais trabalhos, em que as citações foram maiores do que cinquenta (50) citações.

Gráfico 2 - Quantidade de citações maiores a 50 dos trabalhos selecionados

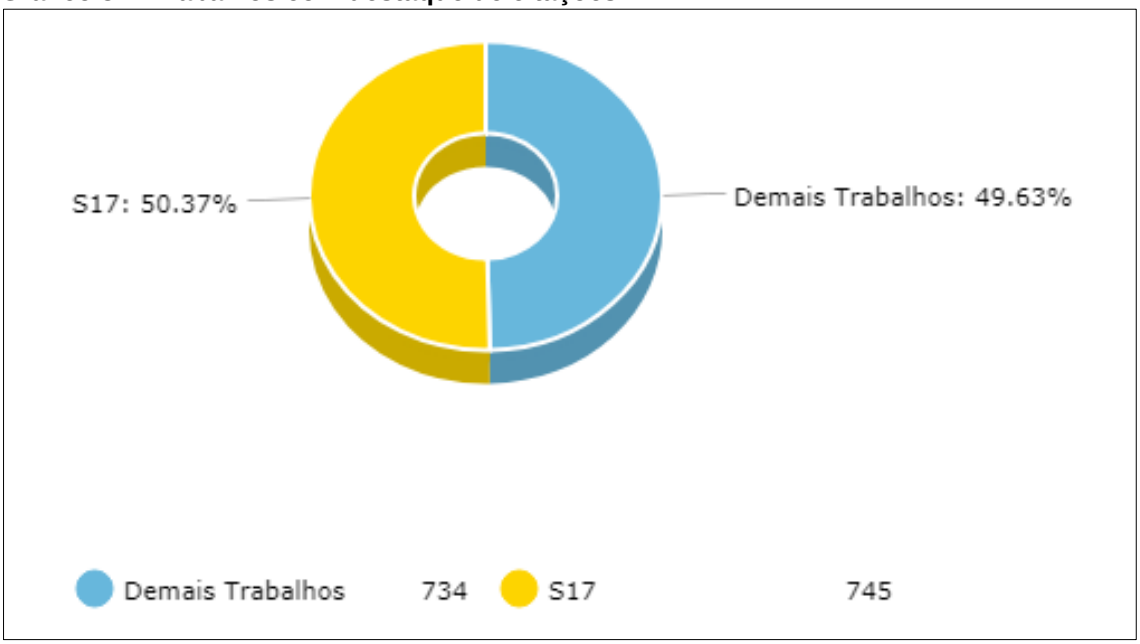


Fonte: Autoria Própria

Os Gráficos 1 e 2 identificam os trabalhos mais relevantes na área, os quais devem ser estudados quando o tema for refatorações voltadas a padrões de projeto.

O Gráfico 3 destaca o grande número de citações do trabalho S17 de Kerievsky (2008), apresentando o número de citações igual a 745, o qual supera o somatório total dos demais trabalhos encontrados (734) para o estudo do tema proposto.

Gráfico 3 - Trabalhos com destaque de citações



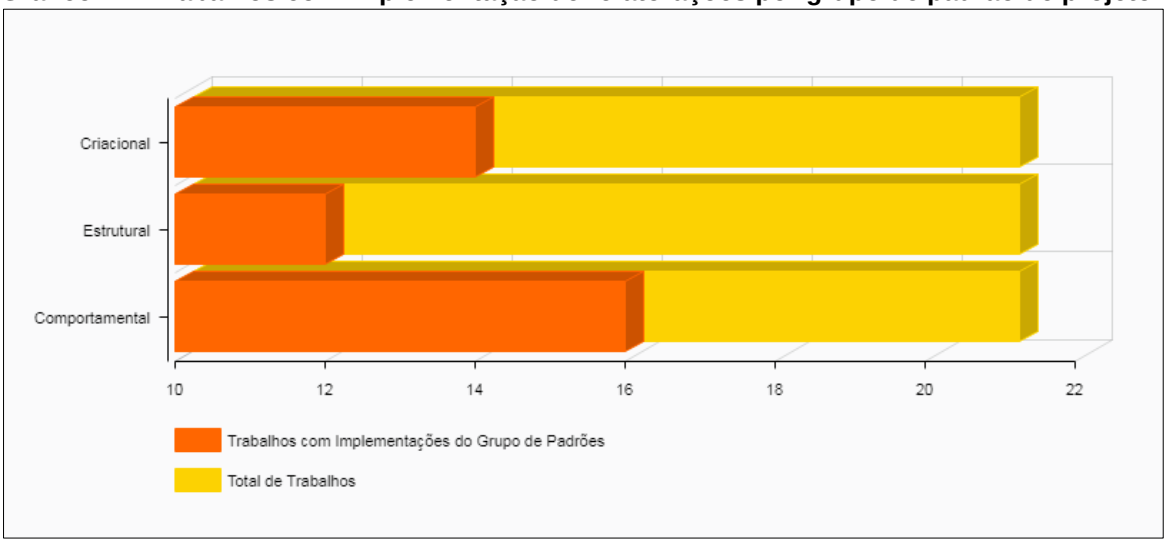
Fonte: Autoria Própria

Portanto, pesquisas desenvolvidas por Kerievsky (2008) são importantes para futuros pesquisadores interessados na área de refatoração de softwares voltada a aplicação de padrões de projeto.

3.2.1.2 Q2: Qual o grupo de Padrão de Projeto (Criacional, Estrutural, Comportamental) o método de inserção utiliza?

O Gráfico 4 apresenta o número de trabalhos que implementa um tipo de refatoração de código-fonte voltada a um grupo específico de padrões de projeto. Têm-se como referênca (barras ao fundo) a totalidade de trabalhos obtidos no mapeamento sistemático, a saber, 21 (vinte e um); em destaque (barras em primeiro plano), o número de trabalhos que se utilizam do grupo de padrões de projeto.

Gráfico 4 - Trabalhos com implementação de refatorações por grupo de padrão de projeto



Fonte: Autoria Própria

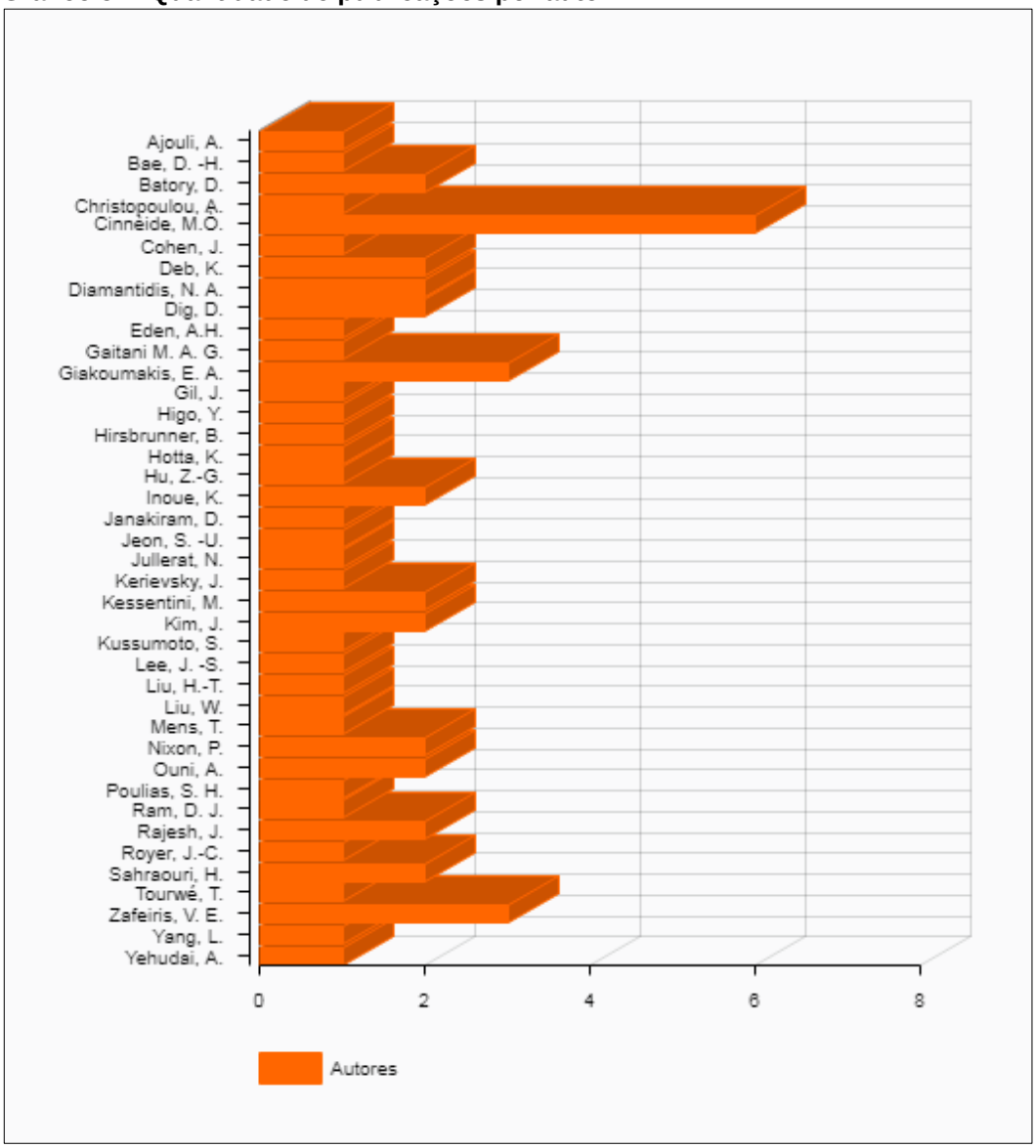
Uma observação a ser feita sobre a escolha dos padrões para estudo de automatização, é que inicialmente em sua grande maioria, os autores preferem estudar a inserção de padrões criacionais, devido a facilidade de criação e detecção de pontos de inserção. A possiblidade de inserção de padrões criacionais é feita por meio de uma avaliação da estrutura estática do código-fonte, ou seja, suas classes, métodos e atributos.

Com o passar dos anos o interesse por outros grupos de padrões aumentou e as descobertas iniciais deste tema permitiram o desenvolvimento de aplicações mais complexas. Isto pode ser observado no Gráfico 4, com o número de trabalhos desenvolvidos aplicando e detectando pontos de inserção de padrões comportamentais sendo mais alto que os métodos envolvendo padrões criacionais.

3.2.1.3 Q3: Quais foram os pesquisadores que desenvolveram métodos para a detecção de pontos de inserção de Padrões de Projeto nos últimos vinte anos? Qual a quantidade de trabalhos de cada autor?

Os autores foram obtidos das extrações, independentemente se é o primeiro autor ou não, ele foi listado para que se saiba em quais trabalhos esteve envolvido. O Gráfico 5 apresenta os resultados da pergunta Q3.

Gráfico 5 - Quantidade de publicações por autor



Fonte: Autoria Própria

Dentre os autores levantados neste mapeamento, de um total de 28 (vinte e oito), 5 (cinco) deles foram destacados por estarem relacionados a no mínimo 3 trabalhos: Cinnéide, M. Ó.; Giakoumakis, E. A.; e Zafeiris, V. E.

O Gráfico 5 exibiu os autores que apresentam uma maior expertise no desenvolvimento de métodos e/ou ferramentas voltadas as refatorações. Podendo, desta forma, serem utilizados como base teórica e prática de futuros trabalhos.

3.2.1.4 Q4: Quais são as ferramentas relacionadas aos métodos encontrados na pergunta Q3?

A mesma listagem da Questão 3 é disposta novamente, porém, desta vez com o objetivo de apresentar as ferramentas desenvolvidas pelos autores. O Quadro 4 apresenta o resultado sobre a pergunta Q4.

Quadro 4 - Ferramenta criada por trabalho

Chave(s)	Ferramenta
S2,S3	*JDeodorant*
S4	*MORE*
S6,S7,S8,S9	*Prototype*
S5,S10	*Design Pattern Tool*
S13	*Creios*
S12,S14	*JIAD*
S15	*MORE*
S1,S19,S20	*Eclipse Plugin*
S21	*JHotDraw*
S11,S16,S17,S18	-

Fonte: Autoria Própria

Nota-se que alguns autores somente apresentam seu método de refatoração, para outros (a maioria) inclui a criação de uma ferramenta. O uso de ferramentas adotada pelos autores aprimora a avaliação de sua proposta pois provê um *feedback* dos reais benefícios da implementação do código-fonte, em alguns casos, mostra limitações no desenvolvimento de seu método.

O Quadro 4 apresentou 7 (sete) ferramentas genéricas relacionada ao *Prototype* e *Eclipse Plugin* e que não foram detalhadas nos trabalhos que as contemplavam (S1, S6, S7, S8, S9, S19, S20). Os trabalhos S2, S3, S5, S10, S12, S14, S15, S13 e S21 apresentam 6 (seis) ferramentas específicas, criadas pelos autores ou atualizadas por eles (*JDeodorant, Design Patterns Tool*, *JIAD, MORE*, *Creios and JHotDraw*). Por fim, quatro (4) trabalhos S11, S16, S17 e S18 não criaram uma ferramenta para sua abordagem.

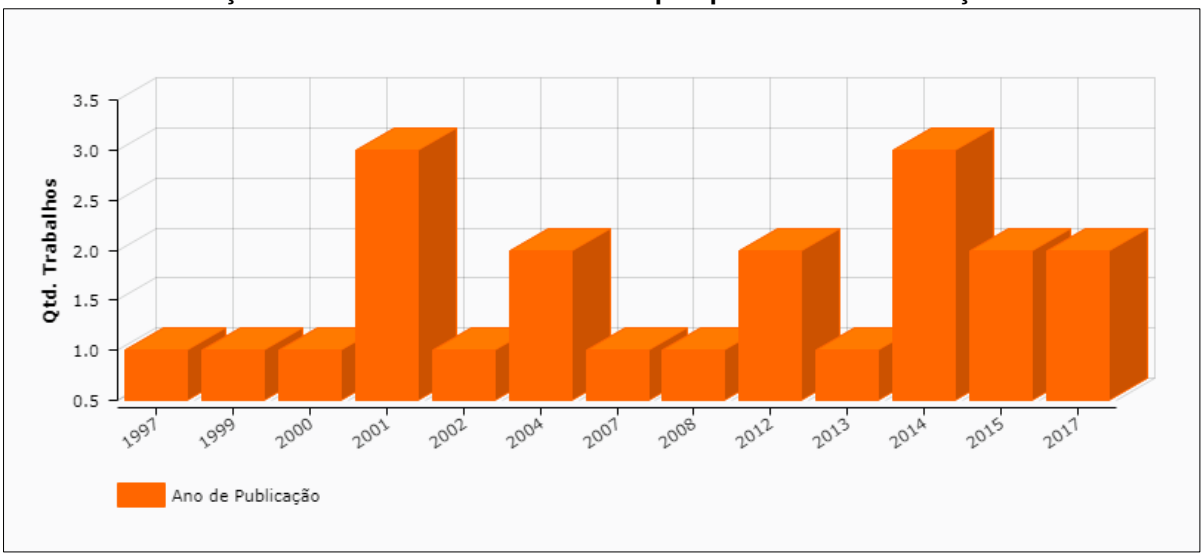
Considerando os dados apresentados no Quadro 4, vale ressaltar que a ferramenta chamada *Prototype*, não se refere a uma ferramenta que implementa os métodos S6, S7, S8 e S9; mas são apenas protótipos individuais feitos por cada um

dos métodos. O *Eclipse Plugin* apesar de ser desenvolvido para uma mesma ferramenta (Eclipse), não estão inseridos em sua versão oficial.

3.2.1.5 Q5: Qual a evolução por ano dos trabalhos selecionados?

Os trabalhos foram agrupados de acordo com seu ano de publicação. O Gráfico 6 mostra que este tipo de pesquisa é frequentemente estudada e que nos últimos cinco anos (2012-2017), exceto 2013, o interesse por este tema tem crescido.

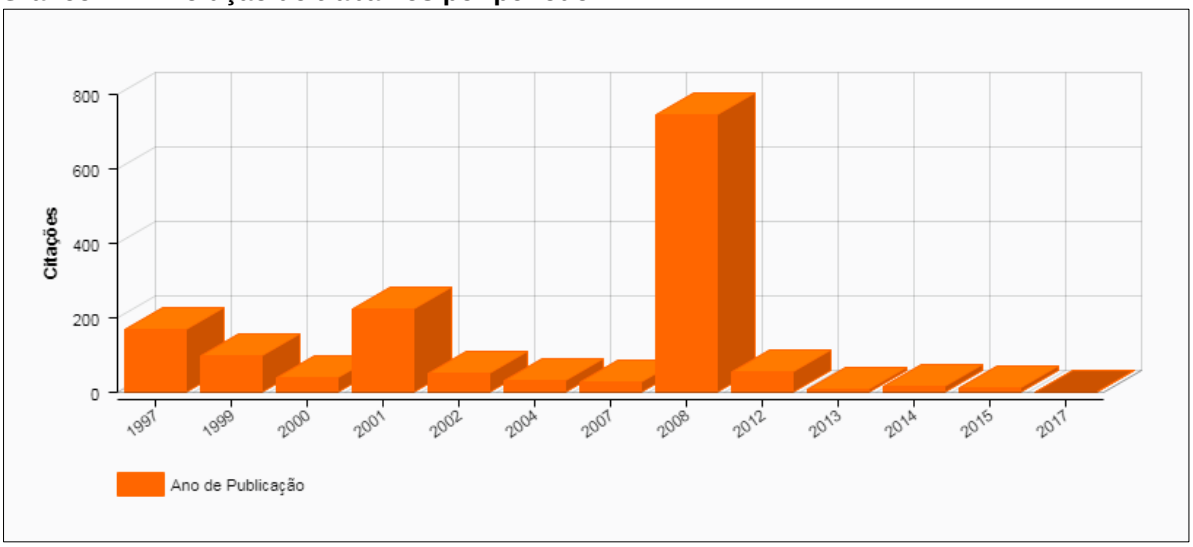
Gráfico 6 - Evolução de trabalhos sobre o tema de pesquisa desta dissertação



Fonte: Autoria Própria

Outra perspectiva relacionada ao ano de publicação é o total de citações que determinado ano possui. O Gráfico 7 apresenta todos os estudos realizados em um determinado ano e bem como as quantidades de citações.

