UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RHUAN FELIPE OLIVEIRA PRADO

ANÁLISE COMPARATIVA DE MÉTODOS DE INSERÇÃO E DETECÇÃO DE PADRÕES DE PROJETO

PONTA GROSSA

2021

RHUAN FELIPE OLIVEIRA PRADO

ANÁLISE COMPARATIVA DE MÉTODOS DE INSERÇÃO E DETECÇÃO DE PADRÕES DE PROJETO

Comparative analysis of insertion and detection methods of the design patterns

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof. Dra. Simone Nasser Matos

Coorientador: Prof. MSc. Giovane Galvão

PONTA GROSSA

2021

RHUAN FELIPE OLIVEIRA PRADO

ANÁLISE COMPARATIVA DE MÉTODOS DE INSERÇÃO E DETECÇÃO DE PADRÕES DE PROJETO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof. Dra. Simone Nasser Matos

Coorientador: Prof. MSc. Giovane Galvão

Data de aprovação: 25 de novembro de 2021

_____

Profa. Simone Nasser Matos
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Profa. Helyane B. Borges
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Prof. Eliana C. M. Ishikawa
Doutorado
Universidade Tecnológica Federal do Paraná

PONTA GROSSA
2021

Dedico este trabalho à minha família, pelos momentos de ausência.

AGRADECIMENTOS

Primeiramente quero agradecer a Deus, agradecer a todas as pessoas que fizeram parte dessa importante fase de minha vida, e também aqueles que não puderam presenciar a finalização desse trabalho.

Agradeço a minha orientadora Profa. Dra. Simone Nasser Matos, pela dedicação com que me guiou nesta trajetória e também ao coorientador Prof. MSc. Giovane Galvão.

Aos meus colegas de sala.

A Secretaria do Curso, pela cooperação.

Gostaria de agradecer à minha família, pois sem o apoio deles seria muito difícil vencer esse desafio.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

RESUMO

A refatoração é uma forma de reestruturar o sistema de um software, sem alterar seu comportamento externo, proporcionando uma melhoria na qualidade do código-fonte. Ferramentas foram desenvolvidas para ajudar no processo de refatoração baseada em padrões de projeto tal como a *Refactoring and Measurement Tool* (RMT). Esta ferramenta contém métodos de inserção e detecção de padrões de projeto, porém novos métodos podem ser adicionados a sua arquitetura. A adição de novos métodos é possível mediante o estudo dos métodos da literatura que detectam e inserem padrões de projeto. Este trabalho realizou uma análise de três métodos capazes de inserir e detectar padrões de projetos em um código-fonte. A análise foi executada elencando características de comparação tais como: processo de leitura de código-fonte, padrões que são identificados pelos métodos, entre outros. Como resultado verificou-se que poucos são os padrões identificados pelos métodos e estes possuem um funcionamento de detecção e inserção de padrões diferentes, apesar de alguns métodos conseguirem contemplar o mesmo padrão de projeto.
.

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

ABSTRACT

Refactoring is a way to restructure a software system, without changing its external behavior, providing an improvement in the quality of the source code. Tools have been developed to help in the process of pattern-based design refactoring such as the Refactoring and Measurement Tool (RMT). This tool contains methods for inserting and detecting design patterns, but new methods can be added to its architecture. The addition of new methods is possible by studying the methods in the literature that detect and insert design patterns. This work performed an analysis of three methods capable of inserting and detecting design patterns in a source code. The analysis was performed by listing comparison characteristics such as: source code reading process, patterns that are identified by the methods, among others. As a result it was verified that few are the patterns identified by the methods and these methods have a different way of detecting and inserting patterns, although some methods manage to contemplate the same design pattern.

Keywords: Refactoring. Design Patterns. Methods.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A refatoração de software é uma forma de reestruturar o sistema melhorando a qualidade do código-fonte. Fowler (2019) comenta que um código bem arquitetado e de boa qualidade, sempre necessita de aperfeiçoamento em que se adiciona novos recursos, por isso, as atualizações são aplicadas. Essas mudanças causam alterações na sua legibilidade e dificulta a manutenibilidade do sistema. A refatoração ajuda o código ser mais compreensível e também modificá-lo por meio da eliminação de possíveis problemas e melhora os atributos de qualidade, permitindo transformar o código do sistema mais modular e organizado (LACERDA *et al*., 2020).

A refatoração pode ser realizada por meio de técnicas (FOWLER *et al*., 2018) ou padrões de projetos (KERIEVSKY, 2008). A refatoração por meio de técnicas define trechos de um código que necessitam de refatoração por causa dos *bad smells* (“Maus Cheiros”) e estabelece uma sugestão de qual técnica pode ser usada. Dentre as técnicas pode-se citar: Código Duplicado, Método Longo, Classes Grandes, Lista Longa de Parâmetros, Alteração Divergente entre outras (FOWLER *et al*., 2018).

A refatoração baseada em padrões de projeto representa um arcabouço que proporciona aprimorar o código simplificando a lógica e aumentando a flexibilidade. Eles ajudam tornar um sistema reutilizável e pode melhorar a manutenção e a documentação (GAMMA *et al*., 1994). Para a aplicação de um padrão de projeto é necessário analisar o código e verificar a relevância da sua aplicabilidade, determinar o tipo de refatoração, escolher os participantes no contexto da aplicação e por fim realizar a reestruturação de código. A aplicação de um padrão de projeto também pode levar, por exemplo, a um aumento desnecessário na complexidade de um programa devido a um alto número de classes com métodos pequenos, como no padrão de projeto *State* (DEREZIŃSKA, 2017).

A refatoração é uma técnica para melhorar a legibilidade de um código em que se aplica uma série de pequenas modificações preservando o comportamento, cada uma dessas modificações terá um efeito acumulativo relevante (FOWLER *et al*., 2018). O uso de ferramentas para realizar o processo de refatoração pode diminuir a probabilidade de erro em relação aos processos manuais (GE; DUBOSE; MURPHY-HILL, 2012).

Beluzzo (2018) fez um mapeamento para levantamento de trabalhos relacionados a detecção de pontos de inserção de padrões de projetos no código-fonte,

bem como de ferramentas para auxiliar no processo de refatoração. O estudo contemplou trabalhos no período de 1997 a 2017 no qual obteve-se 1.149 trabalhos na área, sendo que destes, 26 foram selecionados para uma avaliação detalhada. A metodologia utilizada para a realização do mapeamento foi uma junção de Kitchenham e Charters (2007) e Pagani e Kovaleski e Resende (2015).

Durante a realização do mapeamento, Beluzzo (2018) verificou que poucas são as ferramentas voltadas para refatoração baseada em padrões de projeto e as que existem contemplam somente um processo de refatoração. Por isto, em sua dissertação criou a *Refactoring and Measurement Tool* (RMT) que contempla mais de um método de refatoração baseado em padrões de projeto e é uma ferramenta em que se submete um projeto desenvolvido em linguagem Java e após ser avaliado retorna aos usuários as classes candidatas a refatoração. O trabalho tem vários pontos de extensão, um deles é a incorporação de métodos de refatoração para detecção e aplicação de padrões de projeto dentro da ferramenta RMT.

Este trabalho realizou uma comparação entre três métodos de refatoração definidos no mapeamento feito por Beluzzo (2018), a saber, Kim *et al*. (2014) que é um método que utiliza a reflexão do projeto para analisar quais padrões podem ser aplicados ao código-fonte; Gaitani *et al*. (2015) que tem por objetivo aplicar um método *Null Object* para substituir as comparações de objetos nulos e Mens e Tourwé (2001) utiliza a técnica de meta-programação declarativa para manipular programas orientado a objetos.

A análise comparativa proposta por este trabalho é necessária para identificar pontos comuns e diferentes entre os métodos a fim de que possam ser futuramente incorporados na arquitetura da ferramenta RMT criada por Beluzzo (2018). A análise identificou: i) quais padrões são cobertos pelos métodos; ii) como é o funcionamento interno dos métodos; iii) como é realizada a leitura do código-fonte; iv) como é realizada a aplicação do padrão de projeto no código-fonte; v) aplicação do método em códigos-fontes; vi) os tributos de qualidade que cada método utiliza; vii) se o método apresenta uma ferramenta; e viii) como incorporar os métodos na RMT.

1.1 Objetivos

Esse trabalho tem como objetivo criar um processo de comparação de métodos de detecção e inserção de padrões de projeto identificando os pontos comuns e específicos para que os mesmos possam ser incorporados *Refactoring and Measurement Tool* (RMT).

Os objetivos específicos delimitados para este trabalho são:

- Analisar os métodos de Kim *et al.* (2014) Método de Refatoração Reflexiva R^2, Gaitani *et al*. (2015) Método *Null Object* e de Mens e Tourwé (2001) método refatoração utiliza Meta-programação.
- Identificar pontos comuns e específicos entre os métodos.

1.2 Organização do Trabalho

Este trabalho está organizado em quatro capítulos. O Capítulo 2 traz uma síntese sobre a importância refatoração e apresenta refatorações voltadas a padrões de projeto e os métodos que serão comparados Método de Refatoração Reflexiva R^2 (KIM *et al*. 2014), Método *Null Object* (GAITANI *et al*. (2015) e o método refatoração que utiliza Meta-programação (MENS TOURWE, 2001).

O Capítulo 3 mostrará o processo de comparação entre os métodos apresentados nos Capítulo 2, bem como a discussão sobre os resultados desta comparação.

