Ferramenta RMT: Melhorias em sua Arquitetura

RMT Tool: Architectural in the Improvements

Carlos Eduardo Rodrigues Cardoso*, Simone Nasser Matos†.

RESUMO

A ferramenta Refactoring and Measurement Tool (RMT) tem como objetivo detectar e inserir padrões de projeto em código-fonte escrito na linguagem Java. A execução da refatoração usa vários métodos de refatoração, o que exige muito processamento. Por isto, para identificar melhorias na ferramenta RMT é necessário realizar um estudo em sua arquitetura a fim de levantar quais características podem ser modificadas ou incluídas. Este estudo é apresentado neste artigo e iniciou com uma pesquisa da literatura em relação a métodos de refatoração de código-fonte, destacando os que se baseiam em padrões de projeto. Foram implementados alguns padrões de projetos para esclarecer seu funcionamento, depois foi realizado um estudo mais aprofundado sobre os métodos não clássicos de refatoração com ênfase no método de refatoração *NULL Object*, analisando seu funcionamento, estrutura e formas de avaliação. Como resultado são apresentados 4 (quatro) fatores que podem melhorar a arquitetura da ferramenta RMT sendo eles: introdução de balanceamento de carga, novos atributos de qualidade e métricas de software, utilização de repositório de projetos *GitHub* visando o controle de versões e a inclusão do método de refatoração *NULL Object* a nível de modelo.

Palavras-chave: Refatoração, Padrões de Projeto, Atributos de Qualidade, Controle de Versões

ABSTRACT

The Refactoring and Measurement Tool (RMT) aims to detect and insert design patterns in source code written in Java language. A refactoring run uses multiple refactoring methods, which require a lot of processing. Therefore, to identify improvements in the RMT tool, it is necessary to carry out a study in its architecture in order to determine which features can be modified or included.This study is presented in this article and begins with a survey of the literature regarding source code refactoring methods, highlighting those based on design patterns. Some design patterns were implemented to clarify their operation, then a more in-depth study was carried out on non-classical refactoring methods with emphasis on the NULL Object refactoring method, analyzing its operation, structure and forms of evaluation. As result, 4 (four) factors are needed for to improve the architecture of the RMT tool, namely: introduction of load balancing, new quality attributes and software metrics, use of NULL Object refactoring project repository at the model level.

Keywords: Refactoring, Design Patterns, Quality Attributes, Version Control.

1 INTRODUÇÃO

A refatoração de um código-fonte está relacionada a ação de reorganizar estruturalmente trechos de código de um sistema, sem modificar seu funcionamento externo. Segundo Fowler et al. (1999) é importante refatorar para que as atividades de compreender o código ou inserir uma nova funcionalidade sejam executadas com baixo teor de dificuldade. Um software que é programado sem a preocupação de como o código-fonte está organizado torna complexa a sua manutenibilidade e isso acarreta na diminuição do nível de qualidade

* Bacharelado em Ciência da Computação, Universidade Tecnológica Federal do Paraná, Ponta Grossa, Paraná, Brasil; carloscardoso@alunos.utfpr.edu.br
† Depto. de Informática, Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa; snasser@utfpr.edu.br

atribuído ao projeto. O intuito da refatoração é tornar o código mais flexível eliminando fatos como duplicação de linha de código, redundância de informação, métodos e classes mal estruturadas.

Para que o processo seja executado, a refatoração pode seguir o princípio de técnicas ou baseada em padrões de projetos. Os padrões possuem categorias como: criacionais, estruturais e comportamentais e são descritos por Gamma et al. (2009). Ao aplicar um padrão ao sistema, primeiramente, é necessário compreender bem o contexto do domínio do seu problema, pois utilizar o padrão de projeto onde não há necessidade irá aumentar de maneira desnecessária a complexidade do sistema sem trazer benefício algum (CINNÈIDE; NIXON, 1999, p. 463-472).

Na literatura existem vários métodos baseados em padrões de projeto capazes de realizar a refatoração do código-fonte. Cada método foi implementado de forma individual por seus idealizadores. Beluzzo (2018) definiu um ambiente chamado de *Refactoring and Measurement Tool* (RMT) que reúne mais de um método da literatura capaz de refatorar um código-fonte escrito em linguagem Java, isto ajuda o desenvolvedor pois pode refatorar o seu código usando um único ambiente.