Gráfico 7 - Evolução de trabalhos por período



Fonte: Autoria Própria

Uma conclusão importante a ser extraída deste último gráfico é que apesar dos trabalhos desenvolvidos no tema terem aumentado, ainda se observa que os trabalhos mais antigos são fundamentais para as novas pesquisas.

3.2.1.6 Q6: Existe uma relação (dependência) entre os trabalhos dos pesquisadores do assunto proposto por este mapeamento? Por exemplo, *Autor1* usa o método proposto por *Autor2*.

O propósito final desta busca foi descobrir correlações entre os métodos levantados neste mapeamento sistemático. Dentre as extrações, Mel Ó Cinnéide é responsável por 4 (quatro) dos 13 (treze) trabalhos selecionados para extração. Os trabalhos de Mel Ó Cinnéide se referem a mesma proposta de inserção de padrões de projeto: uma publicação inicial (S6), um pequeno descritivo desta mesma proposta (S10), um trabalho com alguns testes envolvendo mais padrões de projeto (S9) e finalmente a descrição detalhada de todos os conceitos e processos (S5).

Outros autores também apresentam uma evolução de seus trabalhos, tais como: como Rajesh que evoluiu seu trabalho de um método (S12) para uma ferramenta (S14); Ouni *et al*. que apresenta incialmente um conceito de sua ideia (S4), logo após, uma ferramenta aplicando os conceitos propostos (S15); Kim, Batory e Dig publicaram inicialmente uma descrição completa de seu método (S19), e somente depois, uma versão reduzida de seus estudos (S20).

Os únicos estudos que aplicaram mais de um método de refatoração são os dois produzidos por Ouni *et al*. (S4 e S15), ambos abrangem o método de Mens e Tourwé (S7) e Ajouli, Cohen e Royer (S21). O último dos estudos adiciona mais um método, o proposto por Cinnéide e Nixon (S6).

O trabalho de Jeon, Lee e Bae (S8) também usa o método desenvolvido por Mel Ó Cinnéide (S6 e S5) com a finalidade de aplicar padrões de projeto em código-fonte.

Com relação os demais trabalhos (S1, S2, S3, S11, S13, S16, S17, S18), eles não se utilizam de outros trabalhos de autores presentes neste mapeamento sistemático.

3.2.2 Lições Aprendidas

Durante o processo de obtenção, ordenação e extração dos trabalhos, foram observadas várias situações que merecem ser relatadas como lições aprendidas.

Apesar dos esforços em tentar reduzir o número de trabalhos irrelevantes ou que estivessem fora do escopo do mapeamento, utilizando processos de filtragem, ainda durante as leituras constatou-se que haviam trabalhos duplicados ou fora de escopo. Isto se deve em grande parte a divergências nas informações fornecidas pelas bibliotecas digitais de artigos.

Os trabalhos “Methodology for the automated introduction of design patterns” e “A methodology for the automated introduction of design patterns”, por exemplo, foram avaliados como distintos nos processos de filtragem. A duplicação só foi notada no processo de criação de *rankings* uma vez que eles apresentaram um fator de impacto similar. Acredita-se que o problema ocorreu devido a uma inconsistência de informações dos trabalhos contidos nas bibliotecas digitais.

Outro ponto crítico foi a falta de coerência em que certas bibliotecas digitais provêm informação sobre um dado trabalho. Como exemplo, basta analisar o primeiro Ranking H5 (Tabela 3); neste há dois trabalhos que seu ano de publicação não foi encontrado, por isto foram excluídos.

3.3 TRABALHOS RELACIONADOS

Durante o processo de leitura e extração de dados, foi observado algumas características comuns entre os trabalhos que abordaram o tema de refatoração baseada em padrões de projetos como: linguagem de programação na construção de sua ferramenta, tipos de refatoração, abordagens de extensão e validação.

A grande parte dos trabalhos se utiliza da linguagem de programação Java para a refatoração de um projeto, com exceção de S9, S7, S12, S16 e S17. Os métodos S9, S12, S16 e S17 apresentam processos para manipulação do código-fonte, não se aprofundam em detalhes de implementação e não apresentam uma linguagem definida. Por sua vez, S7 se baseia na linguagem de programação SOUL para manipulação do código-fonte (WUYTS, 2001).

Em relação as abordagens de extração de dados do código-fonte, ou seja, as formas que um código-fonte é transformado em determinada estrutura de dados para que possa ser avaliado e refatorado por um dado método, cinco abordagens principais foram encontradas: JavaCC, SOOT Framework, Árvores de Sintaxe Abstrata, Fatos Prolog e Compilador Eclipse IDE e estão apresentadas na Figura 9.

Figura 9 - Abordagens de extração de código-fonte

Fonte: Autoria Própria

Todas as abordagens de extração já foram descritas na Seção 2, porém, a que é destaque neste trabalho é de “Árvore de Sintaxe Abstrata”, devido ao maior número de trabalhos que se utilizam deste tipo de abordagem. Os pesquisadores se utilizam de árvores de sintaxe abstrata pois permitem a análise estática do código-fonte, isto é, que não provê informações sobre o comportamento do código sob avaliação, mas sim, de sua estrutura. Além disso, a consideram como uma forma mais simples e confiável de manipulação de código-fonte.

Os demais trabalhos que não estão presentes na Figura 9 (S7, S13, S16, S17 e S21), possuem abordagens de extração própria não comuns a outras pesquisas da literatura ou não apresentam a implementação da extração. Para esses os casos, os trabalhos não foram considerados como abordagens comuns de extração de dados.

A Figura 10 apresenta os estudos os quais foram agrupados pelo tipo de refatoração como: *minitransformations*, expressões condicionais, *intent aspects*, *reflective refactorings,* baseada em papéis e *composite* para *visitor* e *visitor* para *composite*.

Figura 10 - Tipos de refatoração do código-fonte

Fonte: Autoria Própria

Minitransformations, intent aspects, reflective refactorings e *refatorações baseadas em papéis* são tipos que foram definidos no Capítulo 2. *Composite* para *Visitor* e *Visitor* para *Composite* focam somente na transição entre esses dois padrões.

Os trabalhos S2 e S11 focam em expressões condicionais e transformam este tipo de estrutura em um padrão *Strategy* e *Factory Method*/*Strategy,* respectivamente.

Constatou-se que existem tipos que apresentam formas genéricas de refatoração de código-fonte, sendo voltadas a aplicação de diversos padrões como é o caso de *Ministransformations* e *Intent Aspects*. Existem também aqueles que têm um contexto específico a sua aplicação, como é o caso de *Refatorações de expressões condicionais* e *Refatorações Composite para Visitor e Visitor para Composite*. Portanto, observou-se a multiplicidade de opções de tipos de refatoração já presentes na literatura, ficando sob responsabilidade do pesquisador escolher qual será o foco de seu estudo e qual forma de refatoração utilizar.

Em relação ao processo de validação da abordagem ou método, alguns trabalhos como S6, S7 e S12 não apresentam verificações de eficácia de qualidade de sua proposta, somente descrevem o método proposto. Outros, aplicam sua refatoração e verificam manualmente se foi efetuada com sucesso, como é o caso de S5. Já os demais trabalhos possuem validações comuns entre eles como: se utilizar de outras ferramentas como *JHotDraw*, *JUnit*, *JFreeChart*, etc, para verificar se projetos de terceiros apresentam candidatos a refatoração (S11, S15 e S19); mostrar através de métricas de software a mudança causada pela refatoração, verificando a diferença das métricas de número de linhas de código e complexidade ciclomática entre o antes e o depois da refatoração (S1, S2, S11).

3.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou um mapeamento sistemático que utiliza como base no método de mapeamento proposto por Kitchenham e Charters (2007) e Pagani, Kovaleski e Resende (2015). As fases e a grande parte das etapas usadas no mapeamento são oriundas de Kitchenham e Charters (2007), exceto as etapas de classificação (*ranking*) de artigos que usam os princípios estabelecidos por Pagani, Kovaleski e Resende (2015) por meio do método *InOrdination*.

O objetivo do mapeamento foi aplicado para o levantamento de trabalhos relacionados à detecção de pontos de inserção de padrões de projeto em código-fonte. O mapeamento foi conduzido pelo autor deste trabalho e duas professoras da área de engenharia de software.

Após a realização do mapeamento, constatou-se que o assunto a ser desenvolvido por este projeto é relevante, pois houve um aumento de publicações a partir de 2014 e novas propostas estão sendo desenvolvidas. Foram identificados um total de 21 (vinte e um) trabalhos que abordam o assunto do mapeamento até 2017.

Além disso, a verificação dos processos internos possibilitou a identificação da problemática abordada neste trabalho que é criar uma abordagem que centraliza métodos de refatoração em um único ambiente, prevendo abordagens centralizadas de extração de dados do código-fonte e observando aqueles métodos com processos similares de refatoração.

Apesar de mostrarem melhoras significativas nos projetos sendo avaliados um ponto a ser destacado é que nenhum trabalho apresenta interações significativas com o usuário, pois os autores consideram que usuário não é parte vital ou não é considerado como essencial no processo de refatoração, ele meramente faz escolhas iniciais sobre os padrões a serem aplicados (quando é solicitado) e posteriormente recebe o código-fonte refatorado.

Outro ponto observado é que não são realizadas medidas de qualidade visando avaliar se realmente a refatoração proposta pela ferramenta é válida, se traz reais benefícios ao projeto ou se o usuário deseja realizar a referida refatoração. O trabalho de Ouni *et at*. (2017), por exemplo, faz verificações de qualidade, porém, estas são feitas após o processo de refatoração estar completo.

4 PROPOSTA DE ABORDAGEM DE REFATORAÇÃO

Conforme mencionado no Capítulo 3, os estudos não apresentam ferramentas ou métodos que avaliem a qualidade do código-fonte sobre a aplicação de padrões de projeto e uma interação com o usuário no processo de refatoração. Isto representa um ponto de aprimoramento, pois existe a possiblidade de que o padrão aplicado não forneça melhorias no código-fonte.

Dentre os trabalhos encontrados na literatura, somente os trabalhos de Ouni *et al*. (2015; 2017) se utilizam de mais de um método de refatoração em sua ferramenta MORE. Esta ferramenta contempla 3 (três) métodos de refatoração e prevê a aplicação de 4 (quatro) possíveis padrões de projeto: *Visitor, Factory Method, Singleton e Strategy*. A diferença de Ouni *et at*. (2015; 2017) para a abordagem proposta é que esses trabalhos não possuem o foco de centralizar métodos de refatoração de acordo com sua forma de extração de dados do código-fonte e não provê uma avaliação prévia dos impactos da refatoração no código-fonte sob avaliação.

Este capítulo apresenta a Abordagem para Detecção de Pontos de Inserção e Aplicação de Padrões de Projeto em Código-Fonte capaz de refatorar um código orientado a objeto usando como referência métodos da literatura de identificação de pontos de inserção de padrões, métricas e interação com o usuário.

O detalhamento da abordagem é realizado nas seções deste capítulo, em que a Seção 4.1 fornece uma visão geral da abordagem de refatoração, apontando seus módulos e elementos internos. A Seção 4.2 apresenta os requisitos básicos necessários para o funcionamento da abordagem. A Seção 4.3 detalha os módulos internos. A Seção 4.4 relata as considerações finais do capítulo.

4.1 VISÃO GERAL

A abordagem proposta tem como objetivo trazer novas formas de realizar os processos de refatoração por meio de funcionalidades adicionais e implementar pontos faltantes encontrados em outros trabalhos (Capítulo 3). Uma das melhorias é a possibilidade de inserir novos métodos de refatoração providos pela literatura, ou

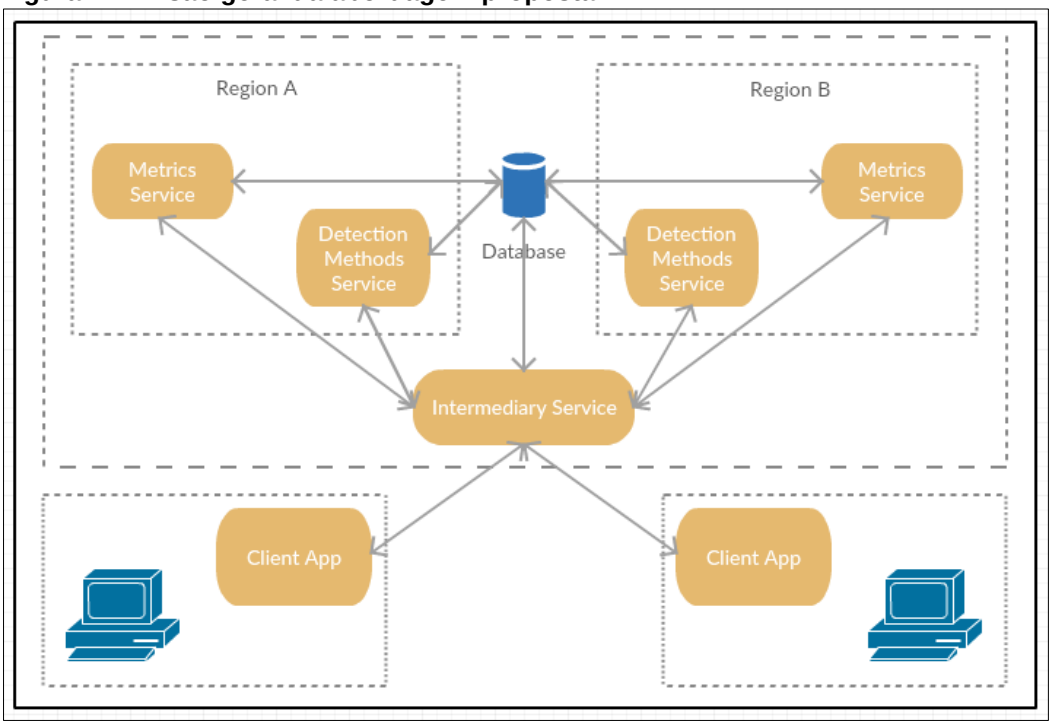
seja, fornecer um modelo em que um novo método pode se acoplar ao que já está desenvolvido.

Outra melhoria é possibilitar que o processo de refatoração forneça informações necessárias, como por exemplo:

- Identificar quais padrões podem ser aplicados.
- Aceitar ou não a aplicação do padrão.
- Avaliar antecipadamente os benefícios que a aplicação do padrão pode trazer para o código-fonte.
- Aplicar efetivamente o padrão escolhido pelo usuário.

A abordagem proposta (Figura 11) é composta de dois módulos internos (*Metrics Service* e *Detection Methods Service*), um intermediário (*Intermediary Service*) e um ou mais aplicativo(s) cliente (*Client App*) (BELUZZO; MATOS; PACHER; 2018). Aqueles que apresentam o termo *Service* em seu nome, são os que se utilizam do padrão Camada de Serviço, promovendo um modelo que tem por objetivo encapsular regras de negócios e controlar as respostas a serem dadas em cada operação (FOWLER *et al*., 2006).

Figura 11 - Visão geral da abordagem proposta



Fonte: Autoria Própria

É importante destacar dois conceitos que são utilizados ao longo do detalhamento dos módulos da abordagem. O núcleo da abordagem é composto pelo *Metrics Service* e *Detection Methods Service* (módulos internos) juntamente com o *Intermediary Service* (módulo intermediário). O núcleo representa a central de processamento da abordagem; já o módulo cliente promove interações com os usuários da abordagem (provavelmente, desenvolvedores de software) e é responsável por se comunicar com o módulo intermediário para realizar as operações de avaliação e refatoração.

Na estrutura da abordagem proposta existem duplicações de módulos, onde a diferença é nas *Regions* (A e B) que eles estão localizados. Essa separação de regiões foi feita visando aplicar prevenções a falha, considerando que, apesar dos Serviços caracterizarem o núcleo da arquitetura, eles não precisam estar presentes em um único computador/servidor, mas sim, podem ser dispersos em meio a rede.

A ideia é que cada membro da *Region* seja responsável por enviar mensagens diretamente ao *Intermediary Service*. O envio frequente de mensagens por parte de um membro das *Regions* o caracteriza como ativo, esses membros são os que permanecem em atividade para as funções providas na abordagem. Por exemplo, se um *Metrics Service* envia constantemente suas mensagens ao ativo *Intermediary Service*, então, este é adicionado a fila dos *Metrics Service*.

Para que o *Intermediary Service* faça a gestão dos membros ativos das *Regions* (Figura 11), existe uma fila para cada módulo interno da abordagem, sendo assim, uma fila para *Detection Methods Services* e outra para *Metrics Services*.

Uma vez que o *Intermediary Service* receba a requisição dos *Client Apps* ele as redireciona ao membro previsto para o processamento da requisição. Por exemplo, o *Intermediary Service* recebe mensagens *ativo* do *Detection Methods Service* da *Region* A, mas não do *Detection Methods Service* da *Region* B; uma vez que a detecção de candidatos a refatoração é requisitada por um *Client App*, o *Intermediary App* redireciona a requisição para o único membro *ativo*, a saber, o *Detection Methods Service* da *Region* A.