O Capítulo 4 apresenta as considerações finais deste trabalho e os possíveis trabalhos futuros que podem ser realizados a partir desta pesquisa.

2 REFATORAÇÃO DE SOFTWARE

Este capítulo apresenta o referencial teórico usado para o desenvolvimento deste trabalho. A Seção 2.1 relata a importância da refatoração em relação a manutenção de software, aplicação da refatoração, métodos de refatoração baseado em padrões de projeto. A Seção 2.2 apresenta os resultados parciais do mapeamento realizado por Beluzzo (2018) e a explicação dos métodos Kim *et al.* (2014), Gaitani *et al.* (2015), Mens e Tourwé (2001) usados neste trabalho para serem comparados. A Seção 2.3 apresenta as características *Refactoring and Measurement Tool* (RMT). Por fim, a última seção apresenta as considerações finais deste capítulo.

2.1 Importância da Refatoração

A refatoração de software é uma forma de reestruturar o sistema, sem alterar seu comportamento externo melhorando a qualidade do código-fonte e ajudando a obter um melhor entendimento do mesmo e assim possibilitar a adição de código de maneira simplificada, facilitando a inserção de novas funcionalidades (KERIEVSKY, 2008).

Fowler *et al*. (2018) comenta que a refatoração é uma ferramenta rica que ajuda a manter o código seguro, auxilia a desenvolver mais rapidamente e evita a deterioração do projeto, deixando-o melhor. A utilização da refatoração é importante pois pode tornar o código mais compreensível e de fácil manutenção (FOWLER *et al*., 2018; KERIEVSKY, 2008).

A refatoração pode ser aplicada usando técnicas (FOWLER *et al*., 2018) ou em padrões de projetos (KERIEVSKY; 2008). A refatoração utilizando técnicas é definida em trechos de um código que necessitam ser reescritos e após é feito uma série de pequenas modificações preservando o seu comportamento, mas essa sequência de transformações pode produzir uma reestruturação significativa, diminuindo a probabilidade que erros sejam inseridos.

Fowler *et al*. (2018) estabelece uma série de técnicas de refatoração, dentre elas: Código Duplicado, Método Longo, Classes Grandes, Lista Longa de Parâmetros, Alteração Divergente entre outras.

A refatoração fundamentada em padrões de projetos representa um arcabouço que proporciona aprimorar o código simplificando a lógica e aumentando a flexibilidade. Os padrões de projeto ajudam a selecionar alternativas de projeto que tornam um sistema reutilizável e evita alternativas que comprometam a reutilização (GAMMA et al., 1994). Padrões de projetos podem aprimorar a documentação e a manutenção de sistemas ao proporcionar uma especificação explícita de interações de classes e objetos e o seu objetivo implícito (GAMMA *et al*., 1994).

Cada padrão de projeto sistematicamente nomeia, explica e avalia um aspecto de projeto significativo, comum em sistemas orientados a objetos (GAMMA *et al*., 1994). Um padrão de projeto tem quatro elementos essenciais:

- Nome do padrão: que faz uma referência para descrever o problema de projeto, suas soluções e resultado em uma e duas palavras;
- Problema: descreve em que situação se deve aplicar o padrão;
- Solução: descreve os participantes que formam o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações;
- Consequências: são as decorrências e análises das vantagens e desvantagens da aplicação do padrão.

No entanto, algumas implementações de padrões podem tornar o código complicado do que ele precisaria ser; quando isso acontece, é preciso voltar as versões anteriores ou aplicar mais refatorações.

Os padrões de projeto são essenciais cada vez que são anexados no processo de refatoração e é adicionado mais flexibilidade ao código (CINNÉIDE, 2000).

Gamma *et al*. (1994) apresentam alguns exemplos de padrões de projeto: *template method, abstract factory, factory method, builder, flyweight, proxy, adapter, observer e visitor*. Fowler *et al*. (1999) e Kerievsky (2008), são uns dos diversos autores que apresentaram a refatoração aplicando padrões de projeto e suas teorias são usadas por outros pesquisadores.

2.2 Métodos de Refatoração Baseado em Padrões de Projeto

Beluzzo (2018) realizou um mapeamento para levantamento de trabalhos referente a detecção de pontos de inserção de padrões de projeto no código-fonte. O mapeamento faz o domínio de trabalhos no período de 1997 a 2017, obteve-se 1.149 trabalhos e destes 26 foram selecionados para uma avaliação detalhada. A metodologia aplicada no mapeamento foi uma junção de Kitchenham e Charters (2007) e Pagani e Kovaleski e Resende (2015). O Quadro 1 apresenta os métodos que foram identificados por Beluzzo (2018).

Quadro 1 - Trabalhos Selecionados para Análise

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

Chave	Autor	Título	Ano Publicação
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
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

Chave	Autor	Título	Ano Publicação
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: Beluzzo (2018)

Este trabalho descreve os três métodos que se encontram nesse mapeamento Kim *et al.* (2014), Gaitani *et al.* (2015) e Mens e Tourwé (2001) e que serão comparados. Um dos critérios utilizados é que o método não deveria estar implementado na arquitetura da ferramenta RMT e também foram selecionados métodos com base na sua diferença de aplicação da refatoração no intuito de procurar semelhança e diferenças entre eles.

2.1.1 Método da Refatoração Reflexiva (R²)

O método faz um aproveitamento da reflexão do projeto que a IDE (*Integrated Development Environment*) realiza. Utiliza Classes, Métodos e os Campos, e assim por diante, que estão em um projeto *P* definidos como *RClass*, *RMethod* e *RField*, respectivamente. Quando *P* é compilado pelo Eclipse, é criado um conjunto de tabelas para *RClass, RMethod, RField*, entre outros, onde cada linha corresponde a uma classe, método ou uma instância de um campo de *P* Kim *et al*. (2014).

Os campos *RClass, RMethod e RField*, a partir de agora são chamados de R² que define os relacionamentos de associação e herança entre as linhas da tabela. Os métodos das classes R² apresentam as refatorações primárias do Eclipse, transformações simples ou refatorações compostas.

Para a utilização das refatorações feitas por R² é apresentado ao programador como selecionar: um campo, método ou classe e faz a invocação da refatoração. O método apresenta 78% dos padrões de projeto da "Gangue dos quatro" e seus inversos como métodos curtos em Java na R².

No Quadro 2 mostra os padrões que o R² pode automatizar totalmente, parcialmente e os que não tem certeza se são automatizáveis. Ressalta-se que 35% são totalmente automatizáveis, 43% parcialmente e o restante 22%.

Quadro 2 - Possibilidade de Automatização de Padrões de Projetos

Padrões de Projeto	**Possibilidade de Automatização**		
	Completa	**Parcial**	**Inseguro**
Abstract Factory	X		
Adapter		X	
Bridge		X	
Builder	X		
Chain of Responsibility		X	
Command	X		
Composite		X	
Decorator		X	
Façade			X
Factory Method	X		
Flyweigth	X		
Interpreter			X
Iterator			X
Mediator			X
Memento	X		
Observer		X	
Prototype		X	
Proxy		X	
Singleton	X		
State			X
Strategy		X	
Template Method		X	
Visitor	X		
Porcentagem	35 %	43 %	22 %

Fonte: Adaptado de Kim; Batory; Dig (2014).

Os padrões que são totalmente automatizados, padrão *Visitor* e variantes, pois não produzem “TODOs” para um usuário. Já os padrões que são parcialmente automatizáveis, a criação desses padrões produz “TODOs” que deve ser preenchido por um usuário. Os padrões restantes são específicos e necessitam de uma abstração que requer um conhecimento profundo de um aplicativo.

2.2.2 Método de Automatização *NULL OBJECT*

O padrão de projeto *NULL OBJECT* tem por objetivo a eliminação de condicionais que fazem verificação de objetos nulos por meio de mecanismos de herança e polimorfismo, o padrão também favorece a reusabilidade Gaitani *et al*. (2015).

Para a automatização do padrão foi proposto pelo autor um algoritmo que faz o processamento de todas as classes do projeto, para analisar sua representação em uma Árvore de Sintaxe Abstrata (AST). Um candidato a refatoração tem um par (C; F) onde C é uma Classe *Context* e F é o campo de C que pode ou não ser inicializado como uma instância de *NULL OBJECT*, após a refatoração.

Para identificar os candidatos a refatoração o algoritmo foca em descobrir quais campos podem não ser inicializados em certas instâncias da respectiva classe *Context*, e assim, o uso em métodos *Context* é acompanhado por verificações nulas repetitivas que evitam exceções de ponteiros nulos (NPEs). Os campos da classe que tem essa característica serão, a partir de agora, declarados como Campos Opcionais. Para cada campo opcional, o algoritmo capta instruções condicionais e assim salva e guarda as invocações do método para esse campo. As instruções condicionais citadas são caracterizadas por haver verificações para saber se o campo opcional é nulo, estas verificações serão referidas como GFI-Conditionals (*Guarded Field Invocation Conditional Statements*).

Para melhor entendimento a definição do campo Opcional é, para um campo F, declarado em uma classe *Context*. Ele só é considerado um campo Opcional se houver instâncias da classe *Context* que é feito a omissão na sua inicialização.