Na RMT, ao inserir um código-fonte como entrada, o software avalia e retorna para o usuário quais classes devem ser refatoradas, quais métodos são recomendados e qual o impacto de cada opção de refatoração na classe. Permite também visualizar de forma percentual o resultado positivo ou negativo da refatoração no código. O impacto é calculado aplicando a combinação entre métricas baseadas em atributos de qualidade.

Com o objetivo de propor melhorias para a ferramenta de refatoração RMT, é apresentado neste artigo um estudo sobre refatoração de software baseado em padrões de projeto e da ferramenta RMT a fim de avaliar quais características podem ser melhoradas visando elevar os atributos de qualidade da ferramenta e sua disponibilização gratuita.

2 MÉTODO DA PESQUISA

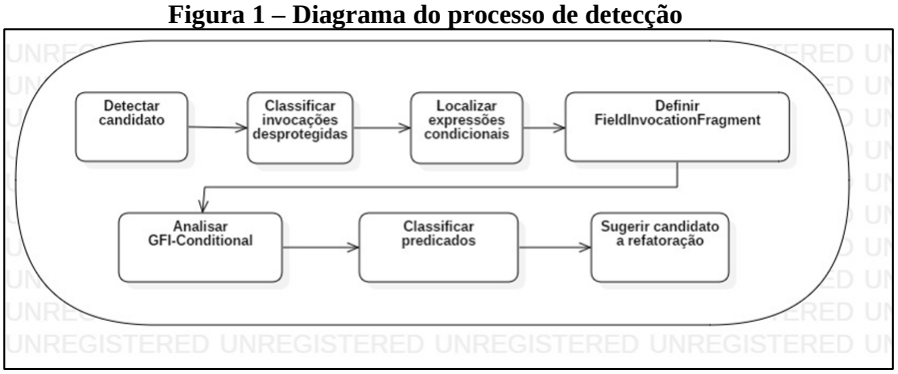
A primeira etapa realizada nessa pesquisa foi o aprimoramento dos conhecimentos em relação a refatoração de código-fonte e padrões de projeto. A refatoração consiste de técnicas pré-definidas que utiliza a ideia de "*bad smells*" para identificar trechos de código que indicam a necessidade de mudanças para aprimorar o seu funcionamento. Fowler (2018) define a refatoração como uma alteração na forma estrutural do software para torná-lo mais compreensível e com custo baixo de modificação sem descaracterizar seu comportamento observável.

Os padrões de projeto (*design patterns*) podem ser descritos como soluções abstratas para problemas recorrentes que aparecem durante o desenvolvimento de software. Os padrões de projeto são compostos por nome, solução para um problema, seu contexto, forma de aplicação e consequências. No livro "Padrões de Projeto – Soluções reutilizáveis de software orientado a objeto" Gamma et al. (2009) descrevem diversos padrões que são divididos em 3 (três) categorias: criacionais, estruturais e comportamentais. No desenvolvimento de um projeto de software muitas vezes nenhum tipo de padrão de projeto é levado em consideração nas primeiras etapas, tornado a evolução do sistema mais complexa. A utilização de padrões de projeto tem o intuito de deixar o código mais flexível aumentando sua reutilização, qualidade, legibilidade e a resiliência a mudanças (GAMMA et al., 2009).

O próximo passo da pesquisa foi compreender como funcionam os padrões de projeto e como autores da literatura unem os dois temas para desenvolver métodos de refatoração de código-fonte baseados em padrões de projeto. Inicialmente foi implementado alguns padrões de projeto para fins de esclarecimento de quais características compunham cada padrão verificando se possuíam semelhanças entre eles. Foram codificados os padrões *Singleton, Abstract Factory, Factory Method* e *Facade*.