O número de *Regions* depende de dois fatores, o primeiro é proporcionar uma estrutura com uma maior tolerância a falhas, quanto mais *Regions* existirem, mais membros estarão disponíveis para processamento. O segundo é o fator de alta disponibilidade, considerando que cada *Region* pode ou deveria ser colocada em um

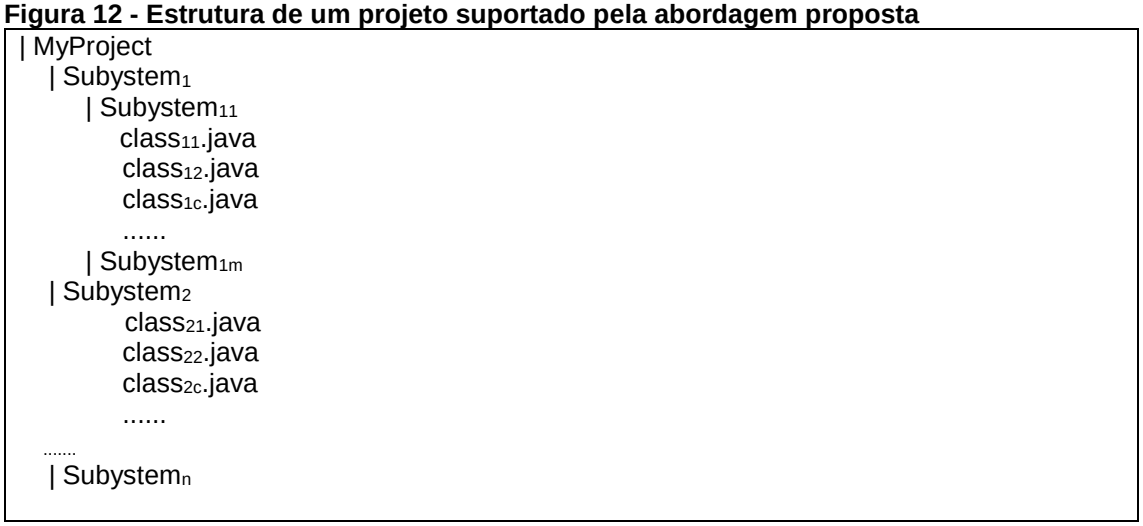
servidor diferente das demais, isso previne que uma única máquina seja sobrecarregada, gerando um tempo de resposta mais baixo para diversas requisições.

As seções seguintes explicam detalhadamente o funcionamento interno de cada um dos módulos previstos na abordagem proposta, bem como seus requisitos.

4.2 REQUISITOS BÁSICOS PARA O FUNCIONAMENTO DA ABORDAGEM PROPOSTA

A abordagem proposta necessita como entrada um projeto desenvolvido em orientação a objetos, mais especificamente, projetos em Java. Esta entrada foi escolhida porque diversos métodos como Cinnèide e Nixon (1999), Gaitani *et al*. (2015), Zafeiris *et at*. (2017) têm sua aplicação voltada a esta linguagem de programação. A entrada corresponde ao projeto que pode ser composto por um conjunto de subsistemas, conforme apresenta a Figura 12.

Figura 12 - Estrutura de um projeto suportado pela abordagem proposta



Fonte: Autoria Própria

Existem sistemas de software que são segregados em diversos projetos. É importante observar que a abordagem proposta só aceita 1 (um) projeto por avaliação, o que significa que se um sistema é constituído de vários projetos menores, então estes serão avaliados e refatorados separadamente. Cada subsistema do projeto (Subsytem$_1$, Subsystem$_2$, ..., Subsystem$_n$ onde n é o total de subsistemas) pode conter ou não outros subsistemas (por exemplo, Subsytem$_{11}$,..., Subsystem$_{1m}$ onde m é o total de subsistemas de Subsytem$_{11}$) e assim sucessivamente. O nível mais baixo de

um subsistema é um ou um conjunto de classes (por exemplo, $class_1$, $class_2$,..., $class_c$ onde c é o total de classes de $Subsytem_{11}$).

Os requisitos para o uso da abordagem são: i) identificar o processo dos métodos da literatura para detecção de pontos de inserção de padrões de projeto e ii) definir métricas para realizar a avaliação de qualidade do código-fonte.

O primeiro requisito se refere ao conhecimento dos processos dos métodos da literatura para inserção e detecção de pontos de padrões, explicados no Capítulo 3. O segundo requisito são meios para obtenção de métricas código-fonte com o objetivo de mensurar se a aplicação dos padrões oferece ganhos ao projeto como por exemplo de reusabilidade, manutenibilidade, entre outras.

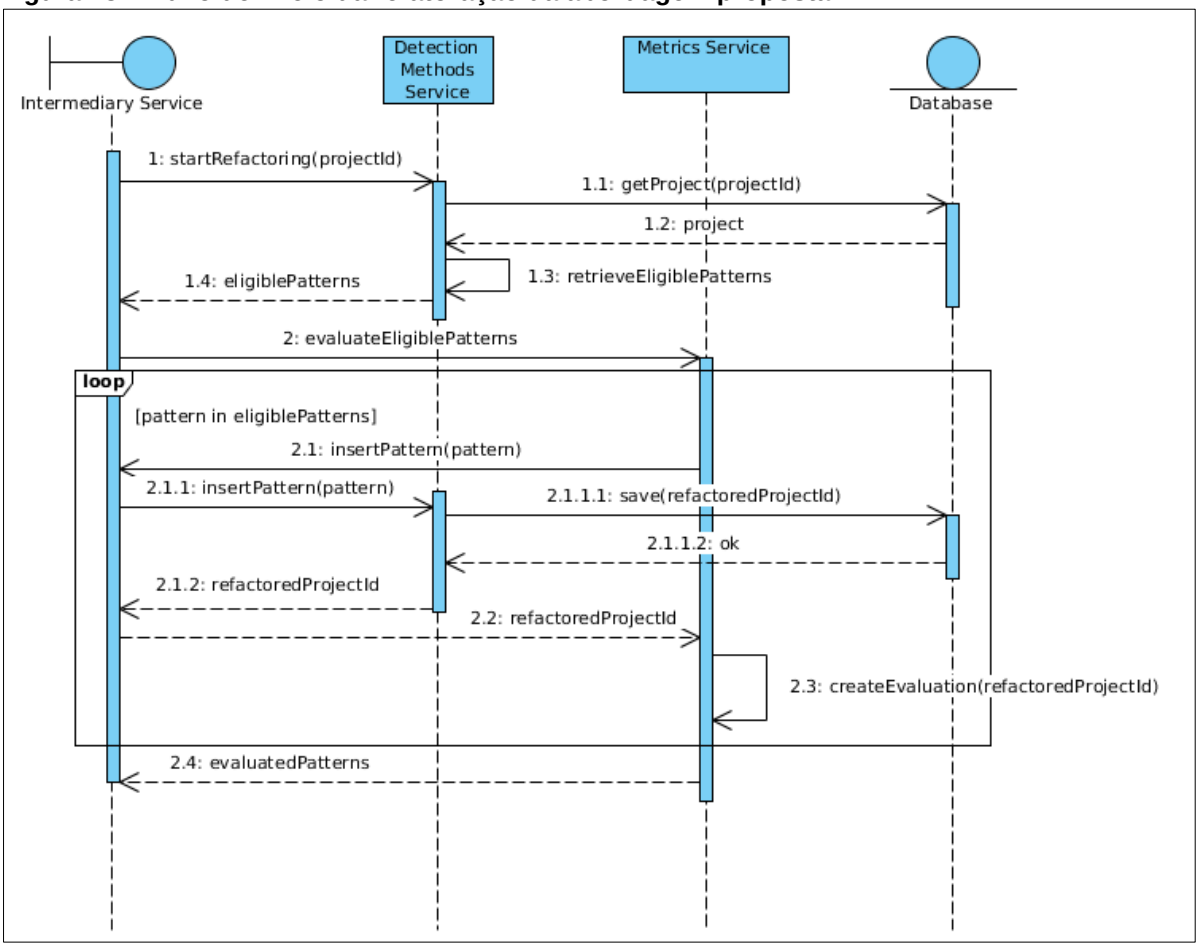
4.3 PROCESSO DE FUNCIONAMENTO DOS MÓDULOS

Cada módulo desempenha seu papel de forma a centralizar a utilização de métodos de refatoração voltados à aplicação de padrões de projeto, prover a análise dos métodos de refatoração no código-fonte antes mesmo de sua aplicação definitiva, e finalmente, refatorar o código-fonte de acordo com a decisão tomada pelo usuário.

Inicialmente, é feito um descritivo de fluxo geral da abordagem proposta, para que nas seções 4.3.1, 4.3.2, 4.3.3 e 4.3.4 seja apronfundado o papel de cada módulo da abordagem. O único fluxo não descrito é o de “Solicitar o registro de código-fonte’ (Seção 4.3.2), pois é considerado um processo simples de inserção de projeto no núcleo da abordagem, detalhado nas seções 4.3.1 e 4.3.2.

Todos os processos do núcleo da abordagem são iniciados a partir de uma requisição do *Client App e* os fluxos das Figuras 13 e 14 são mostrados considerando o momento em que o *Client App* já fez as requisições das funcionalidades “Solicitar a avaliação do código-fonte” e “Solicitar aplicação de padrões de projeto” (Seção 4.3.2) ao *Intermediary Service*. Cabe ao *Intermedary Service* repassar essas solicitações aos demais módulos do núcleo da abordagem. Para tornar o processo de descrição dos fluxos mais simples, foram também adicionados índices de numeração que correspondem a mensagem que está sendo detalhada. Por exemplo, quando for destacado o valor 1.4 da Figura 13, sabe-se que está se refere a uma mensagem de retorno que contém os padrões elegíveis a refatoração.

Figura 13 - Fluxo de início da refatoração da abordagem proposta

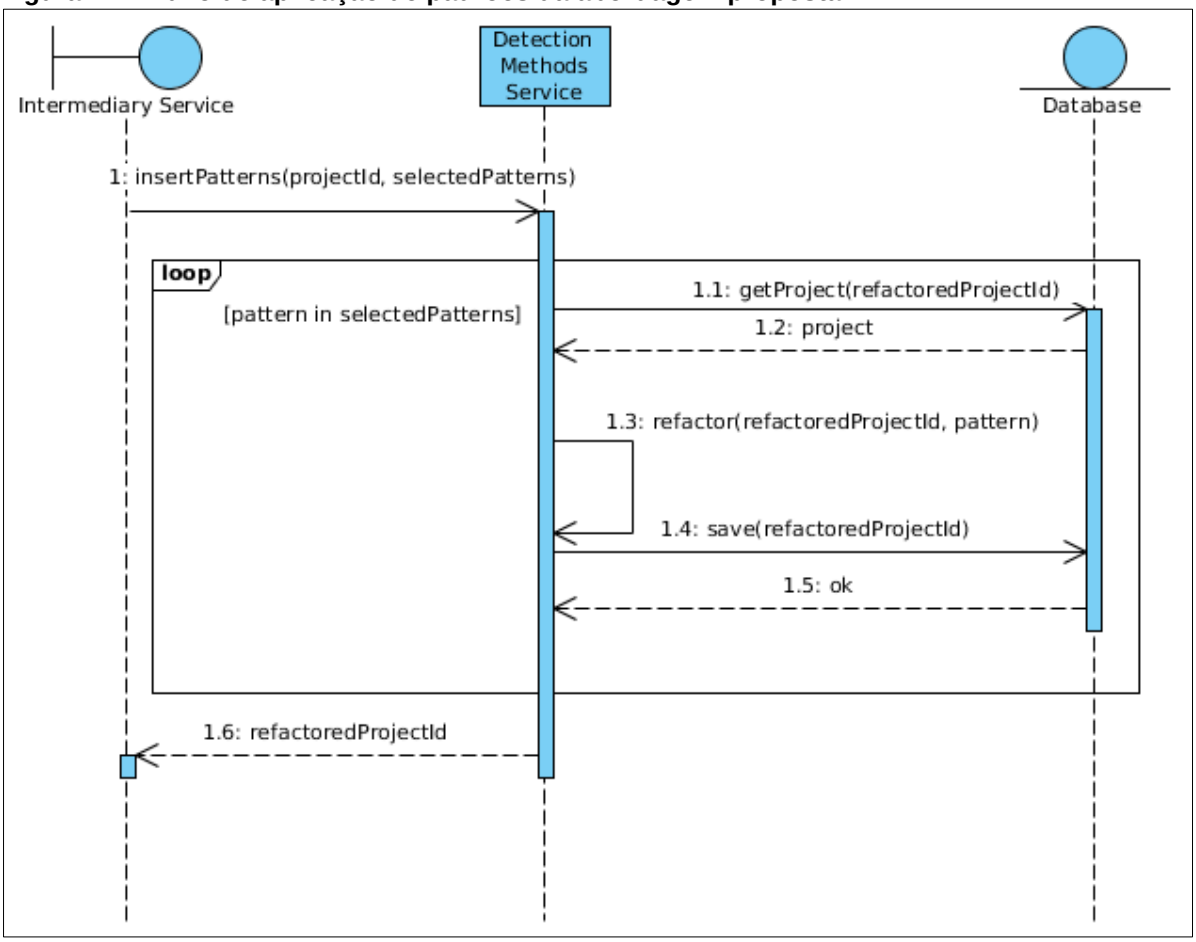


Fonte: Autoria Própria

Ao se observar os diagramas de sequência (Figura 13), nota-se que ambas as requisições esperam receber o identificador do projeto (*projectId*) como parâmetro, isto porque antes mesmo dessas duas operações é necessário que seja feito o registro do projeto (Seção 4.3.1).

O processo de obtenção de candidatos a refatoração da abordagem proposta (Figura 13) é iniciado a partir de uma chamada do *Client App* ao *Intermediary Service*, o qual recebe a solicitação e a repassa ao *Detection Methods Service* (1), que busca o projeto no banco de dados (1.1 e 1.2) e obtém os candidatos a refatoração (1.3, detalhado na Seção 4.3.3), os quais são retornados ao *Intermediary Service* (1.4).

Figura 14 - Fluxo de aplicação de padrões da abordagem proposta



Fonte: Autoria Própria

Uma vez que o *Intermediary Service* recebeu o retorno da chamada ao *Detection Methods Service* (1.4), os candidatos são enviados ao *Metrics Service* (2) para avaliação de métricas.

O *Metrics Service* faz execuções individuais de avaliação para cada candidato a refatoração, enviando-os para que refatorem uma cópia do código-fonte original (2.1 a 2.2) e possa extrair a avaliação individual do candidato em questão (2.2 e 2.3). Logo após, os candidatos são enviados ao *Intermediary Service* (e posteriormente ao *Client App*) como resposta a avalição feita pelo núcleo da abordagem.

O segundo fluxo de interação com o núcleo da abordagem (Figura 14) considera que o usuário já selecionou quais candidatos a refatoração devem ser efetivamente aplicados nos código-fonte e faz uma segunda requisição ao núcleo da abordagem (por meio do *Client App*) solicitando a aplicação dos candidatos selecionados, reenviando-os juntamente com o *projectId*.

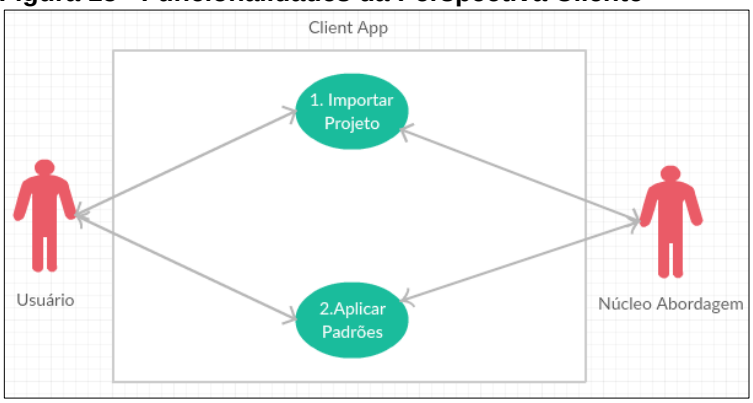
A requisição do *Client App* é recebida pelo *Intermediary Service*, o qual é responsável por aplicar todos os candidatos selecionados no projeto. Os candidatos são enviados ao *Detection Methods Service* (1) para que sequencialmente seja buscado no banco de dados (1.1 e 1.2) a última versão do código-fonte refatorado (caso não exista, o original é obtido) e verifique qual método de refatoração corresponde ao candidato em questão e aplique o padrão desejado (1.3). No fim de cada refatoração, o código-fonte refatorado é salvo no banco de dados (1.4) e o que se tem de resultado da aplicação (1.5) é um identificador do projeto refatorado (*refactoredProjectId*). A versão final de todas as aplicações de padrões é finalmente enviada ao *Intermediary Service* (1.6), o qual é posteriormente a repassado ao *Cliente App*.

As seções a seguir têm por objetivo detalhar os processos internos aos módulos do núcleo da abordagem, bem como apresentar os pontos de expansão previstos para mesma.

4.3.1 Client App

Este módulo abrange duas responsabilidades: interações com o usuário e comunicação com *Intermediary Service*. Dentre as interações com o usuário têm-se a apresentação de informações providas pelo núcleo da abordagem (possíveis padrões para inserção e resultado da análise de métricas). As interações do usuário com o núcleo são intermediadas pelo *Client App*. A Figura 15 apresenta as funções disponíveis ao usuário que posteriormente são repassadas ao núcleo da abordagem.

Figura 15 - Funcionalidades da Perspectiva Cliente



Fonte: Autoria Própria

A funcionalidade de Importar Projeto é acionada pelo usuário e faz com que o *Client App* solicite duas operações fornecidas pelo núcleo. Na primeira operação, o *Client App* é responsável por compactar um projeto e enviá-lo e seu retorno é o *id* do projeto que foi registrado. Na segunda operação, após o registro do projeto, o *Client App* executa uma nova solicitação, agora efetivamente requerendo uma avaliação do projeto. A avaliação é realizada mediante o identificador do projeto e ao final desta solicitação o *Client App* recebe todos os padrões de projeto que podem ser aplicados no código-fonte do projeto, bem como a avaliação dos atributos de qualidade resultantes da avaliação.

