ANÁLISE DO IMPACTO DA REFATORAÇÃO NA MANUTENIBILIDADE DO SOFTWARE

ANALYSIS OF THE IMPACT OF REFACTORING ON SOFTWARE MAINTENANCE

Arthur Gonçalves Barboza [1]
Ricardo Petri Silva [2]

RESUMO

Atualmente o desenvolvimento de software envolve inúmeros processos, tais como planejamento e análise, acompanhamento das etapas de implantação, aplicação de testes e validação de requisitos funcionais e não funcionais. Tais processos presentes no ciclo de vida do desenvolvimento de software desempenham um papel fundamental na qualidade do código fonte do sistema. O presente trabalho busca apresentar as relações entre refatoração e seu impacto na manutenibilidade do software através da aplicação dos conceitos mencionadas anteriormente e métricas orientadoras. Os resultados obtidos sinalizam benefícios com relação à aplicação da refatoração, no entanto algumas observações nos fazem questionar sobre a eficácia do método empregado de forma isolada assim como a utilização do mesmo em quaisquer cenários. Este trabalho apresenta seu foco voltado à qualidade no aspecto de manutenibilidade e é dedicado à desenvolvedores de software, profissionais de Quality Assurance (denominados de QA), gestores e líderes de equipes de software.

Palavras-chave: software; teste; qualidade; manutenibilidade; refatoração.

ABSTRACT

Currently software development involves numerous processes, such as planning and analysis, monitoring of implementation stages, application of tests and validation of requirements functional and non-functional. Such processes present in the software development life cycle play a fundamental role in the quality of the system's source code. The present work seeks to present the relationships between refactoring and its impact on software maintainability through the application of the previously mentioned concepts and guiding metrics. The results obtained signal benefits in relation to the application of refactoring, however some observations in raise questions about the effectiveness of the method used in isolation, as well as the use the same in any scenarios. This work focuses on quality in terms of maintainability and is dedicated to

[1] Discente do curso de Engenharia de Software do Centro Universitário Filadélfia (UniFil).
[2] Docente do do curso de Engenharia de Software do Centro Universitário Filadélfia (UniFil).

software developers, Quality Assurance professionals (called QA), managers and leaders of software teams.

Keywords: software; test; quality; maintainability; refactoring.

1 INTRODUÇÃO

Atualmente muitos recursos são investidos na garantia e controle de qualidade do sistema. Testes de software são tratados como uma etapa crucial (Jamil *et al*., 2016) e por vezes representam até mesmo 50% do tempo de desenvolvimento (Pan, 1999). Esforços são feitos para garantir que o sistema possa atender os requisitos especificados assim como os desenvolvedores possam atuar com eficácia no desenvolvimento do mesmo, minimizando o crescimento gradual do custo de manutenção do sistema (Ren *et al*., 2011).

Ainda assim, muitos desenvolvedores passam a dedicar grande parte de seu tempo em atividades de depuração e compreensão do sistema, devido a alta complexidade do mesmo (Duc, 2017), o qual pode ocasionar em atrasos no desenvolvimento de novas funcionalidades e na entrega final do software.

Robert Martin (2008) afirma que devido à prazos urgentes e delimitadores, muitos profissionais se encontram necessitados de desenvolver de maneira ágil funcionalidades ou alteração no código fonte que, por sua vez, é alheio ao planejamento e análise para a construção de uma boa estrutura e preservação da manutenibilidade, promovendo um sistema custoso de trabalhar e difícil de alterar já ao curto prazo.

Este trabalho tem como objetivo realizar um levantamento de informações sobre as correlações do crescimento da complexidade do software com a qualidade no quesito de manutenibilidade, seus impactos nas tarefas atreladas a desenvolvimento e manutenção, e como a refatoração podem auxiliar para mitigar estes problemas e melhorar a qualidade do sistema, proporcionando um bom design ao mesmo. Para alcançar esse objetivo, o primeiro passo foi a obtenção de métricas de manutenibilidade de módulos de e-commerce Magento 1. O segundo passo consiste na aplicação de técnicas de refatoração. Por fim, no último passo foi realizada a coleta das métricas após a refatoração, a qual nos permitiu realizar uma análise a

fim de compreender os impactos da mesma. Importante mencionar que os sistemas analisados se encontram no paradigma Orientado a Objetos, além de serem de natureza modular dependente de um sistema base.