O GFI-Conditionals (*Guarded Field Invocation Conditional Statements*) representa uma condicional declarada com no máximo (*if* ou if / else), onde a expressão

condicional é uma verificação simples se um objeto é nulo ou não. A comparação deve ser realizada com um campo da classe *Context*. Um GFI-Conditionals faz a proteção do acesso a campos ou métodos de um campo opcional contra exceções de ponteiros nulos (NPEs). O objetivo do método é a eliminação de variados tipos de GFI-Conditionals por meio do padrão *NULL OBJECT*.

A Figura 1 ilustra um exemplos de variações do GFI-Condicionals. O algoritmo foca nas variações de verificação de objeto nulo.

Figura 1 - OptionalField e as variações do GFI-Condicionals

Fonte: Gaitani *et al.* (2015).

A primeira variação (GFI-Conditional Var. 1) se caracteriza por uma expressão condicional em que se verifica *opcionalField* é diferente de nulo; na estrutura da

condicional tem instruções onde é realizado a invocação de um método do opcionalField. Todos fragmentos que têm essa mesma estrutura são chamados de *FieldInvocationFragment*. Após a refatoração, *FieldInvocationFragment* executa polimorficamente os métodos do *NULL OBJECT*. Temos mv que é um método do *optionalField* que é invocado no FieldInvocationFragment e possui um retorno void. Então o método é marcado pelo algoritmo de identificação e refatoração satisfazendo o predicado *emptyOnNull* (*optionalField*, mv).

A segunda variação tem uma ramificação *else* que têm uma exceção específica da aplicação em relação ao *optinalField* nulo. Essa exceção é chamada como *NullFieldException*. O primeiro *if* começa com um *FieldInvocationFragment* depois pode vir qualquer instrução, com as ramificações *if* e *else* reordenadas e uma condição complementar de verificação nula. Nessa variação durante a refatoração para *NULL OBEJECT* é feito uma ramificação da segunda variante, na primeira variante. O *FieldInvocationFragment* executa os métodos polimorficamente de um objeto nulo e gera uma exceção *NullFieldException* para preservar o comportamento. A exceção é marcada pelo algoritmo de identificação e refatoração.

A terceira variação apresentada tem uma ramificação única que verifica a igualdade do campo *optionalField* com nulo na expressão condicional. Sua estrutura apresenta uma instrução *throw NullFieldException*.

O algoritmo de identificação faz o processamento de cada classe individualmente para identificar as classes *Context*. A entrada recebe uma classe de sistema e a saída um conjunto de possíveis refatorações.

Como o algoritmo processa cada método da classe *Context* e faz a análise das condicionais declaradas no corpo do método. Primeiro é feita a verificação de todas as invocações de métodos declarados para invocações não protegidas no campo opcional. Na próxima etapa, o algoritmo repete as declarações condicionais, declaradas na estrutura do método em busca de uma das quatro variantes do GFI-Condicionais.

Depois é feito uma verificação na GFI-Condicional se estão na pré-condições para a refatoração. As pré-condições para *optionalField* são: i) O componente deve pertencer à estrutura do código do projeto analisado; ii) O componente não pode ser uma interface; iii) O componente não deve herdar de outra classe; iv) Os campos não devem

ser acessados diretamente na classe por meio da referência *optionalField*; v) A declaração do campo opcional na classe Context não deve estar visível para as subclasses de *Context*; vi) Caso o componente seja declarado como uma classe interna de *Context*; o último não deve invocar os métodos do Componente por meio da referência do *optionalField*.

Os *GFI-Condicionals* devem satisfazer às seguintes condições prévias: i) Um *NullFieldException* que é lançado em um *GFI-Condicionals* deve pertencer à base de código do projeto analisado; ii) Os argumentos construtores do *NullFieldException*, invocados no *GFI-Condicional* devem ser conhecidos no momento da compilação; iii) Os métodos invocados na parte *FieldInvocationFragment* de um *GFI-Condicional* não devem ser estáticos; iv) Os métodos invocados no *FieldInvocationFragment* de um *GFI-Condicional* não devem ter *NULL OBJECT* conflitantes associados a eles.

Satisfazendo as condições, a transformação do código envolve a criação de novas classes e a modificação das classes já existentes. O algoritmo segue os seguintes passos: 1) Criação da classe *AbstractComponent*; 2) Refatoração de Componente para uma subclasse *AbstractComponent*; 3) Criação da classe *NullComponent*; 4) Refatoração de Contexto e Eliminação de Declarações Condicionais da GFI.

2.2.3 Método Fundamentado em Meta-programação

O método de Mens e Tourwé é um padrão baseado em papéis que utiliza a técnica de meta-programação declarativa, essa técnica foi empregada para tornar possível manipular programas orientado a objetos em uma linguagem simples e intuitiva, está técnica é utilizada para dar suporte na evolução de um software em termos de padrões de projeto essa abordagem é dividida em quatro atividades: Especificação (para descrever padrões de projeto, suas instâncias e suas restrições), Geração (para gerar código de esqueleto para padrão casos com base nas especificações), Transformação (para especificar as transformações de refatoração que pode ser aplicada a uma determinada instância de desenho), Evolução (para detectar e resolver conflitos durante a evolução de um exemplo de padrão) Mens e Tourwé (2001).

A especificação tem o objetivo de identificar os participantes de um padrão de projeto de forma declarativa, para focar somente na estrutura do padrão (como ele se comporta). Além disso, as especificações de restrições são definidas para que o padrão de projeto seja aplicado de forma efetiva. Para a aplicação das restrições é utilizado o padrão *pattern Constraint*. Geração é alcançada no ambiente de desenvolvimento, onde o desenvolvedor requisita a criação do Padrão de Projeto fornecendo os argumentos apropriados para a criação como ela baseada em papéis deve-se definir os participantes do padrão eles podem ser classes, métodos, relações entre classes na figura 2 é apresentado.

Figura 2 - Definição do padrão

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

Fonte: Mens e Tourwé (2001)

Assim, é gerado um esqueleto do padrão desejado para que o desenvolvedor implemente nesse modelo gerado. As transformações são similares às refatorações, com a diferença que elas não exigem que seja preservado o comportamento. A evolução é onde são resolvidos os conflitos ocasionados pela transformação do código para que os participantes do padrão tenham seus devidos papéis corretamente.

O método de Mens e Tourwé utiliza a meta-programação que é uma linguagem para especificar e raciocinar sobre padrões de projeto e para verificar a conformidade de um software implementar a sua arquitetura pretendida. Neste método foi utilizado SOUL uma linguagem de programação lógica implementada sobre uma linguagem orientada a objetos Smalltalk.

2.3 Ferramentas de Refatoração

Essa seção apresenta a importância das ferramentas de refatoração comentando sobre seus benefícios para os desenvolvedores de software e abordando uma ferramenta de refatoração específica criada por Beluzzo (2018) conhecida como *Refactoring and Measurement Tool* (RMT) na qual os métodos que serão comparados neste trabalho podem ser implementados.

O estudo realizou as refatorações manuais são sujeitas a erros. As ferramentas de refatoração automatizada fazem verificações com exatidão, assim permitem uma refatoração aos desenvolvedores rápida e segura do código. Ge e Murphy-Hill (2014) realizou uma pesquisa em que utilizou oito desenvolvedores de software de variados níveis profissionais e observou que a utilização de uma ferramenta (*GostFactor*) auxilia o processo de refatoração, fazendo com que os erros durante o processo reduzissem em 23,3% em comparação com a refatoração manual dos programadores.

A refatoração com uma ferramenta automatizada, pode trazer efeitos positivos como manutenilibidade, reusabilidade e legibilidade do código (FOWLER *et al*., 1999). A seguir será apresentada a ferramenta proposta por Beluzzo (2018).

2.3.1 Refactoring and Measurement Tool (RMT)

Após a realização do mapeamento, Beluzzo (2018) criou uma abordagem e a implementou gerando uma ferramenta. No trabalho existem vários pontos de extensão como: incorporar novos métodos, aplicar diferentes abordagem de extração de dados e realizar mais avaliações de atributos de qualidade.

A abordagem de Beluzzo (2018) tem como objetivo trazer novas formas de refatoração por meio de funcionalidades adicionais e implementar pontos faltantes encontrados em outros trabalhos que ele fez uma seleção por meio de um mapeamento realizado por ele. A partir da sua abordagem criou a ferramenta *Refactoring and Measurement Tool* (RMT) que contempla mais de um método de refatoração.