Ao receber todos os candidatos a refatoração do núcleo da abordagem, o *Client App* apresenta as possibilidades ao usuário, que por sua vez, tem que escolher quais padrões deseja aplicar em seu código-fonte. O usuário ao selecionar os padrões a serem aplicados, tem a possibilidade de acionar a funcionalidade de Aplicar Padrões. Por exemplo, se o núcleo da abordagem verifica que os padrões *Singleton*, *Factory* e *Template Method* são possíveis de aplicação no código-fonte; o usuário escolhe a opção de aplicar o padrão *Singleton*, então somente este será aplicado.

Como auxílio na escolha de quais padrões podem ser aplicados, os atributos de qualidade da avaliação, descritos no fluxo do *Metrics Service* da Seção 4.3.4, são apresentadas ao usuário para que este possa avaliar os efeitos que o padrão escolhido pode trazer ao código-fonte.

Caso o usuário decida por não aplicar os padrões, o processo de refatoração pode ser cancelado. Com isso, o código-fonte é mantido no núcleo da abordagem e não é disponibilizado ao usuário. Se o usuário desejar aplicar os padrões, o *Client App* recebe a versão refatorada do código-fonte (compactado) e permite que o usuário selecione um diretório do seu sistema de arquivos para salvar esta nova versão.

4.3.2 Intermediary Service

Os módulos não se comunicam diretamente um com o outro, todos fazem ou recebem solicitações ao *Intermediary Service* que tem por responsabilidade realizar as interações entre os módulos e manter os códigos-fonte dos projetos a serem refatorados.

Para cada tipo de solicitação recebida pelo *Intermediary Service*, um determinado tipo de *Service* é utilizado para processamento. As funcionalidades providas por este módulo e os respectivos *Services* a serem chamados são:

- Solicitar o registro de código-fonte: registra o código-fonte no banco de dados para que os diversos processos envolvendo a avaliação ou aplicação de padrões obtenha o código-fonte posteriormente.
- Solicitar a avaliação do código-fonte: obtém o primeiro *Detection Method Service* disponível na fila, então, a solicitação de verificação de padrões é repassada a este *Service* ativo para verificar os padrões a serem aplicados (juntamente com o *id* do projeto).

  O *Detection Methods Service* obtém os possíveis padrões a serem aplicados e os repassa ao *Intemediary Service*. A partir destes padrões, são feitas as solicitações de aplicações individuais de cada candidato a refatoração. Para cada candidato são solicitadas refatorações de cada padrão a um membro disponível na fila dos *Detection Methods Services*.

  Cada candidato terá um código-fonte refatorado correspondente a ele, com isso, é solicitada uma avaliação do candidato enviando a identificação do projeto original e a identificação do projeto refatorado. Os detalhes da avalição estão descritos na seção do *Metrics Service* (4.3.4).

  Ao final de todo este processo, cada um dos padrões terá um conjunto de dados referentes aos aprimoramentos ou declínios causados por ele ao código-fonte. É possível que individualmente se tenha noções das vantagens ou desvantagens da aplicação de determinado padrão.
- Solicitar aplicação de padrões de projeto: uma vez recebido os padrões escolhidos pelo usuário, o *Intermediary Service* aplica-os individualmente ao código-fonte original e o código-fonte refatorado é compactado e enviado como resposta a solicitação.

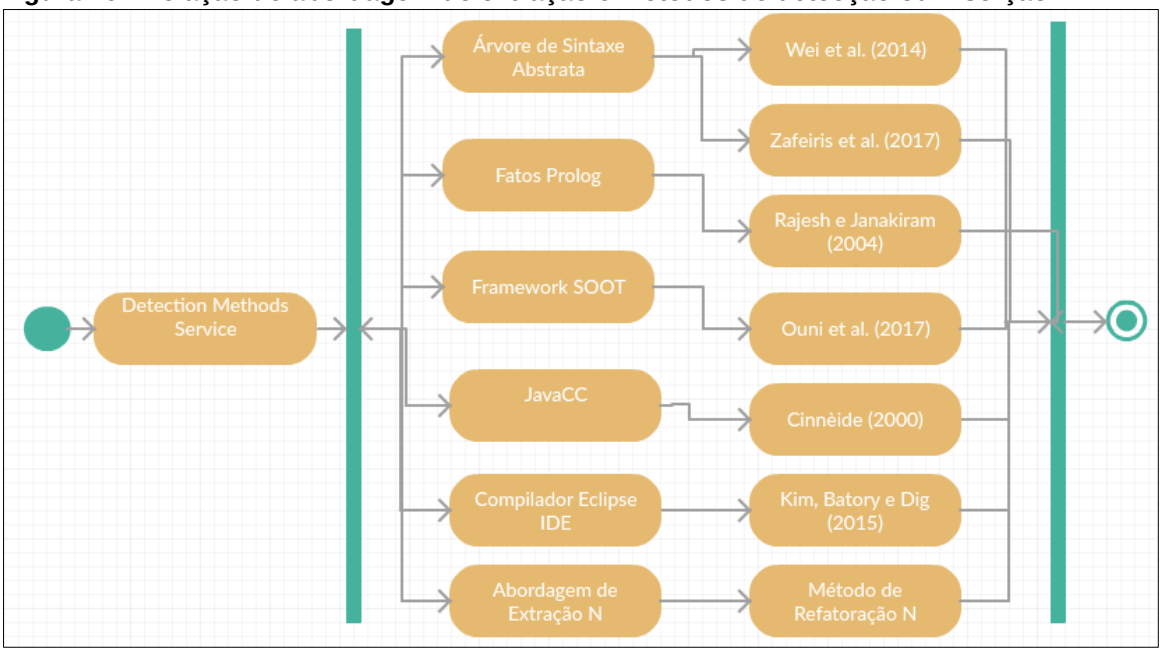
4.3.3 Detection Methods Service

O *Service* verifica quais padrões podem ser aplicados no código-fonte, por isto recebe a solicitação e submete o projeto a avaliações dos métodos de refatoração presentes da literatura.

Os métodos de refatoração encontrados possuem maneiras distintas de detectar pontos de inserção de padrões de projeto em determinado código-fonte. Zafeiris et at. (2017) por exemplo, transforma o código-fonte em Árvores de Sintaxe Abstrata (JONES, 2003) para verificar a possibilidade de inserção do padrão *Template Method*. Rajesh e Janakiram (2004) transforma o código-fonte em Fatos Prolog (CLOCKSIN; MELISH, 2003), o que os permite fazer buscas dos dados por meio de determinadas Regras Prolog. Este tipo de percepção de similaridade em abordagens de extração de informações do código-fonte foi melhor detalhada no capítulo de mapeamento sistemático.

Os métodos para obtenção de candidados são separados em módulos internos a este *Service* e as abordagens de extração de dados abordadas no parágrafo anterior e sua relação com os métodos de refatoração estão exibidos na Figura 16.

Figura 16 - Relação de abordagem de extração e métodos de detecção ou inserção



Fonte: Autoria Própria

A Figura 16 apresenta que a detecção é fragmentada em pequenas formas de extração de dados do código-fonte (abordagens de extração). Considerando a possibilidade de um método *M1* se utilizar da abordagem de extração *AE1* e outro método M2 se utilizar desta mesma abordagem, então, as abordagens tiveram sua execução centralizada e foram vinculadas de acordo com os métodos que as utilizam.

Para que a abstração proposta fosse concretizada, foi criado um *DetectionMethodsManagerImpl* que tem por objetivo abranger as formas de extração de dados e métodos de refatoração e detecção de candidatos a refatoração.

Uma vez que uma solicitação de detecção de candidatos seja feita, a requisição é repassada a todos os *DataExtractionFork* (Figura 17). Isto significa que para cada forma de extração deve existir uma classe concreta que implemente esta interface. No caso da abordagem proposta foi construído um *fork* para implementação de Árvores de Sintaxe Abstrata (JONES, 2003).

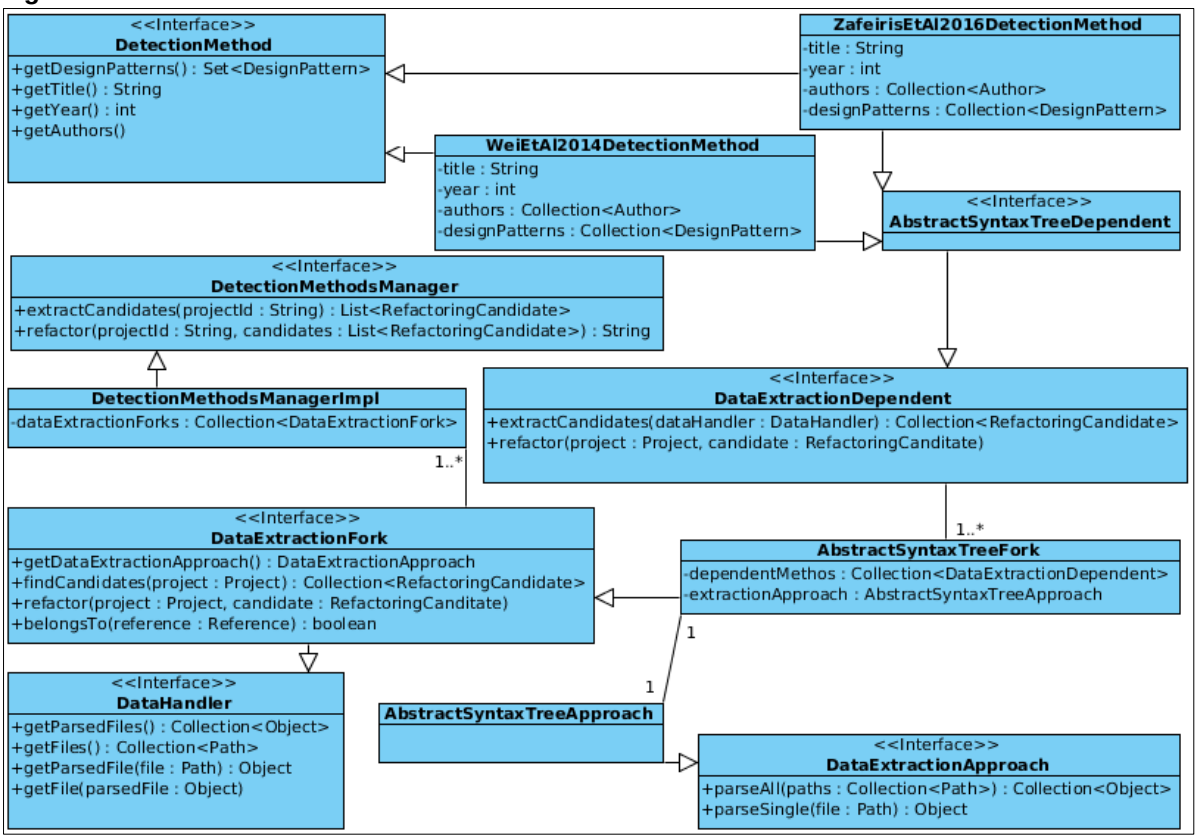
Cada *DataExtractionFork* é responsável por transformar os arquivos (classe não compilada .java) correspondentes as classes .java no meio de extração e por conter todos os métodos de refatoração dependentes de uma única forma de extração. No caso de Árvores de Sintaxe Abstrata (JONES, 2003), foi criado o *AbstractSyntaxTreeFork* que tranforma um arquivo em uma *CompilationUnit*, a qual é uma implementação de árvore de sintaxe abstrata provida pela fundação eclipse (Eclipse JDT, 2018).

O primeiro ponto de extensão previsto na abordagem proposta é a possibilidade de criar diversas classes filha de *DataExtractionFork* (Figura 17), com base na Figura 16 por exemplo, pode-se considerar a possiblidade de implementação de abordagens de extração específicas como: *PrologFork* (Fatos Prolog), *SootFork* (Framework SOOT), *JavaCCFork* (Java CC), *EclipseFork* (Compilador Eclipse IDE), etc.

Outro ponto de extensão são os dependentes do método de extração (*DataExtractionDependent*), os quais se utilizam de determinado tipo de extração. Dentro da abordagem proposta, foram aplicados dois métodos que se utilizam de árvores de sintaxe abstrata em sua execução, Wei *et al*. (2014) e Zafeiris *et al*. (2017). Para isto, é criado um tipo de dependente específico a abordagem de extração de dados (*AbstractSintaxTreeDependent*), o qual serve de identificação para os métodos de refatoração que dependem deste tipo de extração.

O *DetectionMethodsManagerImpl* contém diversas formas de extração dados do código fonte (*DataExtractionFork)* e cada uma das formas implementa os métodos que são dependentes de sua atuação na manipulação das classes .java (*DataExtractionDependent*). Os *DataExtractionDependent* possuem a implementação de refatoração e verificação de candidatos efetivamente.

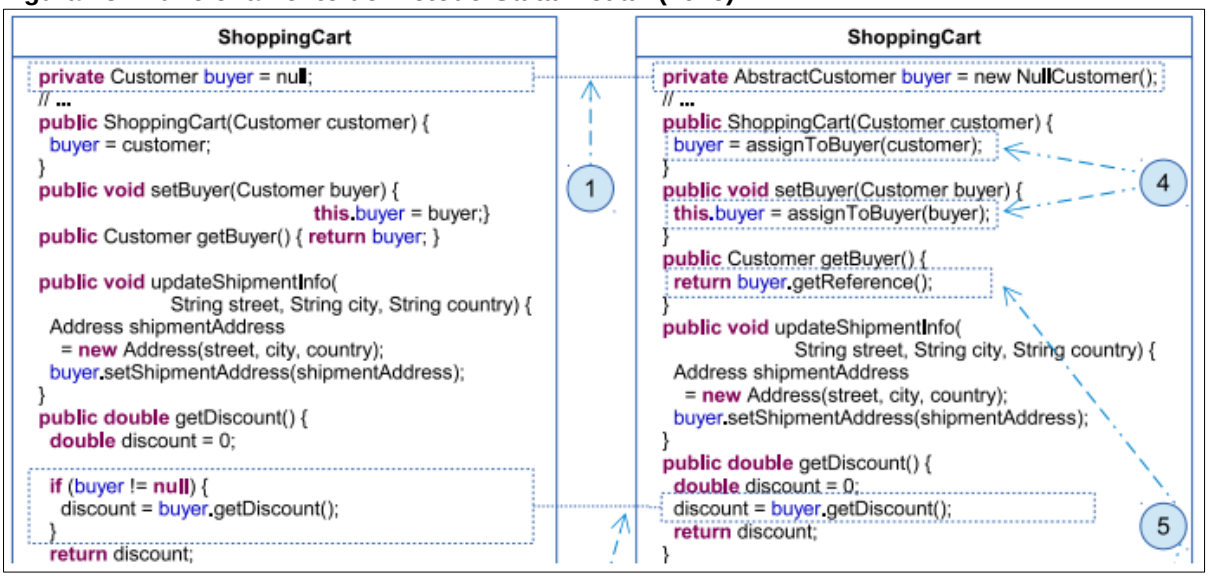
Figura 17 - Modelo de classes do *Detection Methods Service*



Fonte: Autoria Própria

A Figura 18 exemplifica a ação do *Service* a partir do método de Gaitani *et al*. (2015), um dos possíveis métodos implementados por este módulo. As partes destacadas a esquerda na figura representam os pontos passíveis de alteração. O objeto *buyer*, ao invés de ser instanciado nulo pode ter uma instância padrão (*Null Object*) e as verificações de nulo deste mesmo objeto já não precisam mais ser utilizadas. Essas situações mostram a necessidade do método de refatoração para inserção do padrão *Null Object*.

Figura 18 - Funcionamento do método *Gaitani et al*. (2015)

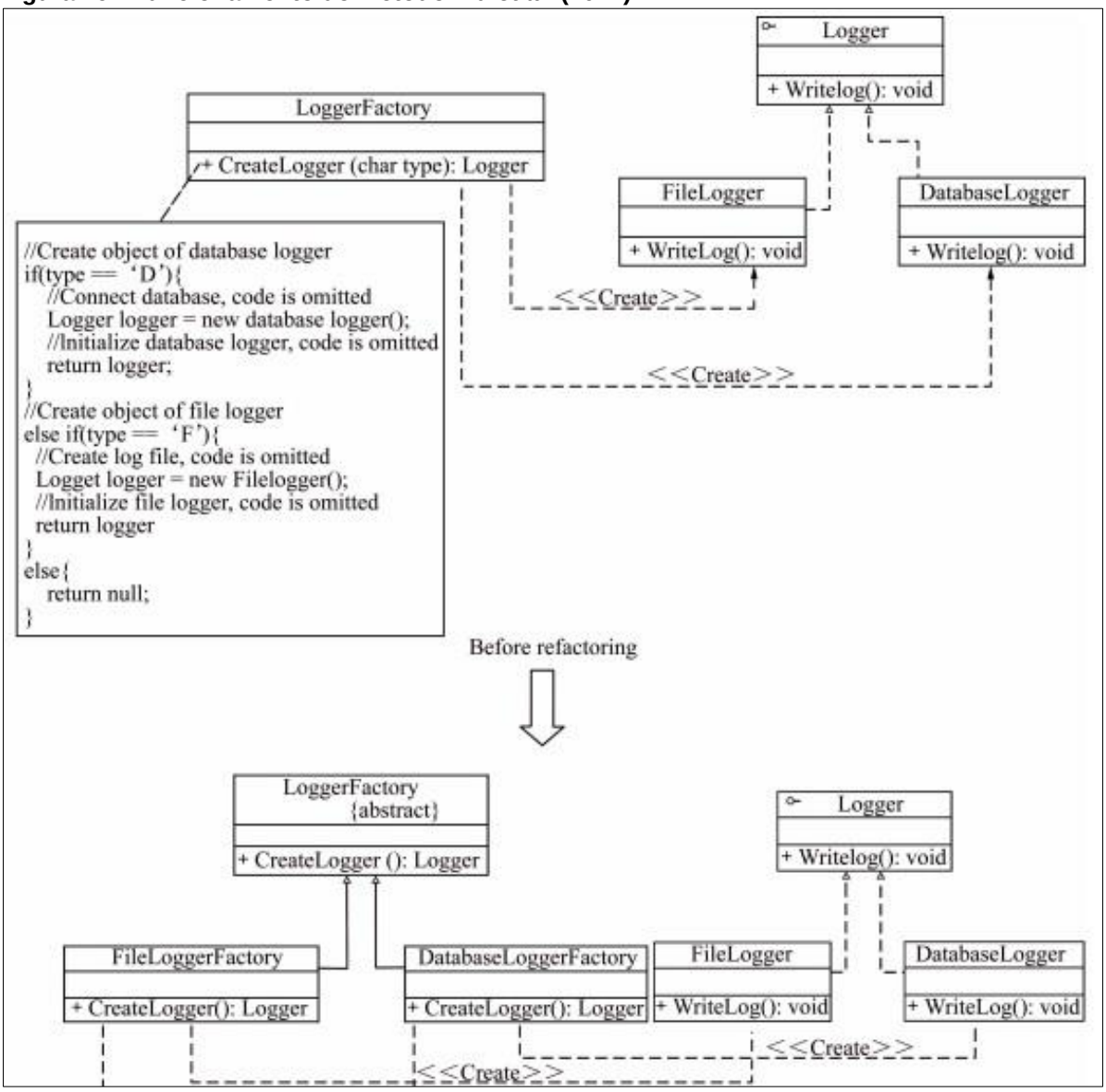


Fonte: Gaitani *et at*. (2015)

Um outro exemplo de detecção de candidato a refatoração previsto para este *Service*, é o padrão *Factory* (Figura 19) proposto por Liu *et al*. (2014). Esta figura exibe uma expressão condicional complexa, avaliando os possíveis valores da variável *type* da classe *LoggerFactory*.