Visando compreender como são elaborados os métodos de refatoração baseada em padrões de projeto foi utilizado como objeto de estudo o método proposto por Gaitani et al. (2015), esse método visa eliminar verificações nulas repetidas adicionando um objeto nulo a classe que está sendo refatorada, com o objetivo de melhorar o desempenho do programa. O código-fonte é analisado pelo algoritmo buscando trechos que o valor do campo opcional é manipulado em instruções condicionais (if ou if/else) nulas, esses trechos são denominados de *Guarded Field Invocation Conditional* (GFI-Conditional). Foi utilizado uma abordagem de regras para definir as especificações dos predicados e relações necessárias que formam o vocabulário do método proposto. Para que um candidato a refatoração se transforme em campo opcional (*optionalField*) é necessário atender as pré-condições de refatoração automatizada para o padrão *NULL Object* com o objetivo de evitar que a reestruturação do código seja errônea inserindo novos problemas ou modificando o funcionamento externo do sistema. Essas pré-condições de refatoração são compostas pelas pré-condições de campo opcional e pré-condições de GFI-Conditional. O fluxo do processo de detecção é representado na forma de um diagrama de atividade desenvolvido na ferramenta *StarUML* e está exibido pela Figura 1.

Figura 1 – Diagrama do processo de detecção



Fonte: Autoria Própria (2021)

Como resultado da avaliação do método foi constatado que em média 65,7% das verificações nulas presentes nos projetos puderam ser removidas com a inserção do *NULL Object*. A redução percentual da métrica de complexidade ciclomática variou entre 5% a 66,67% nos projetos refatorados. Essa redução contribui para que o código-fonte fique mais legível proporcionando uma manutenibilidade mais intuitiva do código. Após executar todas as etapas citadas, o estudo foi direcionado para a ferramenta de refatoração RMT com o propósito de entender sua estrutura e funcionamento visando encontrar características que poderiam ser melhoradas. A ferramenta RMT possui a característica extensiva, que visa a inclusão de novos métodos de refatoração, métodos de detecção de candidatos a refatoração e métricas de avaliação. Estruturalmente a ferramenta possuí 4 (quatro) módulos que são responsáveis por interagir com o usuário, esses módulos são denominados *Client App*, *Interaction Service*, *Detection Methods Service* e *Metrics Service* (BELUZZO, 2018).

Após cada incremento na ferramenta um "novo método" é gerado, o que caracteriza que uma nova versão. Utilizar o controle de versões torna o desenvolvimento do sistema mais ágil e simplificado, acompanhando o arquivo e seu histórico de desenvolvimento (OTTE, 2009).

3 RESULTADOS

Como conclusão de pesquisa foram identificadas características que podem ser aprimoradas na ferramenta RMT, nesta seção é descrito os tópicos de balanceamento de carga, controle de versão e repositório de

armazenamento, métricas de softwares e atributos de qualidade e inserção do método de refatoração *NULL Object*.

3.1 Balanceamento de Carga

Como princípio básico de funcionamento o balanceamento de carga utiliza o compartilhamento de recursos de hardware e software de uma determinada rede de computadores visando elevar a escalabilidade de processamento das requisições da aplicação. Utilizar o balanceamento de carga de maneira uniforme em vários processadores tende a melhorar o desempenho de computação (RATHORE; CHANA, 2014, p. 2089-2125). Dentre as abordagens da literatura as mais comumente utilizadas estão as estratégias de balanceamento de carga estático e balanceamento de carga dinâmico.

O balanceamento de carga estático é definido a partir de informações do comportamento médio do sistema, onde as decisões de transferências são independes em relação ao sistema como um todo, sendo recomendada para aplicações que possuem um comportamento consistente e previsível (PADUA, 2011). A abordagem dinâmica possui um melhor desempenho em relação a estática e sua implementação é considerada complexa, o que dificulta a utilização desta forma de balanceamento (KAMEDA, 2000).

Gurka Júnior (2019) realizou um levantamento de algoritmos comuns na literatura que realizam o balanceamento de carga estático e dinâmico. Dentre os principais algoritmos para balanceamento estáticos estão: i) *Round Robin* e algoritmos randômicos, que divide as tarefas de forma igualitária entre os processadores; ii) Algoritmo de gerenciamento central, onde um ponto central do sistema é responsável por dividir as tarefas entre os processadores levando em consideração para a escolha o processador com a menor carga no momento da distribuição; iii) Algoritmo limiar, onde os processadores são identificados em 3 (três) categorias sendo elas subcarregado, médio e sobrecarregado. Os algoritmos de balanceamento de carga dinâmicos mais comuns são: i) Algoritmo de fila central, que gerência as tarefas a serem processadas por meio de uma fila onde a primeira tarefa a entrar é a primeira a sair; ii) Algoritmo de fila local, onde a fila de requisições é alocada estaticamente até atingir um nível mínimo definido pelo usuário. Esse algoritmo possui alta complexidade e custo computacional, o que torna sua implementação proibitiva.