A ferramenta possibilita que o processo de refatoração forneça informações necessárias como: identificar quais padrões podem ser aplicados, aceitar ou não a

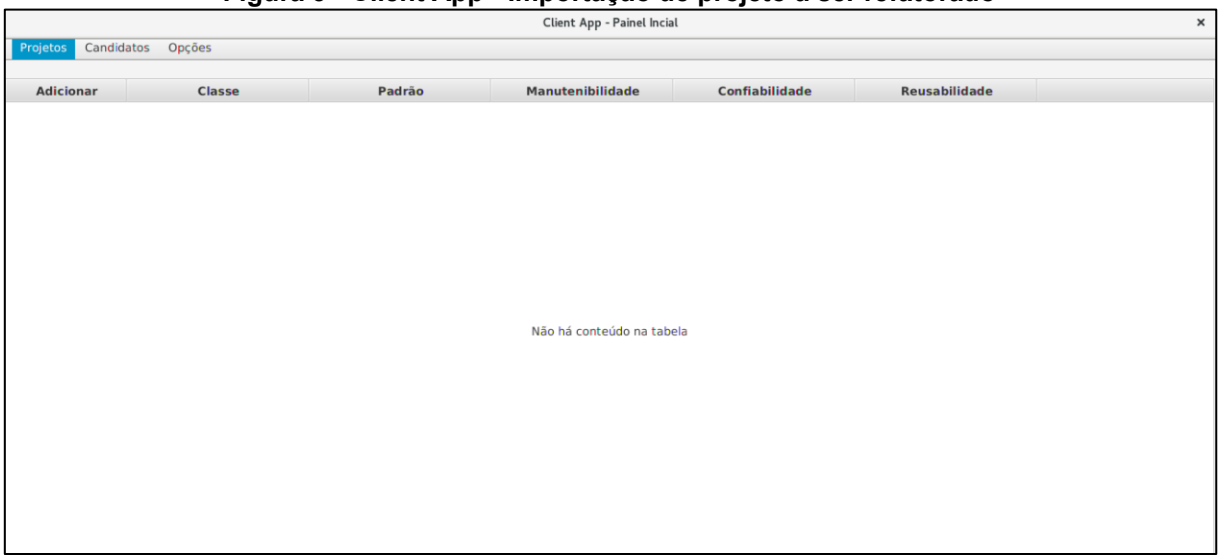
aplicação do padrão, avaliar os benefícios da aplicação do padrão e aplicar efetivamente o padrão escolhido pelo usuário.

A abordagem proposta por Beluzzo (2018) é composta por dois módulos e um ou mais aplicativo(s), que são *Intermediary Service*, *Detection Methods Service*, *Metrics Service*. O *Intermediary Service* tem a responsabilidade de realizar as interações entre os módulos e manter os códigos-fonte do projeto a serem refatorados. *Detection Methods Service* faz a verificação de quais padrões podem ser aplicados. *Metrics Service* faz uma pré-avaliação do candidato a refatoração, antes de ser aplicado a refatoração no código-fonte original.

A abordagem tem como entrada um projeto desenvolvido em orientação a objetos em Java, pois os métodos selecionados em seu mapeamento são voltados para essa linguagem.

A ferramenta contém um módulo externo (*ClientApp*) e é usado como meio de interação com o usuário que apresenta funções Projetos/Importar, Candidatos/Aplicar e Outros/Limpar na Figura 3 apresenta essas funções.

Figura 3 - Client App - Importação de projeto a ser refatorado

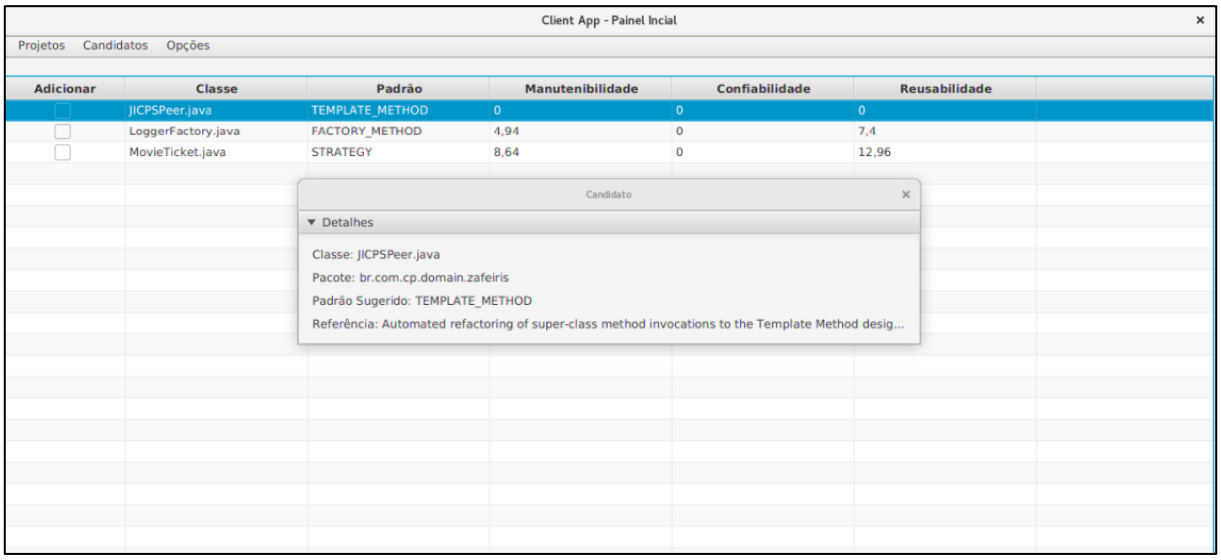


Fonte: Beluzzo (2018).

Ao selecionar um projeto para importação, o projeto é submetido a avaliação e um retorno é apresentado ao usuário e ele visualiza as informações adicionais dos

candidatos a refatoração tais como: nome da classe, pacote, entre outras como podemos ver na figura 4.

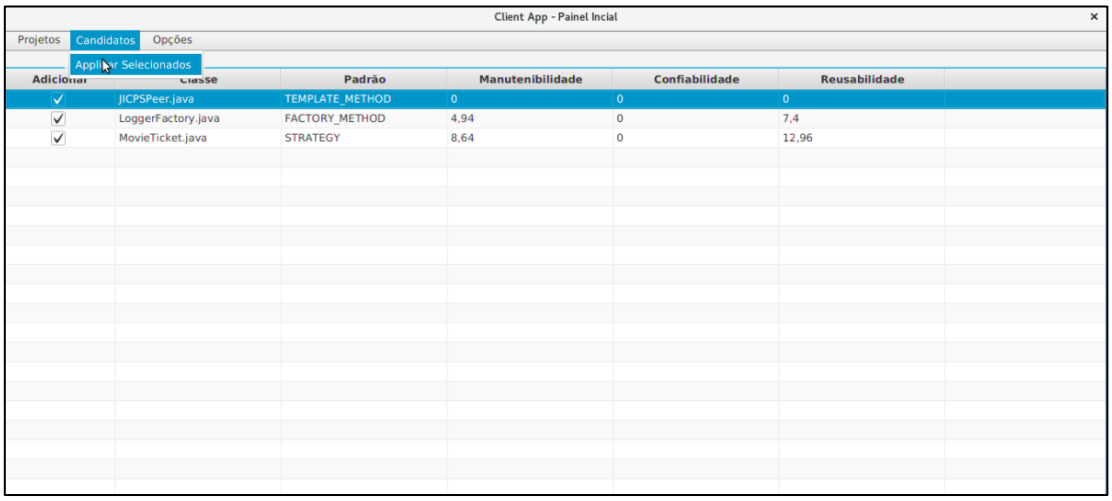
Figura 4 - Client App - Candidatos a Refatoração e Informações Adicionais



Fonte: Beluzzo (2018).

O usuário pode escolher quais padrões vão ser aplicados e quais os candidatos a serão refatorados pela ferramenta dando ao usuário controle do que pode ou não ser alterado na Figura 5 apresenta como realizado essa ação.

Figura 5 - Client App - Aplicação de Candidatos a Refatoração



Fonte: Beluzzo (2018).

A abordagem de foi avaliada considerando 50 projetos que foram selecionados aleatoriamente. Para a avaliação dos projetos foram filtrados apenas por sua linguagem

de programação. Esta avaliação feita por Beluzzo (2018) verificou-se o percentual de projetos, os que apresentaram candidatos a refatorção e o resultado que a avaliação de tributos de qualidade pode ter na seleção de padrões candidatos por parte do usuário.

2.4 Considerações finais do capítulo

Este capítulo apresentou a importância do processo de refatoração e descreveu que a mesma pode ser realizada por meio de técnicas e padrões de projeto. Além disto, descreveu sobre ferramentas de refatoração mais especificamente a *Refactoring and Measurement Tool* (RMT) e seu funcionamento.

Foram descritos três métodos Kim *et al.* (2014) Método de Refatoração Reflexiva R^2, Gaitani *et al*. (2015) Método *Null Object* e Mens e Tourwé (2001) método refatoração que utiliza Meta-programação que são capazes de identificar candidatos a refatoração para padrões de projeto.

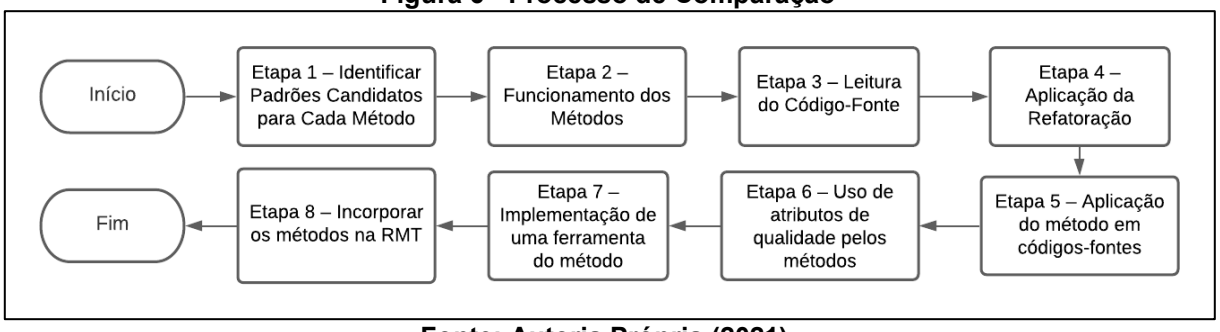
3 COMPARAÇÃO DOS MÉTODOS PARA DETECÇÃO E INSERÇÃO DE PADRÕES DE PROJETO

Este capítulo descreve como o processo de comparação entre os métodos foi realizado. A Seção 3.1 apresenta as etapas do processo de comparação. A Seção 3.2 descreve a identificação de padrões candidatos para cada método. A Seção 3.3 narra o funcionamento interno dos métodos. A Seção 3.4 apresenta como os métodos realizam a leitura do código-fonte. A Seção 3.5 especifica como o método aplica a refatoração. A Seção 3.6 descreve a aplicação prática do método em códigos-fontes. A Seção 3.7 apresenta se o método usa atributos de qualidade. A Seção 3.8 descreve se para o método foi criado uma ferramenta específica. A Seção 3.9 específica como os métodos estudos podem ser incorporados a RMT. A Seção 3.10 apresenta uma análise geral sobre os resultados dos critérios usados para a comparação. Por fim, a última seção relata as considerações finais do capítulo.