A partir da refatoração proposta por Liu *et al*. (2014), a expressão condicional pode ser substituída pelo padrão *Factory Method*, eliminando a necessidade das verificações feitas nos valores da variável *type*, criando-se uma hierarquia de classes *Factory* que gera diversas instâncias de uma mesma classe *Logger*.

Figura 19 - Funcionamento do Método Liu *et al*. (2014)



Fonte: Liu *et at*. (2014)

Considerando que o método de Gaitani *et al*. (2015) possibilita a inserção do padrão *Null Object* e o de Liu *et al*. (2014) o padrão *Factory Method*, a ideia é que mais métodos de verificação de candidatos possam ser incorporados na abordagem proposta. Por isto, cada método tem uma entrada comum (o *id* do projeto a ser analisado), uma abordagem de extração de dados (se for possível desacopla-la) e uma saída comum (um candidato à refatoração).

O *Detection Methods Servi*ce recebe os retornos de todos os métodos de refatoração e agrupa-os em um resultado com todos os métodos verificados. O candidato à refatoração é representado na Figura 20, possui os campos *id* (identificador único do candidato), referência (método utilizado para detecção do ponto

de inserção), pacote (pacote da classe java), classe (nome da classe java), padrão de projeto (sugestão de padrão a ser aplicado) e avaliação (descrita em detalhes na Seção do módulo *Métrics Service*).

Figura 20 - *Detection Methods Service* - Estrutura de um candidato a refatoração

```
{
  "id": "a00b1",
  "reference": {
    "title": "Article Title",
    "year": 1999,
    "authors": [
      "First Author",
      "Second Author"
    ]
  },
  "pkg": "br.com.my.package.example",
  "className": "MyClass",
  "eligiblePattern": "FACTORY_METHOD",
  "evaluation": null
}
```

Fonte: Autoria Própria

Além de fazer a detecção de pontos de inserção de padrões, o *Service* é responsável por refatorar determinado código com padrões solicitados os quais foram enviados pelo *Intermediary Service*.

Em seu funcionamento interno, o *Detection Methods Serv*ice possui a correlação entre métodos de detecção de pontos de inserção de padrões e qual o seu método correlato para refatoração do código-fonte. O trabalho de Jeon, Lee e Bae (2002), por exemplo, é responsável por detectar pontos de inserção de padrões de projeto por meio de Fatos Prolog, porém, quando este obtém o ponto de inserção, o método utilizado para a refatoração é o de Cinnèide (2001).

Finalmente, em um processo similar a Figura 16, o código-fonte é refatorado aplicando os novos padrões sugeridos um a um. O código-fonte resultante da aplicação de todos os padrões é comprimido e inserido no banco de dados, o *id* desta refatoração é enviado como resposta a operação de refatoração do projeto.

4.3.4 Metrics Service

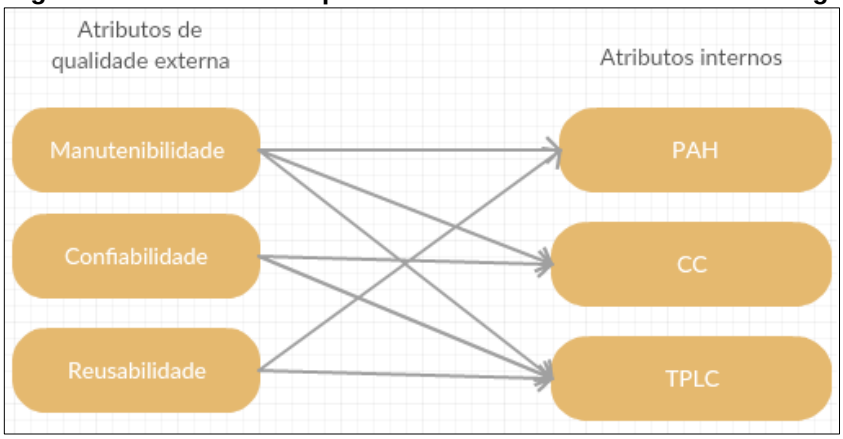
O módulo *Metrics Service* foi inserido para que o usuário receba uma pré-avaliação do candidato a refatoração, antes mesmo que ele seja aplicado no código-fonte original. A avaliação de melhoria é executada com os candidatos à refatoração, os quais foram obtidos na operação do *Detection Methods Service* e se dá em termos de métricas de software e atributos de qualidade.

A refatoração do código é iniciada uma vez que a operação de avaliação é solicitada pelo *Intermediary Service*. Este processo busca o código-fonte original do projeto (através de seu *id*) e o avalia extraindo métricas de classe, herança, método, sistema e acoplamento. Esse conjunto de métricas extraídas do projeto original é denominado *MPO* (métricas do projeto original), que serão utilizadas posteriormente.

Com o identificador do projeto refatorado (enviado juntamente na requisição de avaliação), o código-fonte refatorado é submetido a mesma avaliação de medição obtendo-se as *MPR* (métricas do projeto refatorado).

Para a composição da avaliação, será utilizada a associação proposta por Sommervile (2011). Para esta abordagem foram utilizadas 3 dos 4 (quatro) atributos de qualidade, removendo-se somente o de usabilidade. E dentre as métricas descritas na subseção Métricas (2.2), as métricas de profundidade da árvore de herança (PAH), complexidade ciclomática (CC) e tamanho do programa em linhas de código (TPLC) foram utilizadas. A associação criada para avaliação da abordagem proposta é apresentada na Figura 21.

Figura 21 - Atributos de qualidade e métricas usados na abordagem



Fonte: Autoria Própria

Para que seja calculado o valor de avaliação para os atributos de qualidade (AAQ), a Fórmula 1 foi elaborada, onde o valor de avaliação do atributo é dado pela

média aritmética da porcentagem de mudança de cada métrica que têm relacionamento com ele. No caso de Reusabilidade (Figura 21), este atributo é associado as métricas de Profundidade da Árvore de Herança (PAH) e Tamanho do Projeto em Linhas de Código (TPLC). Logo após, é necessário descobrir a porcentagem de mudança para estas duas métricas.

$$AAQ = (PM^0 + PM^1 \dots PM^n)/n \quad (1)$$

A porcentagem de mudança (*PM*) da métrica sob avaliação representa o quanto a métrica variou (em termos percentuais) de quando foi avaliada no código-fonte original para o código-fonte refatorado. As métricas do projeto original (*MPO*) e do projeto refatorado (*MPR*) são entradas na Fórmula 2 para se gerar uma porcentagem de variação das métricas.

$$PM = \left(\frac{MPR * 100}{MPO}\right) - 100 \quad (2)$$

Ao calcular uma métrica TPLC, em um cenário onde sua *MPO* tem valor 100 e sua tem *MPR* valor 120, têm-se um acréscrimo de 20% na porcentagem de TPLC, para compor o cálculo da Fórmula 2.

É possível que uma determinada métrica tenha seu valor inversamente proporcional ao atributo de qualidade sendo calculado, como por exemplo, a métrica de Complexidade Ciclomática. A medidade que o valor desta métrica aumenta, ela faz com que a Manutenibilidade diminua.

No caso em que as métricas são inversamente proporcionais aos seus atributos de qualidade correlatos, é feito o cálculo da *PM* inversa. Ou seja, *MPR* será substituído pelo valor de *MPO* na Fórmula 2 e o valor de *MPO* será substituído pelo valor de *MPR* na Fórmula 2. O Quadro 5 coloca em evidência as métricas que são diretamente (D) ou inversamente (I) proporcionais ao atributo de qualidade sendo avaliado.

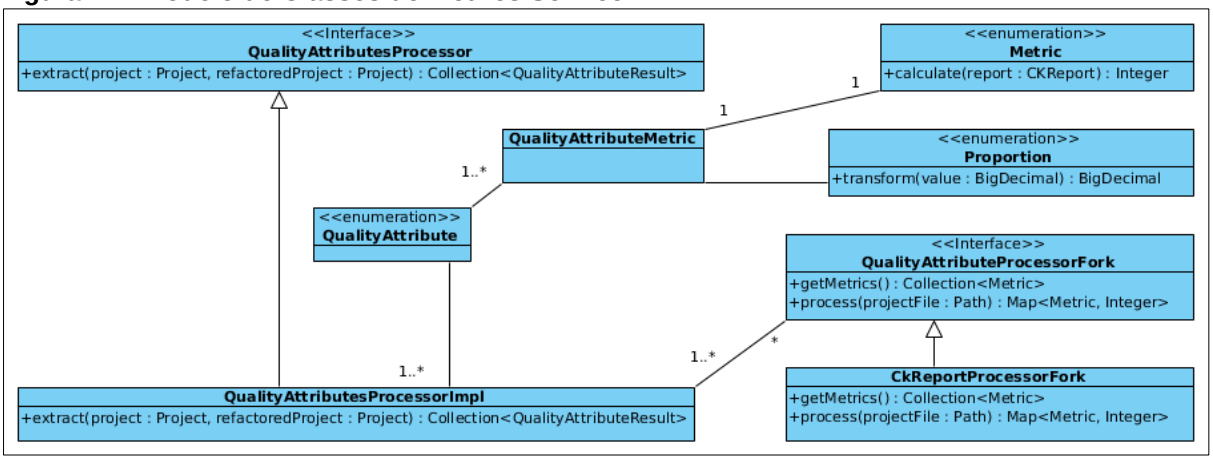
Quadro 5 - Correlação valores de Atributos de Qualidade e Métricas

Atributos de Qualidade	Métricas				
	Profundidade da Árvore de Herança	Complexidade Ciclomática	Tamanho do Projeto em Linhas de Código	Número de Mensagens de Erro	Extensão do Manual do Usuário
Manutenibilidade	D	I	I		
Confiabilidade		I	I	D	
Reusabilidade	D		D		

Fonte: Autoria Própria

Além dos atributos de qualidade implementados (manutenibilidade, confiabilidade e reusabilidade), a abordagem permite a extensão desta avaliação de atributos, ou seja, a implementação de avaliação de mais métricas (Figura 22).

Figura 22 - Modelo de Classes do *Metrics Service*



Fonte: Autoria Própria

A estrutura interna de avaliação do código-fonte dos projetos prevê um processador dos atributos de qualidade (*QualityAttributesProcessorImpl*) que têm tanto os atributos de qualidade a serem avaliados (*QualityAttribute*) quanto os *QualityAttributeProcessorFork*. Para adicionar uma nova avaliação no *Metrics Service*, é necessário que o novo atributo de qualidade seja adicionado e sejam acrescentadas a ele as métricas que serão utilizadas para avaliação.

A correlação entre métricas e atributos de qualidade é feita adicionando uma nova *QualityAttributeMetric* ao *QualityAttribute*. Este *QualityAttributeMetric* contém a métrica (*Metric*) relacionada ao atributo de qualidade quanto a *Proportion.* Por meio da *Proportion* é definido se a métrica é diretamente ou inversamente proporcional ao atributo de qualidade em questão.

Caso a métrica não esteja implementada na abordagem, é necessário que seja adicionado um novo item ao enum *Metric*.

O *QualityAttributesProcessorImpl* faz a avaliação das métricas por meio das classes concretas que são criadas a partir da interface *QualityAttributeProcessorFork*, a qual recebe como entrada o caminho do projeto sob avaliação e retorna o conjunto de métricas processadas por este *fork* juntamente com o valor extraído para cada métrica.

Caso novas métricas sejam adicionadas, é preciso ser criado uma nova classe concreta a partir do *QualityAttributeProcessorFork* que implemente a nova extração de valores, como é o exemplo da classe *CKReportProcessorFork*. Este nome foi dado devido ao projeto *open-source* utilizado para processamento das métricas da abordagem, mais detalhes estão presentes na seção sobre implementação da ferramenta (Seção 5.1).

No exemplo da Figura 23 têm-se o retorno esperado deste *Service*, contendo o resultado da avaliação da aplicação de um projeto. Conforme previsto, cada atributo de qualidade (Figura 21) é retornado com sua porcentagem de melhora ou não em relação ao código avaliado.

Figura 23 - Resultado fornecido pelo *Metrics Service*


```
[
  {
    "qualityAttributeName": "MAINTAINABILITY",
    "changePercentage": "40%"
  },
  {
    "qualityAttributeName": "RELIABILITY",
    "changePercentage": "20%"
  },
  {
    "qualityAttributeName": "REUSABILITY",
    "changePercentage": "25%"
  },
  {
    "qualityAttributeName": "USABILITY",
    "changePercentage": "-5%"
  }
]
```


Fonte: Autoria Própria

Portanto, *Metrics Service* obtém os resultados de avaliações do candidato e agrupá-os a fim de enviar o resultado ao *Intermediary Service* que repassa os resultados ao *Client App* para escolha do usuário de quais padrões deseja aplicar em seu projeto.

4.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou uma abordagem para detecção de pontos de inserção e aplicação de padrões de projeto em código-fonte visando cobrir aspectos de melhoramento em relação aos trabalhos identificados no mapeamento sistemático.

Dentre as melhorias propostas têm-se: contemplar vários métodos de refatoração; avaliar antecipadamente os benefícios da aplicação do padrão usando como critério atributos de qualidade e permitir ao usuário escolher ou não efetivar a aplicação do padrão.

A abordagem proposta possui módulos internos (*Metrics Service* e *Detection Methods Service*), um intermediário (*Intermediary Service*) e um ou mais aplicativo(s) cliente (*Client App*) e sua avaliação é apresentada no próximo capítulo.

5 RESULTADOS

Durante a análise do mapeamento sistemático dos trabalhos relacionados, observou-se que um dos testes de avaliação das ferramentas destinadas à refatoração de software baseada em padrões ocorre principalmente com a verificação de candidatos à refatoração extraídos de projetos de terceiros.

Outro meio de avaliação é a criação de um protótipo que implementa as funcionalidades da refatoração, não considera códigos-fonte de outros pesquisadores e aplica os padrões em um cenário simples, com um número reduzido de classes, contendo muitas vezes uma classe. Isso para que, em um cenário pequeno e isolado, se tenha mais controle do funcionamento do protótipo para realizar ajustes, se houver.

A avaliação da abordagem proposta foi realizada conforme as duas formas citadas anteriormente. A Seção 5.1 descreve o processo de criação da ferramenta, apresenta a definição de requisitos básicos da abordagem, telas principais de interação com o usuário e define as tecnologias usadas na implementação. A Seção 5.2 relata os resultados da abordagem em termos de projetos avaliados e eficácia da abordagem; bem como sua comparação com o cenário atual de refatorações na literatura. A Seção 5.3 descreve as considerações finais deste capítulo.

5.1 IMPLEMENTAÇÃO DA ABORDAGEM PROPOSTA

O processo de refatoração baseado em padrões de projeto é o elemento importante para qualquer abordagem. Por isto, foi desenvolvido inicialmente um protótipo que abrangesse a refatoração com base nos métodos de Wei *et al*. (2014) e Zafeiris *et al*. (2017). O método de Wei *et al*. (2014) apresenta refatoração para dois padrões, *Factory Method* e *Strategy*; já o de Zafeiris *et al*. (2017) contempla refatorações voltadas para o padrão *Template Method*.

Como não foi possível obter as implementações dos métodos originais dos estudos de Wei *at al*. (2014) e Zafeiris *et al*. (2017), foi necessário desenvolver os métodos contidos nestes estudos para posterior avaliação.

O protótipo inicial teve o intuito de aplicar os cenários teste que estavam nos artigos dos métodos que foram implementados (WEI *et al*., 2014; ZAFEIRIS *et al*., 2017), de forma a verificar se a construção relacionada aos métodos de refatoração

estava sendo codificada corretamente. O primeiro cenário de testes exibido na Figura 24 visa demonstrar um candidato à refatoração a partir do padrão *Strategy*, previsto por *Wei et al*. (2014).

Figura 24 - Cenário de testes para a classe MovieTicket

```
public class MovieTicket {

    private double price;

    public double calculate(char type) {

        if(type == 'S') {
            return price * 0.8;
        } else if(type == 'C') {
            return price-10;
        } else if(type == 'M') {
            return price * 0.5;
        }
        return -1;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public double getPrice() {
        return price;
    }

}
```

Fonte: Adaptado de *Wei et al*. (2014)

Neste cenário são verificadas as expressões condicionais em que dependendo do valor de uma variável, no caso *type*, a execução interna do método *calculate* que a implementa pode variar. O que se espera desta refatoração é a remoção da expressão condicional por completo.