Como a ferramenta RMT trabalha com mais de um método de refatoração que detecta o candidato a refatoração, avalia seu impacto e depois refatora (caso seja a escolha do usuário), é possível introduzir o balanceamento de carga dentro dos módulos "*Metrics Service*" e "*Detection Methods Service*" com o objetivo de paralelizar as etapas de cada método, gerando assim, um tempo de resposta menor ao usuário.

3.2 Controle de Versões e Repositório de Armazenamento

Para gerenciar de maneira adequada a evolução do software e de sua documentação, recomenda-se utilizar um sistema de controle de versões adequado para a ferramenta. Visando atender esse objetivo foi criado um repositório no *GitHub* que contém os arquivos da ferramenta para que desenvolvedores possam acessar e contribuir com a evolução do projeto. O repositório da ferramenta pode ser acessado por meio do *link* (https://github.com/Lesic-UTFPR/Ferramenta-RMT.git).

3.3 Métricas de Software e Atributos de Qualidade

Os atributos de qualidade utilizados até o momento pela ferramenta são: Manutenibilidade, Confiabilidade e Reusabilidade. Dentre as métricas de softwares utilizadas na RMT se pode elencar a Profundidade da Árvore de Herança (PAH), Complexidade Ciclomática (CC) e Tamanho do Programa em Linhas de Código (TPLC).

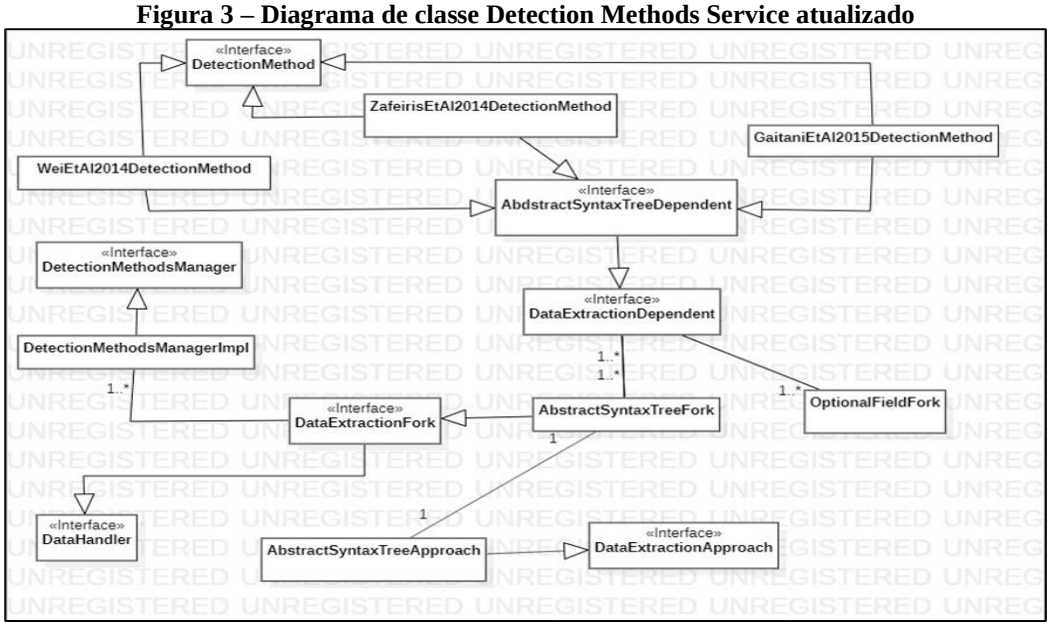
Como outra proposta de melhoria a ferramenta RMT é possível realizar a inclusão de novos atributos de qualidade de softwares como usabilidade, funcionalidade, eficiência, solidez e praticidade. Utilizar mais atributos de qualidade tornará a avaliação da refatoração do código-fonte mais refinada, resultando em um *feedback* mais adequado para o usuário.