3.1 Processo de Comparação

O Processo de Comparação foi dividido em oito etapas conforme é apresentado na Figura 6.

Figura 6 - Processo de Comparação



Fonte: Autoria Própria (2021).

A Etapa 1 descreve quais padrões os métodos conseguem identificar e aplicar. A Etapa 2 apresenta o funcionamento de cada metodologia. A Etapa 3 apresenta como

cada técnica faz a leitura do código-fonte, em sequência na Etapa 4 é descrito como cada método efetua a refatoração,

A Etapa 5 especifica quais foram os resultados da aplicação dos métodos a um código-fonte. A Etapa 6 refere aos atributos de qualidade que foram utilizados para analisar a refatoração efetuada. A Etapa 7 descreve se o método gerou alguma ferramenta para aplicação, por fim na Seção Etapa 8 especifica como esses métodos podem ser incorporados a ferramenta RMT de Beluzzo (2018).

O processo de comparação foi criado com a finalidade de estabelecer pontos comuns e assim facilitar a incorporação dos métodos na ferramenta RMT, nos quesitos de funcionalidades, leitura do código-fonte e aplicação do método.

Os resultados identificados em cada uma das etapas são descritos nas próximas seções.

3.2 Etapa 1 – Identificar padrões candidatos para cada método

A Refatoração Reflexiva R² aplica oito padrões de projeto por completo sendo eles: *Abstract Factory*, *Builder*, *Command*, *Factory Method*, *Flyweight*, *Memento*, *Singleton*, *Visitor*.

O Método de automatização *Null Object* aplica somente padrão *Null Object* para a eliminação de comparação de objetos nulos que não faz parte dos padrões de Gamma *et al.* (1995). Por fim, o método fundamentado em Meta-Programação apresenta um exemplo de *Abstract Factory* e uma parte a implementação dos padrões *Composite* e *Visitor* que causam algumas inconsistências.

O Quadro 3 resume os 23 padrões que foram desenvolvidos por Gamma *et al.* (1995) e quais métodos que foram analisados conseguem detectá-los no código-fonte.

Quadro 3 – Padrões aplicados por cada padrão

Padrões de Projeto de Gamma *et al.* (1995)	**Reflexiva R² Kim *et al.* (2014)**	***Null Object* Gaitani *et al.* (2015)**	**Método de Meta-Programação Mens e Tourwé (2001)**
Abstract Factory	X		X
Adapter			
Bridge			
Builder	X		
Chain of Responsibility			
Command	X		
Composite			X
Decorator			
Façade			
Factory Method	X		
Flyweight	X		
Interpreter			
Iterator			
Mediator			
Memento	X		
Observer			
Prototype			
Proxy			
Singleton	X		
State			
Strategy			
Template Method			
Visitor	X		X

Fonte: Autoria Própria (2021).

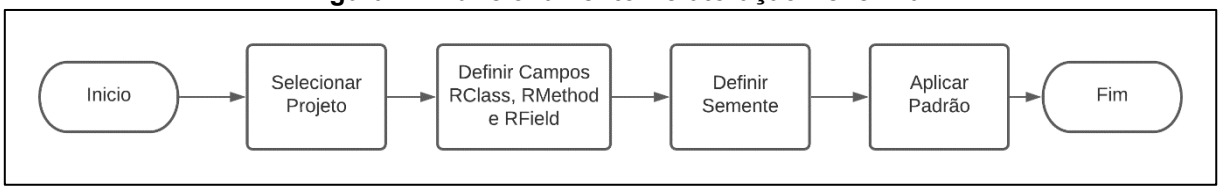
Observa-se que o método da Refatoração Reflexiva é o que mais consegue inserir e aplicar padrões de projeto em código-fonte. Dos 23 padrões propostos pelo Gamma *et al*. (1995) ele 8 são contemplados pelo método. Notou-se também que os padrões de projeto *Abstract Factory* e *Visitor* são contemplados pelos métodos de Kim *et*

al. (2014) e Mens e Tourwé (2001) o *Null Object* como podemos observar não aplica nenhum dos padrões.

3.3 Etapa 2 – Funcionamento dos Métodos

A Refatoração Reflexiva R² apresenta ao programador como selecionar: um campo, método ou classe e faz a invocação da refatoração e seu processo de funcionamento está apresentado na Figura 7.

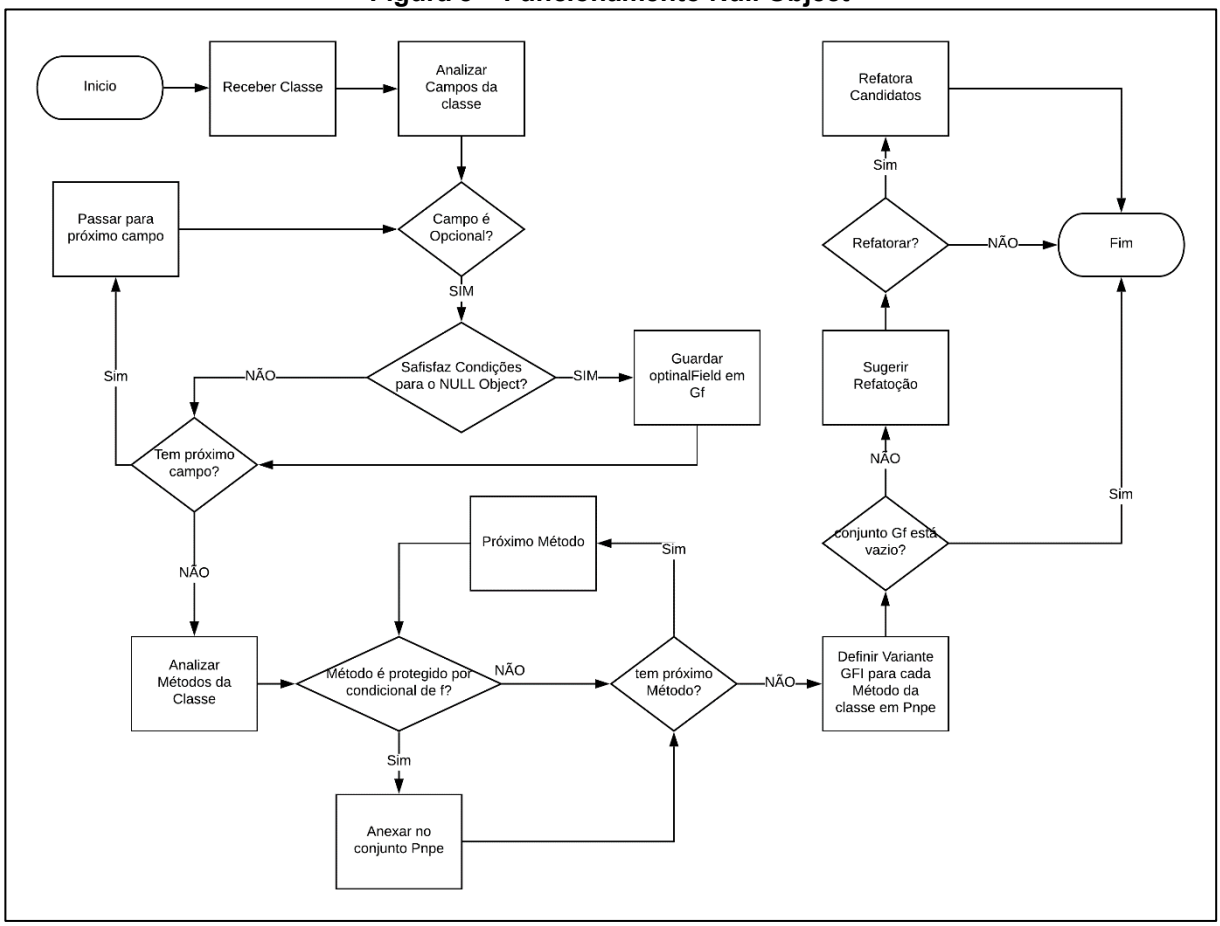
Figura 7 – Funcionamento Refatoração Reflexiva R²



Fonte: Autoria Própria (2021).

Já o *Null Object* tem o objetivo de eliminação de condicionais que fazem verificação de objetos nulos por meio de mecanismos de herança e polimorfismo. O funcionamento do método é exibido na Figura 8.