Após a aplicação da refatoração Figura 25, é feita a criação da hierarquia de classes que implementam o padrão *Strategy*, em que cada classe filha do padrão tem em seu método o cálculo específico previsto para a variável *price* e que a chamada do *calculate* da classe *MovieTicket* é delegada ao novo padrão.

Figura 25 - Aplicação do padrão *Strategy* para o cenário de teste *MovieTicket*

```
public class MovieTicket {

    private double price;

    public double calculate(Strategy strategy) {
        return strategy.calculate(this.price);
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public double getPrice() {
        return price;
    }
}
```

```
public abstract class Strategy {

    public abstract double calculate(double price) {
    }
}

public class ConcreteStrategyC extends Strategy {

    public double calculate(double price) {
        return price - 10;
    }
}

public class ConcreteStrategyM extends Strategy {

    public double calculate(double price) {
        return price * 0.5;
    }
}

public class ConcreteStrategyS extends Strategy {

    public double calculate(double price) {
        return price * 0.8;
    }
}
```

Fonte: Autoria Própria

Outro cenário exemplo também descrito por Wei et al. (2014) é o de aplicação do padrão *Factory Method*, onde o cenário de testes para a classe *LoggerFactory* apresentado na Figura 26 também contém expressões condicionais que avaliam o valor de uma variável (*type*), porém neste caso, verifica situações em que são criados novos objetos a partir dessas expressões.

Figura 26 - Cenário de testes para a classe *LoggerFactory*

```
public class LoggerFactory {

    public Logger createLogger(char type) {

        if(type == 'D') {
            return new DatabaseLogger();
        } else if(type == 'F') {

            final Logger logger = new FileLogger();

            return logger;
        } else {
            return null;
        }

    }

}
```

Fonte: Adaptado de *Wei et al*. (2014)

O cenário da Figura 27 tem a estrutura da classe *LoggerFactory* mudada de forma drástica, onde a classe é transformada em uma classe abstrata e seu método *createLogger* é igualmente alterado para *abstract*. São criadas também, classes filha para a nova classe abstrata, cada uma condiz ao objeto sendo criado internamente ao seu método *createLogger*, ou seja, um *DatabaseLoggerFactory* para criação de um *DatabaseLogger* e um outro *FileLoggerFactory* para a criação de um *FileLogger*.

Figura 27 - Aplicação do padrão *Factory Method* para a classe *LoggerFactory*

```
public abstract class LoggerFactory {

    public abstract Logger createLogger();
}

public class DatabaseLoggerFactory extends LoggerFactory {

    public Logger createLogger() {
        return new DatabaseLogger();
    }
}

public class FileLoggerFactory extends LoggerFactory {

    public Logger createLogger() {
        final Logger logger = new FileLogger();
        return logger;
    }
}
```

Fonte: Autoria Própria

O último cenário de testes é o de aplicação do padrão *Template Method* exibido na Figura 28. Este cenário provido por Zafeiris et al. (2017) apresenta a situação em que determinada classe contém uma (única) chamada super em determinado método, possibilitando a aplicação de um *Template Method*. Dentre as várias verificações feitas para aplicação deste método, é previsto que os fragmentos de código que vem depois da chamada super não podem se utilizar de mais 1 (uma) variável local dos fragmentos de vem antes da chamada super.

Ao aplicar este padrão, a sua classe pai também deve ser alterada, no caso, *JICPPeer*. A partir desta classe, são definidos os métodos *hook* (*beforeActivate* e *afterActivate*) e *template* (*activate*) previstos para o padrão *Template Method*.

Figura 28 - Cenário de testes para a classe *JICPSPeer*

```
public class JICPSPeer extends JICPPeer{
    private MyLogger myLogger = null;
    private SSLContext ctx = null;

    @Override
    public TransportAddress activate(Listener i, String peerId, Profile p) throws ICPException {

        if(myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "About to Activate JICP peer.");
        }

        ctx = SSLHelper.createContext();

        setUserSSLAuth(SSLHelper.needAuth());

        if(myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "activate(0 context created ctx=" + ctx);
        }

        TransportAddress ta = super.activate(i, peerId, p);

        if(myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "JICP Secure Peer activated");
        }

        return ta;
    }

    private void setUserSSLAuth(boolean needAuth) {
    }

}
```

Fonte: Adaptado de *Zafeiris et al*. (2017)

Após a alteração da classe pai *JICPPeer*, a classe filha pode ser alterada. Nota-se que os fragmentos de código que estavam presentes antes da chamada *super* foram realocados para o método *beforeActivate* e os fragmentos que estavam dispostos após a chamada *super*, foram transferidos para o método *afterActivate*. A refatoração da classe *JICPPeer* é apresentada nas Figuras 28 e 29.

Figura 29 - Aplicação do padrão Template Method para a classe *JICPPeer* - Parte I

```
public class JICPPeer {

    private final int CONNECTION_TIMEOUT = 120;

    private final int POOL_SIZE = 5;

    private String myID = null;

    private int connectionTimeout = 20;

    private JICPClient client = null;

    private JICPServer server = null;

    private Ticket ticket = null;

    public final TransportAddress activate(Listener i, String peerId, Profile p) throws ICPException {
        beforeActivate(i, peerId, p);
        TransportAddress superReturnVar = doActivate(i, peerId, p);
        return afterActivate(i, peerId, p, superReturnVar);
    }

    private TransportAddress doActivate(Listener i, String peerId, Profile p) throws ICPException {
        myID = peerId;
        connectionTimeout = p.getparameter(CONNECTION_TIMEOUT, 0);
        client = new JICPClient(getProtocol(), getConnectionFactory(), POOL_SIZE);
        server = new JICPServer(p, this, i, getConnectionFactory(), POOL_SIZE);
        server.start();
        ticket = new Ticket(60000);
        ticket.start();
        TransportAddress localIta = getProtocol().buildAddress(server.getLocalHost(), server.getLocalPort(), null, null);
        return localIta;
    }

    protected void beforeActivate(Listener i, String peerId, Profile p) throws ICPException {
    }

    protected TransportAddress afterActivate(Listener i, String peerId, Profile p, TransportAddress ta) throws ICPException {
        return ta;
    }
```

Fonte: Autoria Própria

Figura 30 - Aplicação do padrão Template Method para a classe *JICPPeer* - Parte II

```
public class JICPSPeer extends JICPPeer {

    private MyLogger myLogger = null;

    private SSLContext ctx = null;

    protected void beforeActivate(Listener i, String peerId, Profile p) throws ICPException {
        if (myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "About to Activate JICP peer.");
        }
        ctx = SSLHelper.createContext();
        setUserSSLAuth(SSLHelper.needAuth());
        if (myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "activate(0 context created ctx=" + ctx);
        }
    }

    protected TransportAddress afterActivate(Listener i, String peerId, Profile p, TransportAddress ta) throws ICPException {
        if (myLogger.isLoggable(Logger.FINE)) {
            myLogger.log(Logger.FINE, "JICP Secure Peer activated");
        }
        return ta;
    }

    private void setUserSSLAuth(boolean needAuth) {
    }
}
```

Fonte: Autoria Própria

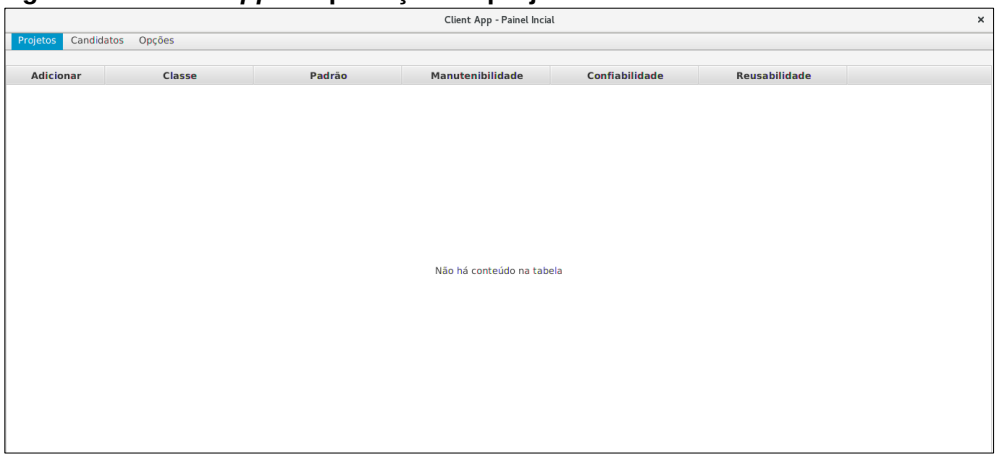
Após a execução com sucesso dos cenários teste preliminares, com as suas respectivas refatorações, o protótipo proposto para a abordagem foi transformado no *Detection Methods Service* e os demais módulos da abordagem foram criados, gerando assim o *RMT (Refactoring and Measurement Tool)*.

Além dos métodos de refatoração previstos como requisito básico para a abordagem, outro requisito foi a implementação de extratores de métricas. O extrator do valor das métricas foi encontrado na web e a partir de um projeto chamado CK (CK GITHUB, 2018), que continha as métricas PAH, CC, TPLC. Esse extrator foi adaptado

internamente ao *Metrics Service* para realizar as devidas avaliações dos projetos (Seção 4.3.4).

O *RMT* contém também o módulo externo (*Client App*), conforme definido no Capítulo 4. O *Client App* é usado como meio de interação com o usuário e apresenta as funções de acordo com as opções que estão visíveis na Figura 31: *Projetos/Importar*, *Candidatos/Aplicar* e *Outros/Limpar* (ver Seção 4.3.1).

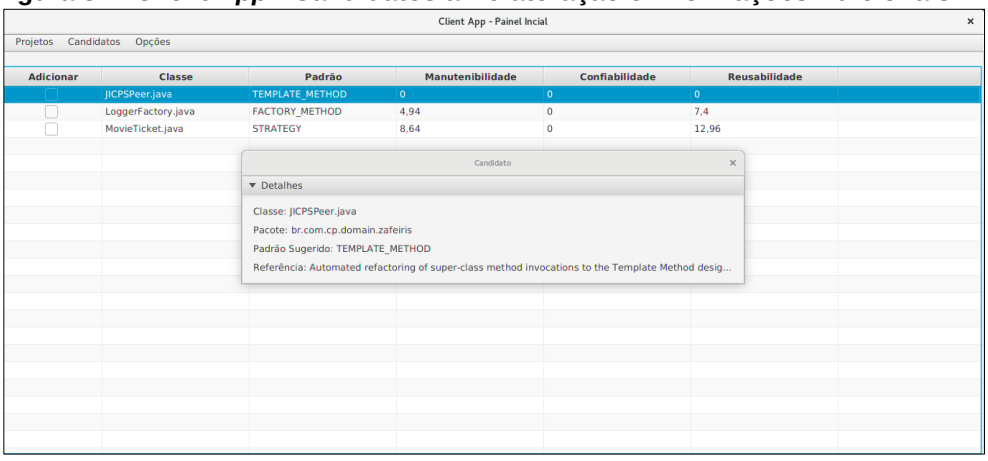
Figura 31 - *Client App* - Importação de projeto a ser refatorado



Fonte: Autoria Própria

Ao selecionar um projeto para importação, o projeto é submetido a avaliação e seu retorno é apresentado ao usuário, o qual pode também visualizar informações adicionais do candidado a refatoração tais como: nome da classe, pacote, entre outros (Figura 32).

Figura 32 - *Client App* - Candidatos a Refatoração e Informações Adicionais

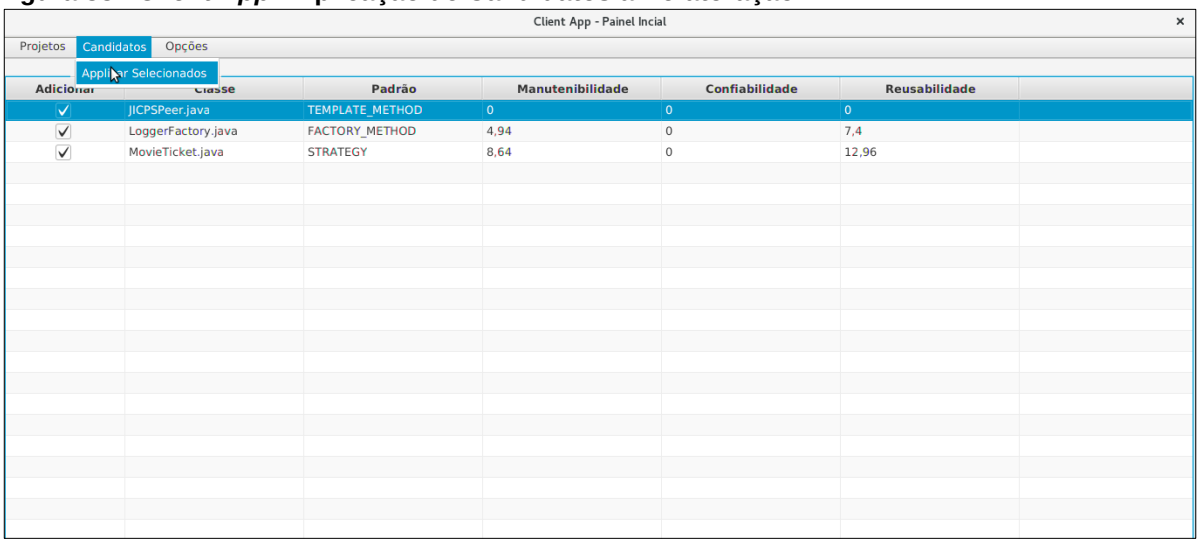


Fonte: Autoria Própria

O usuário pode escolher os candidatos a serem aplicados (Padrões/Aplicar), neste caso todos os canditos a refatoração selecionados são efetivamente aplicados

e uma nova versão do projeto é salva em um diretório definido pelo usuário (Figura 33).

Figura 33 - *Client App* - Aplicação de Candidatos a Refatoração



Fonte: Autoria Própria

O protótipo criado inicialmente, conforme relatado anteriormente, foi submetido aos cenários de testes que estavam nos artigos dos seus respectivos métodos (WEI et al., 2014; ZAFEIRIS et al., 2017), com o objetivo de avaliar seu retorno e aplicação dos padrões. Outros cenários de testes foram escolhidos para avaliação da ferramenta criada (*RMT*) e estão descritos na Seção 5.2.

5.1.1 Tecnologias utilizadas

Dentre os métodos de refatoração levantados, como Cinnèide (2001), Jullerat e Hirsbrunner (2007), Gaitani *et al*. (2015) e outros mencionados no Capítulo 3, se utilizam da linguagem de programação Java (JAVA, 2018). Por isto, esta foi utilizada na concepção da ferramenta (*RMT*) descrita na seção anterior. Outro ponto de vantagem apresentado por esta tecnologia é o fato de ser independente de plataforma.

A implementação da Camada de Serviço presente no núcleo da abordagem (*Intermediary Service*, *Detection Methods Service* e *Metrics Service*) foi implementada com a tecnologia JAX-RS (JAX-RS, 2018). O framework JAX-RS provê meios para implementações de *Webservices* REST com recursos a serem consumidos pelo aplicativo cliente. A comunicação entre os módulos da abordagem foi implementada baseando-se nestas interações entre estes Serviços.

O módulo cliente da abordagem (*Client App*) foi desenvolvido com base em JavaFX (JAVAFX, 2018), o qual faz parte da atual especificação Java como *API* de interface com o usuário.

O armazenamento de dados resultantes da execução da abordagem foi feito utilizado o banco de dados MongoDB (MONGODB, 2018). Este é um banco NoSQL *open-source* orientado a documentos que está sendo utilizado por diversas empresas de grande porte como SAP, Ebay, Adobe, entre outras.

Visando à aplicabilidade da abordagem proposta tem-se a especificação GridFS (MONGODB, 2018). A Especificação GridFS trabalha no armazenamento de arquivos, a qual ao inserir novos arquivos, fragmenta-os no banco de dados em *chunks* (pedaços). Este tipo de especificidade é útil para inserção/obtenção de projetos sendo avaliados pela abordagem. Ao registrar o projeto no núcleo da abordagem, por exemplo, o mesmo é compactado e inserido no banco de dados para posterior avaliação.

5.2 AVALIAÇÃO DA ABORDAGEM PROPOSTA

Além dos cenários de testes contemplados nos artigos dos métodos de Wei *et al*. (2014) e Zafeiris *et al*. (2017), também foram selecionados outros projetos para avaliar a abordagem.

A obtenção dos projetos foi obtida na plataforma de desenvolvimento *GitHub* (GITHUB, 2018), que foi escolhida devido sua ampla utilização por grandes empresas como IBM, Paypal, Walmart e outros. Esta plataforma permite a gestão tanto de projetos privados quanto públicos, a partir disso, outros 50 (cinquenta) projetos foram escolhidos aleatoriamente para a segunda fase de avaliação. Nesta seleção, os projetos foram filtrados apenas por sua linguagem de programação, a saber, Java.

Após selecionar os projetos de testes, todos foram submetidos a avaliação da ferramenta implementada para a abordagem proposta. A Tabela 5 apresenta os projetos que não tiveram candidatos à refatoração, ou seja, as classes dos projetos foram avaliadas uma a uma, mas os métodos de refatoração que estavam contemplados na *RMT* não encontraram nenhum candidato à refatoração. Esta tabela identifica os projetos com um código “P” precedido de seu identificador numérico, o local de onde o projeto foi obtido e o número de classes.