Como a ferramenta é baseada na correlação entre atributos e métricas, é necessário relacionar esses novos atributos com novas métricas de software. Dentre as métricas ainda não implementadas na ferramenta se tem número de mensagens de erro e extensão do manual do usuário, tais métricas possuem relação com o funcionamento da ferramenta e com o objetivo de melhorar seu gerenciamento.

3.4 Inserção do Método de Refatoração NULL Object

Para a introdução do método *NULL Object* (GAITANI *et al*., 2015) a nível de modelo na RMT é necessário incrementar o diagrama de classe do "Detection Methods Service" que representa os métodos que estão presentes na ferramenta juntamente com a sua forma de extração de candidatos a refatoração. A ferramenta já possuía os métodos de Zafeiris et al (2014) e Wei et al. (2016), após a inclusão, o método de Gaitani *et al.* (2015) também passa a fazer parte dos métodos presentes na ferramenta de refatoração como se pode observar na Figura 3, que foi criado a partir da ferramenta *StarUML* (Unregistered). Outra mudança necessária no diagrama do "Detection Methods Service" é a inclusão da classe responsável por identificar os possíveis que podem ser substituídos durante o processo de refatoração. O método de Gaitani *et al.* (2015) tem como dependência o "OptionalFieldFork", que encontra campos opcionais que podem ser substituídos por instâncias nulas.

Figura 3 – Diagrama de classe Detection Methods Service atualizado



Fonte: Autoria Própria (2021)

4 CONCLUSÃO

Um sistema sem atualizações tende a não acompanhar o avanço da tecnologia. Como resultado desta pesquisa foi possível identificar 4 (quatro) melhorais importantes que agregariam qualidade a ferramenta RMT. O balanceamento de carga vai permitir melhorar atributo eficiência, introduzir novas métricas de softwares e atributos de qualidade visa aumentar a confiabilidade nos resultados antes da ferramenta aplicar os padrões de

projeto. Efetuar o controle de versões juntamente com um repositório, torna a manutenibilidade da ferramenta mais intuitiva. Por fim, introduzir um novo método de refatoração além de aumentar o conjunto de opção agrega em melhorias dos atributos usabilidade e funcionalidades da ferramenta RMT.

AGRADECIMENTOS

Agradeço a Fundação Araucária por me auxiliar financeiramente durante o desenvolvimento desta pesquisa. Também gostaria de agradecer os membros da Universidade Tecnológica Federal do Paraná, em especial minha orientadora Dra. Simone Nasser Matos por todos os ensinamentos e conselhos passados a mim, com sabedoria e gentileza.

REFERÊNCIAS

BELUZZO, Luan Bukowitz. **Abordagem para avaliar e detectar pontos de inserção e aplicação de padrões de projeto em código-fonte**. 2018. 101f. Dissertação (Mestrado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

CINNÉIDE, Mel Ó.; NIXON, Paddy. **A methodology for the automated introduction of design patterns**. IEEE International Conference on Software Maintenance, 1999. p. 463-472.

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

FOWLER, Martin. **Refactoring: improving the design of existing code**. Addison-Wesley Professional, 2018.

GAITANI, Maria, Anna, 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. Boston: Addison Wesley**, 2009.

GURKA JÚNIOR, Mácio José**. miRQuest 2**: solução computacional para integração de ferramentas de predição de micro RNA utilizando balanceamento de carga. 2019. 56f. Dissertação (Mestrado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

KAMEDA, Hisao et al. **A performance comparison of dynamic vs. static load balancing policies in a mainframe-personal computer network model**. IEEE Conference on Decision and Control. v. 2, p. 1415-1420, 2000.

OTTE, Stefan. **Version control systems**. Computer Systems and Telematics, p. 11-13, 2009.

PADUA, David. **Encyclopedia of Parallel Computing**. Springer Science & Business Media, 2011.

WEI, Liu et al. **Automated pattern-directed refactoring for complex conditional statements**. Journal of Central South University, v. 21, n. 5, p. 1935-1945, 2014.

RATHORE, Neeraj; CHANA, Inderveer. Load balancing and job migration techniques in grid: a survey of recent trends. **Wireless personal communications**, v. 79, n. 3, p. 2089-2125, 2014.

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