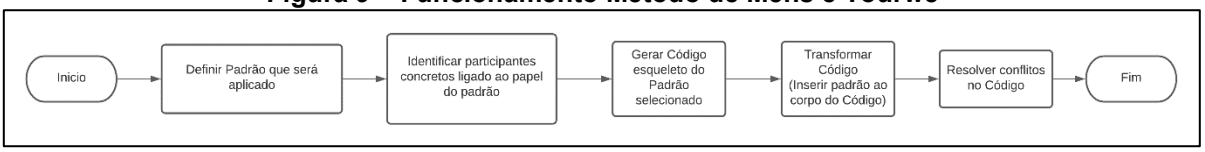
Figura 8 – Funcionamento Null Object



Fonte: Autoria Própria (2021).

O Método de Mens e Tourwé utiliza a técnica de meta-programação declarativa, essa técnica foi empregada para tornar possível manipular programas orientado a objetos. O funcionamento do método é exibido na Figura 9.

Figura 9 – Funcionamento Método de Mens e Tourwé



Fonte: Autoria Própria (2021).

Observou-se que cada um dos métodos tem uma forma diferente de funcionamento. A técnica na R² solicita ao usuário apontar qual método ou classe e

invoca a refatoração. Já no *Null Object* analisa a classe e encontra onde deve fazer a invocação das refatorações e apresenta uma sugestão para o usuário e no método de Mens e Tourwé pode-se concluir que é necessário definir qual padrão será aplicado. A Refatoração Reflexiva tem pontos semelhantes com o método de Mens e Tourwé, pois os dois precisam apontar o padrão que vai ser aplicado, mas na R² é aplicado de forma automatizada e o de Mens e Tourwé deve-se apontar os participas do padrão.

3.4 Etapa 3 – Leitura do Código-Fonte

A Refatoração Reflexiva utiliza a Reflexão do código que contém Classes, Métodos e os Campos criando um conjunto de tabelas *RClass, RMethod, RField*, entre outros, onde cada linha corresponde a uma classe, método ou uma instância de um campo.

Figura 10 - Exemplo de script padrão Visitor

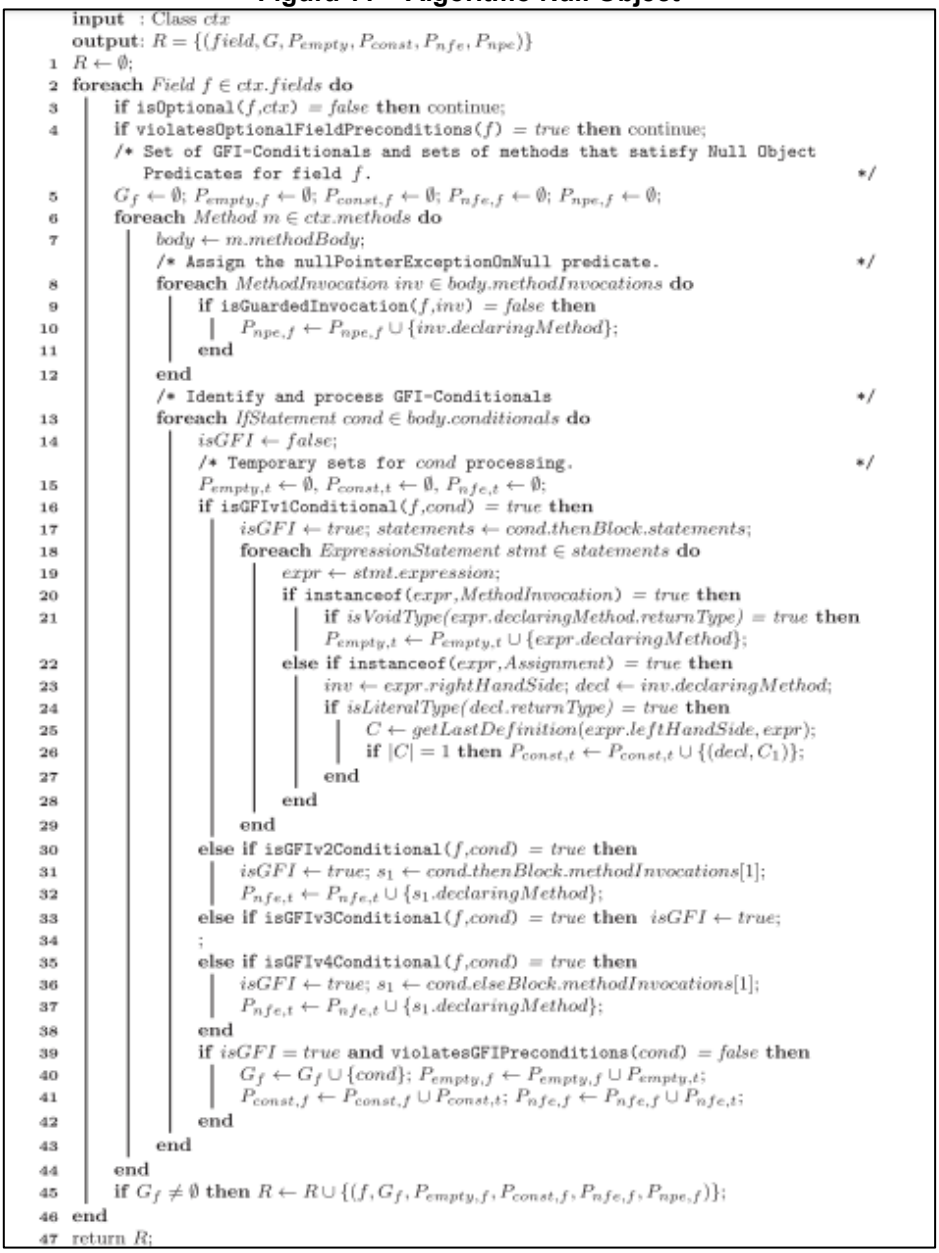
```
RPackage p = RProject.getPackage("R", "p");
RClass   c = p.getClass("p.C");
RMethod  m = c.getMethod("void", "m", null);
m.makeVisitor("Visitor");
```

Fonte: Kim *et al.* (2014).

Na figura 10 na primeira linha do *script* é selecionado um pacote do projeto e na próxima é separado uma determinada classe. Nesta classe, ele irá selecionar uma método para criar o Visitor finalizando o processo.

Null Object faz uma análise estática do código-fonte, em que um algoritmo processa todas as classes do projeto e analisa a representação da Árvore de Sintaxe Abstrata (AST). A Figura 11 mostra como é feita a identificação do campo para refatoração.

Figura 11 – Algoritmo Null Object



```
    input  : Class ctx
    output: R = {(field, G, P_empty, P_const, P_nfe, P_npe)}
 1  R ← ∅;
 2  foreach Field f ∈ ctx.fields do
 3      if isOptional(f,ctx) = false then continue;
 4      if violatesOptionalFieldPreconditions(f) = true then continue;
        /* Set of GFI-Conditionals and sets of methods that satisfy Null Object
           Predicates for field f.                                          */
 5      G_f ← ∅; P_empty,f ← ∅; P_const,f ← ∅; P_nfe,f ← ∅; P_npe,f ← ∅;
 6      foreach Method m ∈ ctx.methods do
 7          body ← m.methodBody;
            /* Assign the nullPointerExceptionOnNull predicate.             */
 8          foreach MethodInvocation inv ∈ body.methodInvocations do
 9              if isGuardedInvocation(f,inv) = false then
10                  P_npe,f ← P_npe,f ∪ {inv.declaringMethod};
11              end
12          end
            /* Identify and process GFI-Conditionals                        */
13          foreach IfStatement cond ∈ body.conditionals do
14              isGFI ← false;
                /* Temporary sets for cond processing.                      */
15              P_empty,t ← ∅, P_const,t ← ∅, P_nfe,t ← ∅;
16              if isGFIv1Conditional(f,cond) = true then
17                  isGFI ← true; statements ← cond.thenBlock.statements;
18                  foreach ExpressionStatement stmt ∈ statements do
19                      expr ← stmt.expression;
20                      if instanceof(expr,MethodInvocation) = true then
21                          if isVoidType(expr.declaringMethod.returnType) = true then
                            P_empty,t ← P_empty,t ∪ {expr.declaringMethod};
22                      else if instanceof(expr,Assignment) = true then
23                          inv ← expr.rightHandSide; decl ← inv.declaringMethod;
24                          if isLiteralType(decl.returnType) = true then
25                              C ← getLastDefinition(expr.leftHandSide, expr);
26                              if |C| = 1 then P_const,t ← P_const,t ∪ {(decl, C_1)};
27                          end
28                      end
29                  end
30              else if isGFIv2Conditional(f,cond) = true then
31                  isGFI ← true; s_1 ← cond.thenBlock.methodInvocations[1];
32                  P_nfe,t ← P_nfe,t ∪ {s_1.declaringMethod};
33              else if isGFIv3Conditional(f,cond) = true then isGFI ← true;
34              ;
35              else if isGFIv4Conditional(f,cond) = true then
36                  isGFI ← true; s_1 ← cond.elseBlock.methodInvocations[1];
37                  P_nfe,t ← P_nfe,t ∪ {s_1.declaringMethod};
38              end
39              if isGFI = true and violatesGFIPreconditions(cond) = false then
40                  G_f ← G_f ∪ {cond}; P_empty,f ← P_empty,f ∪ P_empty,t;
41                  P_const,f ← P_const,f ∪ P_const,t; P_nfe,f ← P_nfe,f ∪ P_nfe,t;
42              end
43          end
44      end
45      if G_f ≠ ∅ then R ← R ∪ {(f, G_f, P_empty,f, P_const,f, P_nfe,f, P_npe,f)};
46  end
47  return R;
```

Fonte: Gaitani *et al.* (2015).