Tabela 5 - Projetos de Cenários de Testes - Sem Candidatos a Refatoração

Código	Repositório	Nº de Classes
P1	https://github.com/happyfish100/fastdfs-client-java.git	34
P2	https://github.com/kungfoo/geohash-java.git	24
P3	https://github.com/soundcloud/java-api-wrapper.git	38
P4	https://github.com/Froussios/Intro-To-RxJava.git	93
P5	https://github.com/eclipsesource/J2V8.git	99
P6	https://github.com/lzyzsd/JsBridge.git	9
P7	https://github.com/udacity/Just-Java.git	1
P8	https://github.com/shyiko/mysql-binlog-connector-java.git	97
P9	https://github.com/opentracing/opentracing-java.git	74
P10	https://github.com/kpelykh/docker-java.git	31
P11	https://github.com/firebase/firebase-admin-java.git	358
P12	https://github.com/codeborne/selenide.git	407
P13	https://github.com/sendgrid/sendgrid-java.git	113
P14	https://github.com/dawei101/shadowsocks-android-java.git	40
P15	https://github.com/swagger-api/swagger-parser.git	152
P 16	https://github.com/twilio/twilio-java.git	1499
P 17	https://github.com/pedrovgs/Algorithms.git	166
P18	https://github.com/isee15/captcha-ocr.git	21
P19	https://github.com/cucumber/cucumber-java-skeleton.git	3
P20	https://github.com/xpinjection/java8-misuses.git	52
P21	https://github.com/MagnusS/Java-BloomFilter.git	3
P22	https://github.com/dmpe/JavaFX.git	32
P23	https://github.com/DomHeal/JavaFX-Chat.git	32
P24	https://github.com/scream3r/java-simple-serial-connector.git	7
P25	https://github.com/jcip/jcip.github.com.git	144
P26	https://github.com/etcd-io/jetcd.git	141
P27	https://github.com/jsonld-java/jsonld-java.git	40
P28	https://github.com/bwaldvogel/liblinear-java.git	31
P29	https://github.com/jasonross/Nuwa.git	8
P30	https://github.com/taskadapter/redmine-java-api.git	138
P31	https://github.com/wg/scrypt.git	19
P32	https://github.com/oxo42/stateless4j.git	45
P33	https://github.com/soundcloud/java-api-wrapper.git	38

Fonte: Autoria Própria

Há também aqueles projetos que ao serem avaliados pela abordagem apresentaram candidatos à refatoração. A Tabela 6 exibe os projetos que

contemplavam candidados à refatoração. Esta tabela, da mesma forma que a Tabela 6 combina informações de Código, Repositório e Nº de Classes do projeto avaliado.

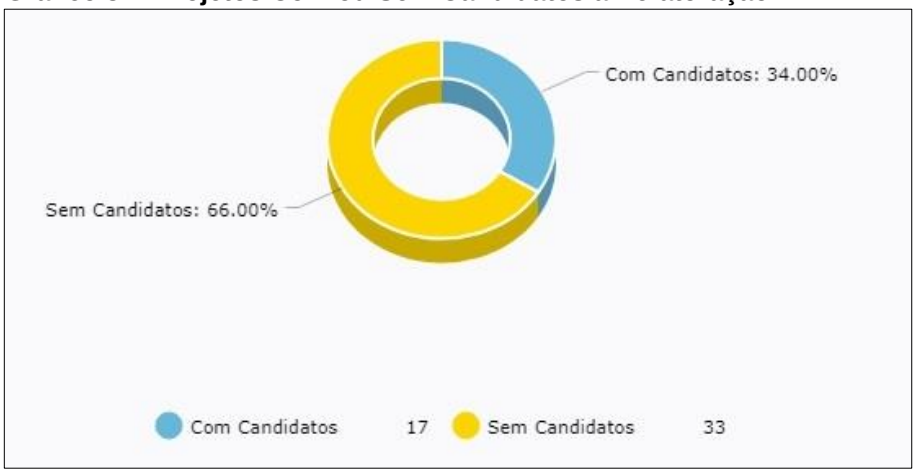
Tabela 6 - Projetos de Cenários de Testes - Com Candidatos a Refatoração

Código	Repositório	Nº de Classes
P34	https://github.com/evant/gradle-retrolambda.git	23
P35	https://github.com/Progether/JAdventure.git	61
P36	https://github.com/menacher/java-game-server.git	202
P37	https://github.com/google/caliper.git	291
P38	https://github.com/Devskiller/jfairy.git	99
P39	https://github.com/jpush/jpush-api-java-client.git	87
P40	https://github.com/dain/leveldb.git	105
P41	https://github.com/census-instrumentation/opencensus-java.git	465
P42	https://github.com/raml-org/raml-java-parser	493
P43	https://github.com/Javacord/Javacord.git	548
P44	https://github.com/maxmind/geoip-api-java.git	28
P45	https://github.com/kwhat/jnativehook.git	41
P46	https://github.com/cardillo/joinery.git	52
P47	https://github.com/Kurento/kurento-java.git	514
P48	https://github.com/joscha/play-authenticate.git	369
P49	https://github.com/caprica/vlcj.git	469
P50	https://github.com/careermonk/DataStructureAndAlgorithmsMadeEasyInJava.git	160

Fonte: Autoria Própria

O Gráfico 8 apresenta os projetos com ou sem candidato a refatoração. Dos 50 projetos selecionados, 34% apresentaram candidatos e 66% não. Dentre aqueles projetos que tiveram avaliações positivas para refatoração (com candidatos), alguns candidatos se referiam ou o padrão *Strategy* ou ao *Template Method*. O padrão *Factory Method*, que também é uma das sugestões previstas nos métodos implementados, não está presente nos gráficos e tabelas de resultados pois nenhum dos projetos com candidatos a refatoração apresentou este padrão como possível candidato a refatoração.

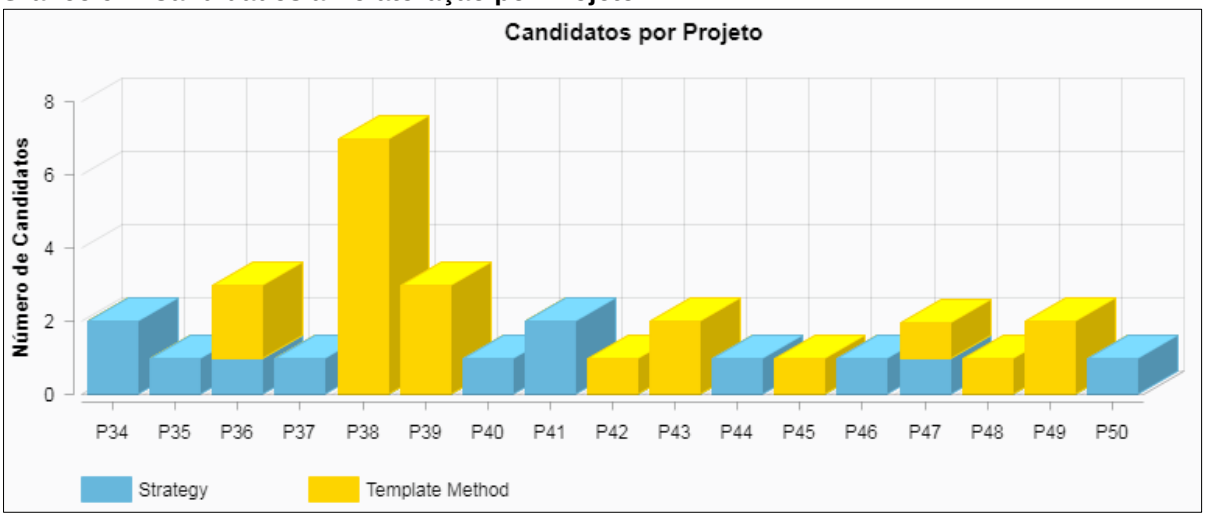
Gráfico 8 - Projetos Com ou Sem Candidatos a Refatoração



Fonte: Autoria Própria

O Gráfico 9 mostra o total de classes candidatas a refatoração por projeto avaliado. O número de candidatos foi dividido entre os dois padrões que foram encontrados; por exemplo, o projeto identificado como P3 tem um total de 3 (três) candidatos a refatoração, sendo 1 (um) com a sugestão de aplicação do padrão *Strategy* e outros 2 (dois) ao padrão *Template Method*.

Gráfico 9 - Candidados a Refatoração por Projeto



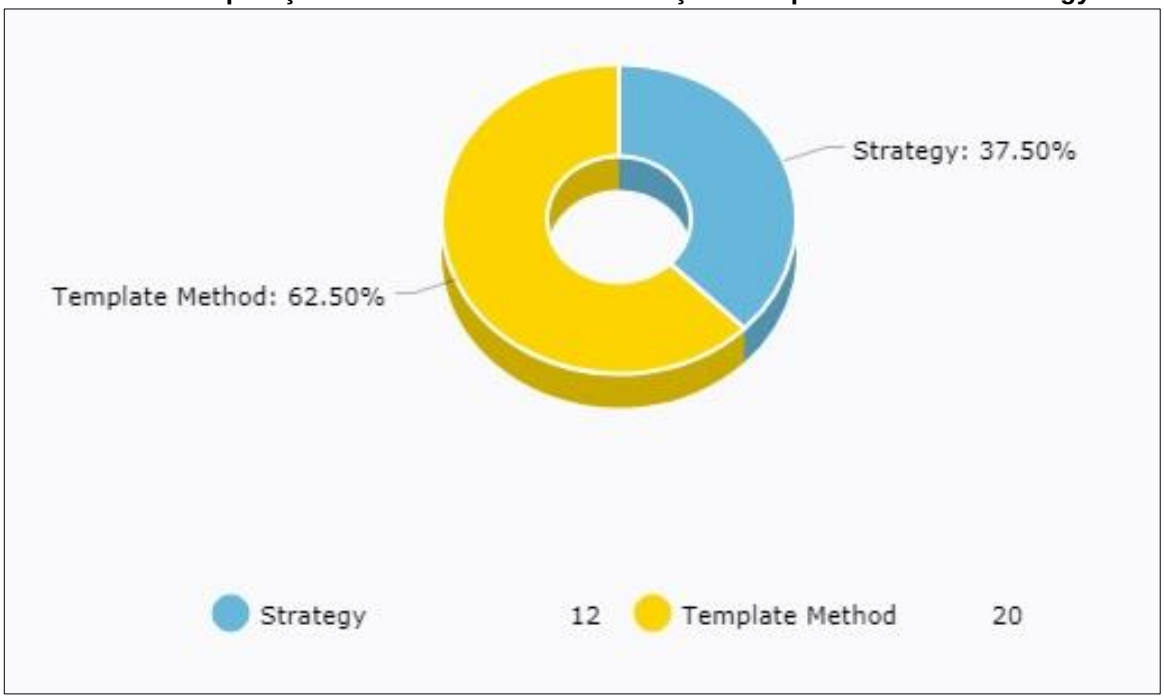
Fonte: Autoria Própria

Considerando o Gráfico 9 consta-se que os padrões *Strategy* e *Template Method* foram detectados em proporções similares e há projetos que apresentam um ou outro padrão.

O Gráfico 10 apresenta em destaque esta distribuição de detecção de candidatos a refatoração de padrões entre os trabalhos, sendo que 47,06% dos

trabalhos (P34, P35, P37, P40, P41, P44, P46 e P50) detectaram somente candidatos voltados ao padrão *Strategy*; 41,18% dos estudos (P38, P39, P42, P43, P45, P48 e P49) apresentam candidatos a aplicação do padrão *Template Method*; e os outros 2 (dois) trabalhos restantes P36 e P47 apresentaram detecções voltadas ambos os padrões (11,76%). Constata-se, então, que ao submeter um projeto *Proj1* para avaliação da abordagem, não existe uma tendência de aplicação do padrão *Strategy* ou *Template Method*.

Gráfico 10 - Comparação dos Candidatos a Refatoração: Template Method e Strategy



Fonte: Autoria Própria

Comparando a Tabela 5 com a Tabela 6, observou-se por meio do resultado da aplicação da abordagem que projetos com um número significativo de classes (mais de 1000) acabam não apresentando candidatos a refatoração e outros com um número mais reduzido de classes (menos de 30) tiveram candidatos a refatoração. Concluiu-se que o número total de classes de determinado projeto não é necessariamente um fator positivo na obtenção de candidatos a refatoração.

Além do número de classes usadas no processo de verificação de candidatos, tem-se detalhes da avaliação de atributos de qualidade: Manutenibilidade, Confiabilidade, Reusabilidade e Usabilidade. A Tabela 7 mostra o resultado das avaliações dos projetos submetidos ao processo de refatoração contendo o identificador do projeto (Código) que identifica de onde foi extraído, classe candidata

a refatoração; o padrão sugerido para aplicação: *Strategy* (S) ou *Template Method* (T); e finalmente, a avaliação do candidato.

Tabela 7 - Projetos de Cenários de Testes - Avaliações dos Atributos de Qualidade

Código	Classe Candidata	Padrão	Manutenibilidade	Confiabilidade	Reusabilidade
P34	Lib	S	3.8	-0.82	6.36
P34	Feature	S	3.8	-0.82	6.36
P35	LocationRepository	S	1.21	0.14	1.75
P36	NettyTPCServer	T	0.02	0.04	0.14
P36	SessionHandlerLatchCounter	T	0.02	0.03	0.13
P36	JetlangEventDispatcher	S	0.48	0.19	0.75
P37	ShortDuration	S	0.26	0	0.41
P38	KaFairyModule	T	-0.26	-0.04	-0.01
P38	EnFairyModule	T	-0.26	-0.04	-0.01
P38	PIFairyModule	T	-0.26	-0.04	-0.01
P38	SvFairyModule	T	-0.26	-0.04	-0.01
P38	ZhFairyModule	T	-0.26	-0.04	-0.01
P38	EsFairyModule	T	-0.26	-0.04	-0.01
P38	DeFairyModule	T	-0.26	-0.04	-0.01
P39	AndroidNotification	T	-0.14	-0.2	0
P39	WinphoneNotification	T	-0.13	-0.2	-0.02
P39	IosNotification	T	0.76	0.04	-0.01
P40	Slices	S	0.76	0.04	1.17
P41	StackDriverExporterWorker	S	0.16	0.04	0.28
P41	BenchMarkUtils	S	0.19	0	0.24
P42	ReferenceTypeSuggester	T	-0.01	-0.02	0.03
P43	ServerVoiceChannelUpdaterDelegateImpl	T	-0.02	-0.04	0
P43	TextChannelUpdaterDelegateImpl	T	-0.02	-0.04	0
P44	LookupService	S	3.59	0.54	5.38
P45	NativeMouseWeelEvent	T	-0.27	-0.4	-0.16
P46	DataFrameAdapter	S	1.26	0.12	1.93
P47	AbstractJsonRpcClientWebSocket	T	-0.01	-0.01	0.02
P47	JsonRpcConnectionListenerKurento	S	0.06	-0.02	0.1
P48	LinkedinAuthProvider	T	-0.04	-0.06	-0.01
P49	VideoTrackInfo	T	-0.04	-0.06	-0.01
P49	AudioTrackInfo	T	-0.04	-0.06	-0.01
P50	FibonnaciWithDP	S	0.98	-0.1	1.52

Fonte: Autoria Própria

A avaliação dos atributos é dada pela porcentagem de melhoria ou piora na classe sendo avaliada ao aplicar determinado candidato no código-fonte do seu respectivo projeto.

Considerando que, se o atributo de qualidade em questão apresentar um resultado positivo (maior que 0) é sinal de que o atributo foi aprimorado, ao passo que, se o resultado for negativo (menor que 0) é possível que o processo de refatoração seja prejudicial ao projeto em relação a classe candidata. Consequentemente, caso o resultado seja 0, a aplicação da classe no processo de refatoração não resultará em melhora do código-fonte em termos de atributos de qualidade.

Dentre as avaliações de candidatos, dez deles (31,25%) apresentaram índices positivos para a aplicação do respectivo candidato, isto significa que ao aplicar o padrão sugerido o projeto só é aprimorado com esta refatoração proposta. É importante observar que os projetos que apresentaram ao menos um índice negativo (demais 22 projetos), assim há vantagens na aplicação dos padrões, basta avaliar se a perda de um atributo de qualidade compensa o ganho de outro. Como por exemplo, a classe *FibonnaciWithDP* do projeto P50, que apresenta ganhos em termos de manutenibilidade e reusabilidade, porém perde em relação ao atributos de confiabilidade.

Um total de 12 (doze) projetos (37,5%) dos avaliados apresentaram resultados negativos nos 3 (três) atributos de qualidade avaliados (manutenibilidade, usabilidade e confiabilidade). Mesmo nesta situação, fica sob responsabilidade do usuário analisar se deseja ou não aplicar o padrão prevendo seus benefícios para a estutura do projeto.

Após a visualização dos resultados em relação aos atributos de qualidade, o usuário os analisa e escolhe quais os candidatos a refatoração deseja aplicar no código-fonte original, resultando em um código-fonte refatorado pelos métodos (*Strategy* e *Factory Method*) propostos por Wei *et al*. (2014) e/ou pelo método (*Template Method*) de Zafeiris *et al*. (2017). O resultado desta refatoração pode ser salvo em um diretório a escolha do usuário e manipulado posteriormente (conforme descrito na seção 4.3.3).

A Tabela 8 mostra um comparativo feito entre a abordagem proposta e o que já foi desenvolvido na literatura. A abordagem se utiliza de alguns conceitos já encontrados na literatura, como a mesma linguagem de programação (Java) da

maioria dos métodos de refatoração propostos, manipula os dados com base no conceito de Árvores de Sintaxe Abstrata e apresenta refatorações voltadas a expressões condicionais.

Nota-se também, que foi possível incorporar mais de um método de refatoração em uma única abordagem de refatoração, como apresentado pela Tabela 7. Como já foi mencionado, somente as pesquisas de Ouni et al. (OUNI et al., 2015; OUNI et al., 2017) se propõem a aplicar diversos métodos em uma única ferramenta de refatoração. A diferença entre as propostas de Ouni *et al*. e as apresentadas nesta abordagem, é que esta abordagem prevê pontos de extensão para que sejam adicionados mais métodos de refatoração, fornecendo um ambiente único de refatoração que engloba diversos métodos de refatoração da literatura.