O algoritmo recebe de entrada uma classe e lê campo a campo (linha 2 e 3) para encontrar campos de comparação de nulos e os separa para uma futura refatoração.

O método de Mens e Tourwé utiliza a meta-programação que é uma linguagem para especificar e raciocinar sobre padrões de projeto e para verificar a conformidade de um software implementar a sua arquitetura pretendida. Neste método foi utilizado SOUL uma linguagem de programação lógica implementada sobre uma linguagem orientada a

objetos Smalltalk. O Quadro 4 apresenta os predicados para realizar um mapeamento representacional.

Quadro 4 – Predicado de Mapeamento Representacional

Predicado de Mapeamento Representacional	**Descrição**
class(?C)	C deve ser uma classe
subclass(?P,?C)	a classe c deve ser uma subclasse direta da classe P
concreteSubclass(?P, ?C)	a classe C deve ser uma subclasse concreta da classe P
abstractMethod(?C, ?M)	M deve ser um método abstrato da classe C
concreteMethod(?C, ?M, ?B)	M deve ser um método concreto do corpo B na classe C
classMethod(?C, ?M, ?B)	M deve ser um método da classe com corpo B na classe C
instanceVariable(?C, ?V)	V deve ser uma variável instanciada na classe C
objectCreationBody(?M, ?B, ?C)	O corpo B do método M deve criar uma instância de C

Fonte: Adaptado de Mens e Tourwé (2001).

A leitura do código é realizada por meio de consultas lógicas como é exibido na figura 12. Como exemplo de uma consulta de uma hierarquia pode-se usar a função *hierarchy(Widget,MacButton)* que verifica se a classe *MacButton* é descendente do *Widget* e retorna verdadeiro.

Figura 12 - Exemplo da sintaxe

```
% two examples of logic facts:
subclass(Widget, Button).
subclass(Button, MacButton).
% two examples of logic rules:
hierarchy(?P,?C) :- subclass(?P,?C).
hierarchy(?P,?C) :- subclass(?P,?D), hierarchy(?D,?C).
```

Fonte: Mens e Tourwé (2001).

Percebe-se com que a leitura do código-fonte de cada um dos métodos é realizada classe por classe de uma forma diferente a Refatoração Reflexiva faz a leitura através da reflexão do projeto, *Null Object* faz a leitura através da Árvore de Sintaxe Abstrata (AST) lendo cada classe campo a campo do projeto, e a fundamentada em Meta-Programação a leitura do código é realizada por meio de consultas lógicas três formas distintas de leitura.

3.5 Etapa 4 – Aplicação da Refatoração

Na Refatoração Reflexiva um usuário faz o apontamento para um campo, método ou classe na GUI do Eclipse e invoca uma refatoração / script R^2 como uma refatoração nativa o script é feito utilizando linguagem Java de meta-programação.

No Padrão *Null Object* um candidato a refatoração compreende um par (C; F) onde C é uma classe que tem o papel de *Context* e F é um campo de C que é opcional e pode ser inicializado com uma instância de Objeto Nulo após a refatoração.

No método de Mens e Tourwé a instanciação do padrão pode ser alcançada por meio do ambiente de desenvolvimento de software que pede ao desenvolvedor as informações necessárias, e então aciona uma consulta *createPattern* com os argumentos apropriados.

Identifica-se que na aplicação de cada método a Refatoração Reflexiva aplica o padrão através de um Script e o método de Mens precisa definir os participantes fazer o apontamento aonde aplicar determinado padrão, sendo que o *Null Object* faz a leitura do projeto e apresenta ao usuário aonde aplicar.

3.6 Etapa 5 – Aplicação do método em códigos-fontes

Para adicionar um Visitante dentro do código-fonte foram usados um total de 554 refatorações do Eclipse; para desfazer o Visitante coincidentemente também usou 554 refatorações, considerado o maior experimento de Kim *et al.* (2014) no projeto jak2java.

O *Null Object* foi implementado como parte do *plug-in*. A metodologia foi avaliada experimentalmente quanto à sua solidez por meio da aplicação do método, eficácia e praticidade foram avaliadas baseado nos números de candidatos à refatoração identificados em um conjunto de projetos e o impacto das refatorações aplicado à complexidade ciclomática dos métodos que tiveram as verificações de nulos eliminadas. A Tabela 1 apresenta as propriedades estruturais dos projetos que foram usados como experimentos de testes.

Tabela 1 – Propriedades Estruturais dos Projetos

Projeto	Linhas	Classes	Métodos	Instruções abrangidas (%)
xml Commons ext. 1.4.1	12.856	153	3.168	-
violet 1.0	19.965	370	1.741	-
nutch 1.1	26.500	356	1.741	30
myfaces-impl 2.19	78.457	792	5.438	44
jackrabbit-core 2.9	90.125	978	7.249	61
jmeter 2.9	91.979	1063	7.932	58
apache ant 1.8.2	103.148	1167	9.430	46
jade 4.1	106.036	1569	8.582	-
xerces 2.11.0	112.511	791	8.458	24
jfreechart 1.0.14	143.104	936	10.664	54
xalan 2.7	171.492	1158	9.430	19
batik 1.7	179.122	2288	14.021	-
fop 1.1	197.924	2157	14.965	67

Fonte: Gaitani *et al.* (2015)

Já o método de Mens e Tourwé não apresenta nenhum teste realizado em seu trabalho por conta.

Como podemos ver a os métodos de R^2 e o *Null Object* elaboraram testes em projetos reais para apresentar que os métodos são efetivos e confiantes para os desenvolvedores.

3.7 Etapa 6 – Uso de atributos de qualidade pelos métodos

A Refatoração Reflexiva apresenta atributos de qualidade como: produtividade; escalabilidade e segurança. *Null Object* utiliza como atributos de qualidade a questão de solidez, eficácia e praticidade.

Mens não apresenta nenhum atributo de qualidade envolvendo seu método de refatoração, porque não fez testes devidos aos conflitos estruturais. Ele comenta da necessidade de uma ferramenta mais integrada para avaliar sua abordagem e mostra a necessidade da abordagem criada por eles ser avaliada em casos reais sendo que foram apresentados somente exemplos.

A Produtividade foi a avaliada na questão de tempo de aplicação de um determinado padrão sendo assim os usuários que realizaram as refatorações sem a ferramenta levaram cerca de 107 minutos para concluir, utilizando a R^2 levou 13 segundos. No quesito de escalabilidade foi realizado testes em aplicativos selecionados por eles e assim concluindo sua capacidade de ser escalonável. Na questão de segurança foi analisado de forma que as alterações automatizadas do método é mais adequado do que as manuais, e concluiu que as alterações manuais são de tediosas e conclui que a R^2 deve ser mais seguro que aplicar alterações manuais.

No Padrão *Null Object* a solidez foi avaliado empiricamente com base na correção sintática e na preservação do comportamento dos projetos refatorados. Foi aplicado um *benchmark* em todas as refatorações realizadas e não resultou nenhum erro de compilação dos projetos refatorados. A Eficácia e Praticidade foi avaliada no questão de redução ciclomática deixando o código mais sucinto e compreensível onde foi efetuado as alterações. Concluiu-se que os dois métodos que avaliaram a aplicação dos padrões verifica se a avaliação é efetivada por eles é se a metodologia foi aplicada corretamente sem ocasionar erros e sem a alteração de comportamento do projeto que foi submetido aplicação do método.

3.8 Etapa 7 – Implementação de uma ferramenta do método

Nos métodos comparados dois deles apresentam protótipos e aplicação dos métodos em cenários reais, na Refatoração Reflexiva *Scripts*, que são incorporados as IDEs e o *Null Object*, foi criado um *plugin* para testar os códigos-fontes em um ambiente de desenvolvimento Eclipse. Kim *et al.* (2014) apresenta críticas que quando estavam desenvolvendo perceberam que existia pouco suporte aos *scripts* nas IDEs. Eles desenvolveram em Java com linguagem de meta-programação e as refatoração de JDT (*Java development tools*) nativas no script; método *Null Object* foi implementado como um *plug-in Eclipse*.

O método de Mens e Tourwé não apresenta ferramenta implementada, mas comenta sobre que seria interessante a criação de uma ferramenta integrada ao

ambiente de desenvolvimento onde ela apresenta uma lista de transformações na qual o desenvolvedor deve escolher um determinado padrão e aplica-lo e que pode ser integrada diretamente na sua abordagem. Dois métodos apresentaram a criação de uma ferramenta, mas não disponibilizam acesso gratuito para testá-las.

3.9 Etapa 8 – Incorporar os métodos na RMT

Com base na comparação realizada conclui-se que na primeira etapa do processo de comparação os métodos de Refatoração Reflexiva e o Fundamentado em Meta-Programação tem dois métodos em que ambos aplicam sendo eles Visitor e o Abstract Factory. Observou-se que os dois métodos utilizam meta-programação . Porém, o funcionamento de aplicação é diferente sendo que a Refatoração Reflexiva é totalmente automatizada e o método de Mens é baseada em papéis onde deve-se apontar os participantes do padrão aplicado.

No quesito funcionamento, os três métodos usam meios diferentes para realizar sua refatoração, sendo que R² efetua sua refatoração por meio do apontamento do usuário, o *Null Object* realiza o apontamento de uma classe para iniciar o processo de análise do código e apresenta uma sugestão para refatoração e o da Meta-Programação solicita que usuário faça o apontamento de qual padrão aplicar e não se preocupa com o comportamento do código.

Para a incorporação desses métodos na RMT com base na comparação realizada cada um terá que ser incorporado individualmente, não havendo a reuso de processo.

3.10 Análise geral da comparação dos métodos

Na comparação realizada nesse capitulo verificou-se que os padrões de projeto *Abstract Factory* e *Visitor* são contemplados pelos métodos de Kim et al. (2014) e Mens e Tourwé (2001). Analisando os métodos da Refatoração Reflexiva e de Mens e Tourwé possui pontos semelhantes pois os dois precisam apontar o padrão que vai ser aplicado,.

Já o Null Object analisa a classe e encontra onde deve fazer a invocação das refatorações e apresenta uma sugestão para o usuário. Comparando a forma de leitura dos códigos fontes percebe-se que cada um dos métodos realiza a leitura classe por classe, mas de forma diferente um do outro, e o Null Object faz a leitura por meio da Árvore de Sintaxe Abstrata (AST) lendo cada classe campo a campo do projeto.

Durante a comparação da aplicação da reafatoração percebe-se uma grande diferença entre os métodos no quesito de aplicação de padrões de projeto, o R² aplica o padrão por meio de um *Script* e o método de Mens precisa definir os participantes para fazer o apontamento de onde aplicar determinado padrão, sendo que o Null Object faz a leitura do projeto e apresenta ao usuário onde aplicar.

Os atributos de qualidades foram encontrados somente em dois métodos, pois Mens e Tourwé não realizou testes para avaliar a ferramenta. Os métodos de R² e o Null Object elaboraram testes em projetos reais para apresentar quais os métodos são efetivos e confiantes para os desenvolvedores. Como resultado os dois métodos que avaliaram a aplicação dos padrões aplicaram sem ocasionar erros.

Na implementação de ferramentas a Refatoração Reflexiva realizou-se testes criando *e* no ambiente de desenvolvimento e o método Null Object criou um *plug-in* junto a IDE e o método de Mens e Tourwé não gerou ferramenta. Em relação aos métodos que geraram a ferramenta, ambos não a disponibilizaram gratuitamente para realização de testes. Conclui-se que existem algumas semelhanças entre os métodos, mas como cada método utiliza formas distintas de aplicação dos padrões para incorporação na RMT terá que ser realizada individualmente, não havendo reuso de processo.

3.11 Considerações finais do capítulo

Este capítulo descreveu o processo de comparação usado para realizar a análise comparativa dos três métodos de inserção e detecção de padrões de projeto com o foco de identificar pontos comuns e específicos. A comparação foi realizada criando critérios de comparação tais como: quais padrões de projeto são contemplados pelos métodos, como é o funcionamento dos métodos, entre outros. Como resultado do processo de

comparação percebeu-se que os métodos podem ter padrões que ambos aplicam; analisando seu funcionamento no fluxograma na Figura 6 e Figura 8 comparando tem um fluxo parecido, mas aplicam o padrão de forma destinta, foi observado que quando o método não é testado em códigos-fontes apresentando somente exemplos de aplicação, sendo que não testado apresenta a falta da avaliação da abordagem sem a apresentação de atributos de qualidade. Concluiu-se que cada método têm um processo diferente de funcionamento e não havendo à possibilidade de reuso de requisitos.

A análise identificou: i) quais padrões são cobertos pelos métodos; ii) como é o funcionamento interno dos métodos; iii) como é realizada a leitura do código-fonte; iv) como é realizada a aplicação do padrão de projeto no código-fonte; v) aplicação do método em códigos-fontes; vi) os tributos de qualidade que cada método utiliza; vii) se o método apresenta uma ferramenta; e viii) como incorporar os métodos na RMT. Os pontos comuns identificados entre o método R² e de Mens e Tourwé 2 dois padrões em comum sendo eles o Visitor e o Abstract Factory. O funcionamento interno dos métodos percebeu-se que usam técnicas diferentes, mas a R² e método de Mens eTourwé ambos utilizam para manipular o código a meta-programação, a leitura do código em cada um deles é realizado de forma diferente, assim como a aplicação, no quesito de tributos de qualidade verificou-se que somente a R² e o *Null Object* realizaram testes e o tributo em comum entre eles foi a segurança da aplicabilidade do método. A geração de uma ferramenta pelos métodos percebeu-se que dois deles criarão uma ferramenta o R² e o *Null Object* , mas sem disponibilidade para testes. E por fim para a incorporar a RMT verificou-se que cada método terá que ser introduzido individualmente.

4 CONCLUSÃO

Este trabalho apresentou uma análise comparativa entre métodos na literatura que detectam e inserem padrões de projeto em código-fonte. Foi apresentado cada método explicando seu funcionamento e como cada um aplica os padrões. Kim *et al.* (2014) Método de Refatoração Reflexiva R^2 utiliza a Reflexão do projeto para aplicar padrões, Gaitani *et al*. (2015) (Método *Null Object*) aplica refatorações para a eliminação de comparações de objetos nulos e Mens e Tourwé (2001) utiliza Meta-programação para aplicações de padrões de projeto.

Com as informações obtidas durante o processo de comparação conclui-se que a Refatoração Reflexiva em relação os outros métodos é que abrange a maior parte dos padrões em comparação com o método de Mens e Tourwé que tem dois métodos que são aplicados, o *Null Object* aplica somente um padrão para identificar os padrões foram identificados.

Na verificação do funcionamento dos métodos para melhor entendimento foi elaborado um fluxograma a fim de visualizar o processo de refatoração efetuado pelo método. Os métodos R^2 e o de Mens e Tourwé utilizam meta-programação e apresentam um fluxo parecido. Na implementação de uma ferramenta de refatoração, somente dois métodos apresentam que foi produzida uma ferramenta, mas não disponível para testes. Se as ferramentas estivessem disponíveis poderia ser facilitado a incorporação nos métodos na RMT. O trabalho de Mens e Tourwé comenta sobre o desejo de criação de uma ferramenta que se incorpora métodos de refatoração e percebe-se isso na abordagem criada por Beluzzo (2018).

Concluiu-se com a análise comparativa que cada método utiliza técnicas diferentes para aplicação de padrões de projeto, sendo assim, cada método apresentado nesse trabalho terá que ser incorporado a ferramenta *Refactoring and Measurement Tool* (RMT) individualmente, em que se deve criar abstrações específicas. Neste caso, não haverá reuso de funcionalidades.

4.1 Trabalhos futuros

Com a comparação realizada nesse trabalho, novas pesquisas podem ser realizadas como trabalhos futuros:

- Incorporar os métodos três métodos na ferramenta RMT.
- Realizar a análise de outros métodos que não foram incorporados a ferramentas usando as características de comparação que foram identificadas neste trabalho.

REFERÊNCIAS

BELUZZO, L. **Competências requeridas pelos gestores de Instituições de ensino superior privadas:** um estudo em Ponta Grossa. 2018. 173 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Tecnologia, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

DEREZIŃSKA, A. **A Structure-Driven Process of Automated Refactoring to Design Patterns**. United States, North America: Springer International Publishing, 2017. DOI 10.1007/978-3-319-67229-8_4. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsbas&AN=edsbas.161CDFE4&lang=pt-br&site=eds-live&scope=site. Acesso em: 6 dez. 2021.

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

FOWLER, M. *et al*. **Refactoring: improving the design of existing code.** [S.l.]: Addison-Wesley Professional, 2018.

FOWLER, M. Refactoring lowers the cost of enhancements. 2019. Disponível em: <https://refactoring.com/>. Acesso em: 07 maio 2019.

GAITANI, M. A. G. et al. Automated refactoring to the Null Object design pattern. **Information and Software Technology**, [s. l.], v. 59, p. 33–52, 2015. DOI 10.1016/ j.infsof.2014.10.010. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0950584914002389&lang=pt-br&site=eds-live&scope=site. Acesso em: 7 maio. 2019.

GAMMA, E. **Design patterns : elements of reusable object-oriented software**. [s. l.]: Addison-Wesley, 1995. ISBN 9780201633610. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=cat07269a&AN=utfpr.139550&lang=pt-br&site=eds-live&scope=site. Acesso em: 3 nov. 2020

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

KIM, J.; BATORY, D.; DIG, D. **Scripting Refactorings in Java to Introduce Design Patterns**. UTexas-Austin, Tech. Rep. TR-14-14 (2014). p.24

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

LACERDA, G. et al. **Code Smells and Refactoring: A Tertiary Systematic Review of Challenges and Observations**. [s. l.], 2020. DOI 10.1016/j.jss.2020.110610. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.2004.10777&lang=pt-br&site=eds-live&scope=site. Acesso em: 6 dez. 2021.

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.

TOM MENS; TOM TOURWE. A Declarative Evolution Framework for Object-Oriented Design Patterns. **http://prog.vub.ac.be/Publications/2001/vub-prog-tr-01-08.pdf**, [s. l.], 2001. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsbas&AN=edsbas.E33D27AF&lang=pt-br&site=eds-live&scope=site. Acesso em: 3 nov. 2020.

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

XI GE; QUINTON L. DUBOSE; EMERSON MURPHY-HILL. Reconciling manual and automatic refactoring. **http://people.engr.ncsu.edu/ermurph3/papers/icse12.pdf** , [s. l.], [s. d.]. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=edsbas&AN=edsbas.1D8E5FC5&lang=pt-br&site=eds-live&scope=site. Acesso em: 3 maio. 2020.