Tabela 8 - Comparativo da Abordagem com o Estado da Arte

Critérios de Comparação	**Trabalhos da Literatura**	**Abordagem Proposta**
Abordagens de Extração de Dados	Árvores de Sintaxe Abstrata, Fatos Prolog, Framework SOOT, Java CC e Compilador Eclipse IDE	Árvores de Sintaxe Abstrata
Aplicação de Vários Métodos da literatura	Ouni et al. (2015) Ouni et al. (2017)	Zafeiris et al. (2017) Wei et al. (2014)
Extensibilidade para aplicar mais Métodos da literatura	Nenhum trabalho	Sim
Avaliações das Refatorações ANTES de aplicá-las	Nenhum trabalho	Sim
Linguagens de Programação	Java e Soul	Java
Tipos de Refatoração	Refatoração Composite para Visitor e Vice-Versa; Minitransformations; Reflective Refactorings; Refatorações baseadas em papéis; Intent Aspects; e Refatorações de expressões condicionais.	Refatorações de expressões condicionais

Fonte: Autoria Própria

Além disso, uma estrutura que é expansível a diversas abordagens de extração de dados também foi criada (Seção 4.3.3), implementando inicialmente a abordagem de Árvores de Sintaxe Abstrata para os métodos de Wei *et al*. (2014) e

Zafeiris *et al*. (2017). Os trabalhos estudados não contemplam a centralização de extração de dados entre métodos que se utilizam da mesma abordagem de extração.

Outro diferencial da abordagem está na sua forma de avaliação, que permite ao usuário decidir aplicar ou não as refatorações a partir de informações de atributos de qualidade. A Seção 4.3.4 apresentou como o *Metrics Service* foi construído de forma genérica visando a expansão deste módulo interno que permite ao desenvolvedor implementar outros atributos de qualidade que não foram contemplados pela abordagem proposta.

Como as formas de avalição mais comuns da literatura são avaliações de extração de candidatos, Então esta proposta se valeu do mesmo tema para verificar se os candidatos estavam sendo realmente obtidos (Tabela 6), e se as avaliações providas (Tabela 7), realmente dão subsídios às escolhas dos usuários. A partir do que foi observado nas tabelas e análises acima, todos os pontos foram cobertos com sucesso.

5.2.1 Limitações da Abordagem Proposta

A primeira restrição da abordagem se trata na remoção de candidatos voltados a refatoração de uma mesma classe. Se por exemplo, o padrão *Singleton* é escolhido a ser aplicado na classe *Classe1* e um outro padrão *Strategy* é igualmente escolhido para refatorar a mesma *Classe1*, então ambos os padrões não serão apresentados como candidatos a refatoração da *Classe1*. Isto prevendo conflitos de refatoração entre os métodos em casos que vão se utilizar dos blocos de código internos as classes.

Outra limitação da abordagem está presente na adição de mais métricas de software. Caso não exista uma implementação disponível na *web* para extração da métrica em questão, é necessário que o desenvolvedor implemente essa extração com base nas diretrizes da métrica que se deseja aplicar.

A terceira limitação está relacionada à disposição dos módulos da abordagem, como é observado na Seção 4.1. Na descrição da visão geral da abordagem, é observado que o módulo *Intermediary Service* gerencia todas as interações entre os demais módulos da abordagem, com isso, nota-se que qualquer falha apresentada neste módulo pode acarretar no mal funcionamento de toda a abordagem.

As restrições mencionadas anteriormente focam na abordagem proposta, porém uma restrição foi detectada em nível da ferramenta implementada. A RMT foi concebida inicialmente como uma ferramenta a parte do ambiente de desenvolvimento do usuário, posteriormente, aconselha-se que seja desenvolvido um *plugin* a um ambiente integrado de desenvolvimento, como por exemplo, Eclipse.

5.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou a implementação da abordagem proposta e sua avaliação. Um protótipo inicial foi desenvolvido visando avaliar a implementação dos métodos de refatoração incorporados na abordagem. Este protótipo foi submetido a avaliação de extração de candidatos, que consiste em verificar se o método de refatoração está efetivamente obtendo candidatos a refatoração.

A avaliação tinha por objetivo verificar se os padrões *Strategy*, *Factory Method* e *Template Method* seriam apresentados como candidatos a refatoração. Além disso, após a obtenção de candidatos e posteriormente a solicitação de refatoração por meio deles, foi verificado o resultado satisfatório em relação ao que foi alcançado pelas abordagens Wei *et al*. (2014) e Zafeiris *et al*. (2017).

A *RMT* foi construída incorporando os módulos previstos na abordagem. A partir dele foram analisados mais cinquenta projetos *open-source*, desta vez, não apresentando apenas candidatos a refatoração e os refatorando, mas também informando valores percentuais de melhoria ou não que os candidatos ao serem aplicados poderiam trazer ao código-fonte.

6 CONCLUSÃO

Devido a ausência de revisões e mapeamento sistemáticos no tema proposto, este trabalho foi iniciado com a execução de um mapeamento sistemático contemplando os anos de 1997 a 2017, o qual identificou características como: linguagens de programação comumente utilizadas, abordagens de extração de dados presentes nos projetos, tipos de refatorações, pontos de melhoria, entre outros.

Com as informações obtidas no mapeamento sistemático, esta dissertação criou uma abordagem de refatoração voltada a aplicação de padrões de projeto. A estrutura interna da abordagem é definida com quatro módulos que realizam interações com o usuário (*Client App*), promovem interações entre os módulos do núcleo da abordagem e o módulo cliente (*Interaction Service*), incorporam métodos de refatoração da literatura que apresentam funções de extração de candidatos e refatorações voltadas a padrões de projeto (*Detection Methods Service*) e avaliam o código-fonte (Metrics Service).

A abordagem foi avaliada com a criação de um protótipo inicial contendo refatorações simples presentes nos métodos de refatoração por elas incorporados; e posteriormente, foi construído uma ferramenta (*RMT*) que abrangeu todos os módulos da abordagem proposta. A avaliação da abordagem foi realizada utilizando as formas adotadas pelos trabalhos que foram encontrados no mapeamento sistemático.

O último cenário de teste para avaliação abrangeu a análise de cinquenta projetos *open-source* encontrados na *web*, onde cada um destes foi submetido a obtenção de candidatos da ferramenta e a sua refatoração. Esta avaliação permitiu verificar o percentual de projetos, dentre os avaliados, que apresentaram sugestões a refatoração e o impacto que avaliação de atributos de qualidade pode ter na escolha de padrões candidatos por parte do usuário.

A partir deste cenário, comprovou-se que foi possível atender a proposta inicial da abordagem, de aplicar diversos métodos de refatoração da literatura; proporcionar um ambiente expansível a aplicação de mais métodos de refatoração, avaliar atributos de qualidade; e por fim, promover mais interações com o usuário, dando a ele mais controle e *insigts* para realizar as refatorações.

Dentre as contribuições apresentadas por esta pesquisa, têm-se como as principais: um mapeamento sistemático referente à detecção de pontos de inserção

de padrões de projeto em código-fonte; a pré-visualização de candidatos à refatoração para que a seleção dos padrões a serem aplicados fique sob a responsabilidade do usuário e juntamente a visualização de canditados tem-se também a avaliação do padrão sugerido em termos de métricas e atributos de qualidade.

6.1 DIFICULDADES E LIMITAÇÕES

Uma dificuldade encontrada ao desenvolver o *Metrics Service*, foi encontrar projetos *open-source* que já apresentassem as métricas de software desenvolvidas. Neste caso, apenas uma implementação não foi encontrada, a saber: número de mensagens de erro. O fator agravante a isto é que nem mesmo foi encontrado um referêncial teórico confiável para que se pudesse desenvolver a implementação desta métrica.

6.2 TRABALHOS FUTUROS

Como a abordagem criada tem vários pontos de extensão, como sugestão de trabalhos futuros têm-se:

- Incorporar ao módulo *Detection Methods Service* mais métodos de refatoração voltados a aplicação de padrões de projeto;
- Aplicar ao *Detection Methods Service*, diferentes abordagens de extração de dados, pois a implementação atual contempla uma abordagem de extração da literatura;
- Realizar mais avaliações de atributos de qualidade para um candidato a refatoração.

Além dos pontos de extensão previstos na abordagem, há também limitações que podem ser resolvidas futuramente:

- Apresentar uma solução para o módulo *Intermediary Service*, o qual em caso de falhas, pode comprometer o processo de refatoração;
- Resolver conflitos em refatorações realizadas em uma única classe, no presente momento, candidatos a refatoração que são aplicados em uma mesma classe são descartados das sugestões a refatoração.

Alguns testes podem ser executados uma vez que a abordagem esteja completamente desenvolvida:

- Submeter a própria implementação da abordagem no processo de refatoração, verificando a qualidade da mesma;
- Avaliar tempo de execução da abordagem proposta, o tempo de execução dos módulos internos a ela individualmente; visando encontrar pontos de melhoria em relação a performance.

REFERÊNCIAS

ACM. **SCIJournal**. Disponível em: <http://www.scijournal.org>. Acesso em 02 fev. 2018.

BELUZZO, L.B; MATOS, S.N; PACHER, T.H. A Refactoring architecture for measuring and identifying spots of design patterns insertion in source code. In: ICSOFT. **Proceedings of the 13th International Conference on Software Technologies**. [S.l.], 2018. p. 632-639.

CHRISTOPOULOU, A. *et al*. Automated refactoring to the strategy design pattern. **Information and Software Technology**, Elsevier, v. 54, n. 11, p. 1202-1214, 2012.

CINNÈIDE, M. Ó. **Automated application of design patterns:** a refactoring approach. 2001. 242 f. Thesis (Doutorado) — Programa de Pós-Graduação University of Dublin, Trinity College Dublin, 2001.

CINNÈIDE, M. Ó. Automated refactoring to introduce design patterns. In: ACM. **Proceedings of the 22nd international conference on Software engineering**. [S.l.], 2000. p. 722-724.

CINNÈIDE, M. Ó.; NIXON, P. A methodology for the automated introduction of design patterns. In: IEEE. **Software Maintenance, 1999. (ICSM'99) Proceedings. IEEE International Conference on**. [S.l.], 1999. p. 463-472.

CINNÈIDE, M. Ó; NIXON, P. Automated software evolution towards design patterns. In: ACM. **Proceedings of the 4th international workshop on Principles of software evolution**. [S.l.], 2001. p. 162-165.

CK GITHUB. Disponível em < https://github.com/mauricioaniche/ck/>. Acessado em: 21 Setembro.de 2018.

CLOCKSIN, W. F.; MELLISH, C. S. **Programming in Prolog**, New York, 2003.

DUCASSE, S; RIEGER, M; GOLOMINGI G. **Tool support for refactoring duplicated OO code**. In ECOOP, 1999.

DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: Asystematic review. **Information and software technology**, Elsevier, v. 50, n. 9, p. 833-859, 2008.

ECLIPSE IDE. Disponível em <http://eclipse.org>. Acessado em: 06 Agosto.de 2018.

ECLIPSE JDT. Disponível em < https://www.eclipse.org/jdt/>. Acessado em: 20 Agosto.de 2018.

FOWLER, *et al*. **Refactoring: improving the design of existing code**. [S.l.]: Addison-Wesley Professional, 1999.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**, Porto Alegre, 2006.

GAITANI, M. A. G *et al*. Automated refactoring to the null object design pattern. **Information and Software Technology**, Elsevier, v. 59, p. 33-52, 2015.

GAMMA, E. *et al*. **Design Patterns: Elements of Reusable Object-Oriented Software**, Indianapolis, 1994.

GAROUSI, V. MÄNTYLÄ, M. V. A systematic literature review of literature reviews in software testing. **Information and Software Technology**, Elsevier, v. 80, p. 195-216, 2016.

GE, X. DUBOSE, Q. L. MURPHY-HILL, Emerson. Reconciling manual and automatic refactoring. In: IEEE. **Software Engineering (ICSE), 2012 34th International Conference on**. [S.l.], 2012. p. 211-221.

GE, X. MURPHY-HILL, E. Manual refactoring changes with automated refactoring validation. In: ACM. **Proceedings of the 36th International Conference on Software Engineering**. [S.l.], 2014. p. 1095-1105.

GITHUB. Disponível em < https://github.com>. Acessado em: 26 Setembro.de 2018.

GOOGLE SCHOLAR. Disponível em <http://scholar.google.com.br>. Acessado em: 02 Fev.de 2018.

GUINEA, A. S. NAIN, G.; TRAON, Y. L. A systematic review on the engineering of software for ubiquitous systems. **Journal of Systems and Software**, Elsevier, v. 118, p. 251-276, 2016.

JAVA PARSER. Disponível em < https://github.com/javaparser/javaparser>. Acessado em: 01 Outubro.de 2018.

JAVA. Disponível em <https://www.oracle.com/br/java/>. Acessado em: 06 Agosto.de 2018.

JAVACC. Disponível em <http://javacc.org>. Acessado em: 06 Agosto.de 2018.

JAVAFX. Disponível em <https://www.oracle.com/technetwork/pt/java/javafx/overview/index.html>. Acessado em: 06 Agosto.de 2018.

JAX-RS. Disponível em <https://jcp.org/en/jsr/detail?id=370>. Acessado em: 06 Agosto.de 2018.

JEON, S. LEE, J.; BAE, D. An automated refactoring approach to design pattern-based program transformations in java programs. In: IEEE. **Software Engineering Conference, 2002. Ninth Asia-Pacific**. [S.l.], 2002. p. 337-345.

JONES, J. Abstract Syntax Tree Implementation Idioms. In: PLOP **Proceedings of the 10th conference on pattern languages of programs**. [S.l.], 2003. p. 1-10.

JUILLERAT, N.; HIRSBRUNNER, B. Toward an implementation of the “form template method” refactoring. **Seventh IEE International Workding Conference on Source Code Analysis and Manipulation, SCAM**, 2007. p.81-90.

KERIEVISKY, J. **Refatoração para Padrões**, Porto Alegre, Bookman, 2008.

KIM, J.; BATORY, D.; DIG, D. **Scripting refactorings in Java to introduce design patterns**. [S.l.], 2014.

KITCHENHAM, B. CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. [S.l.], 2007.

LIU, W.; *et al*. Automated pattern-directed refactoring for complex conditional statements. **Journal of Central South University**, Springer, v. 21, n. 5, 2014.

MARTIN, R. C. **Clean code:** a handbook of agile software craftsmanship. [S.l.]: Pearson Education, 2009.

MARTINS, L. E. G. GORSCHEK, T. Requirements engineering for safety-critical systems: A systematic literature review. **Information and Software Technology**, Elsevier, v. 75, p. 71-89, 2016.

MENS, T.; TOURWÉ, T. A Declarative evolution framework for object-oriented design patterns. **Proceedings on the IEEE International Conference on Software Maintenance (ICMS)**, IEEE 2001.

MENS, T.; TOURWÉ, T. A survey of software refactoring. **IEEE Transactions on software engineering**, v. 30, n. 2, p. 126-139, 2004.

MONGODB. Disponível em <https://www.mongodb.com/>. Acessado em: 21 Setembro.de 2018.

MURPHY-HILL, E; BLACK, A. P. **Breaking the barriers to successful refactoring: Observations and tools for extract method**. In ICSE, 2008.

OLSINA, L. et al. Specifying Quality Characteristics and Attributes for Web Sites. In: ACM. **Proceedings of the first ICSE Workshop on Web Engineering**. [S.l.], 1999.

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

OUNI, A. *et al.* A multi-objective refactoring approach to introduce design patterns and fix anti-patterns. **North American Search Based Software Engineering Symposium**, [S.l.], 2015. p. 1-15.

OUNI, A. *et al.* More: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells. **Journal of Software: Evolution and Process**, Wiley Online Library, v. 29, n. 5, 2017.

PAGANI, R. N. KOVALESKI, J. L.; RESENDE, L. M. **Methodi ordinatio: a proposed methodology to select and rank relevant scientific papers encompassing the impact factor, number of citation, and year of publication**. Scientometrics, Springer, v. 105, n. 3, p. 2109-2135, 2015.

RAJESH, J.; JANAKIRAM, D. JIAD: A tool to infer design patterns in refactoring. **Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming**, 2004. p.227-237.

RAM, J. D. RAJESH, J. Detecting Intent Aspects from Code to Apply Design Patterns in Refactoring: An Approach Towards a Refactoring Tool. In: **Proceedings of 2nd Workshop of Software Design and Architecture**. [S.l.], v.4, 2004.

ROBERTS, D. B. JOHNSON, R. **Practical analysis for refactoring**. [S.l.]: University of Illinois at Urbana-Champaign, 1999.

SOMMERVILLE, I. **Engenharia de Software**, São Paulo, 2011.

TYRUBA. Disponível em < http://tyruba.sourceforge.net/ >. Acessado em: 02 Outubro.de 2018.

VALLE-RAI, R. *et al*. Soot: A java bytecode optimization framework. **Proceedings of the 1999 conference of the Center of Advanced Studies on Collaborative Research**. IBM Corp. 2010. p. 214-224.

WEI, L. *et al.* Automated pattern-directed refactoring for complex conditional statements. **Springer**. 2014. p.1935-1945.

WILDFLY. Disponível em < http://www.wildfly.org//>. Acessado em: 26 Setembro.de 2018.

WUYTS, R. *et al*. **A logic meta-programming approach to support the co-evolution of object-oriented design and implementation**. Tese (Doutorado) — PhD thesis, Vrije Universiteit Brussel, 2001.

XENOS, M. *et al*. Object-Oriented metrics: a survey. **Proceedings of the Federation of European Software Measurement Associations**, Madrid, Spain, 2000.

ZAFEIRIS, V. E. *et al.* Automated refactoring of super-class method invocations to the template method design patterns. **Information and Software Technology**. 2017. p.19-35.

SOMMERVILLE, I. **Engenharia de Software**, São Paulo, 2011.