A estrutura do trabalho apresenta primeiramente a seção de fundamentação teórica a qual consiste dos conceitos e temas básicos para a compreensão da experimentação, seguido pelo estado da arte o qual apresenta pesquisas recentes e relevantes da área, seguido pela seção da metodologia e procedendo para a experimentação, que antecede a conclusão da pesquisa.

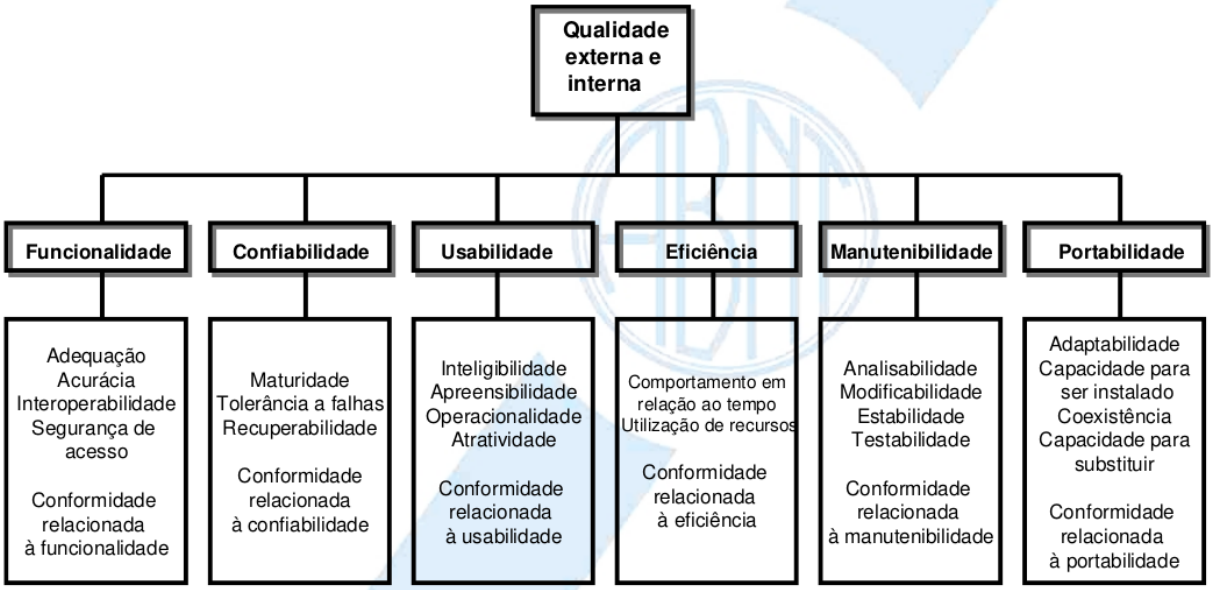
2 QUALIDADE DE SOFTWARE

Sobre os conceitos de qualidade, a NBR ISO 9000:2015 (Associação Brasileira de Normas Técnicas (ABNT), 2015) define qualidade como um conjunto de processos, comportamentos e atividades com o intuito de agregar valor satisfazendo as necessidades do cliente.

Sommerville (2018, p. 457) reafirma este conceito na área de Engenharia de Software ao afirmar que a qualidade do software baseia-se em grande parte em suas características não funcionais as quais refletem na experiência do usuário e ainda conclui: "Portanto, entendemos que a qualidade de software não implica apenas se a funcionalidade de software foi corretamente implementada", se referindo aos atributos de qualidade que por vezes está relacionado com a experiência do usuário. Sommerville também afirma que em um projeto é impossível que todos os atributos de qualidade sejam otimizados e que portanto, deve ser definido os atributos críticos do projeto a fim das partes interessadas trabalharem em conjunto para promover os mesmos.

Conforme demonstrado na figura 1, a ISO/IEC 9126-1 (International Organization for Standardization (ISO), 2001) apresenta um modelo de qualidade composto principalmente por usabilidade, eficiência, manutenibilidade, portabilidade, confiabilidade e outros atributos, cada um podendo ser fragmentado e aprofundado em subatributos os quais se encontram correlatos com áreas mais específicas da Engenharia de Software.

Figura 1 – Modelo de Qualidade para Produto de Software

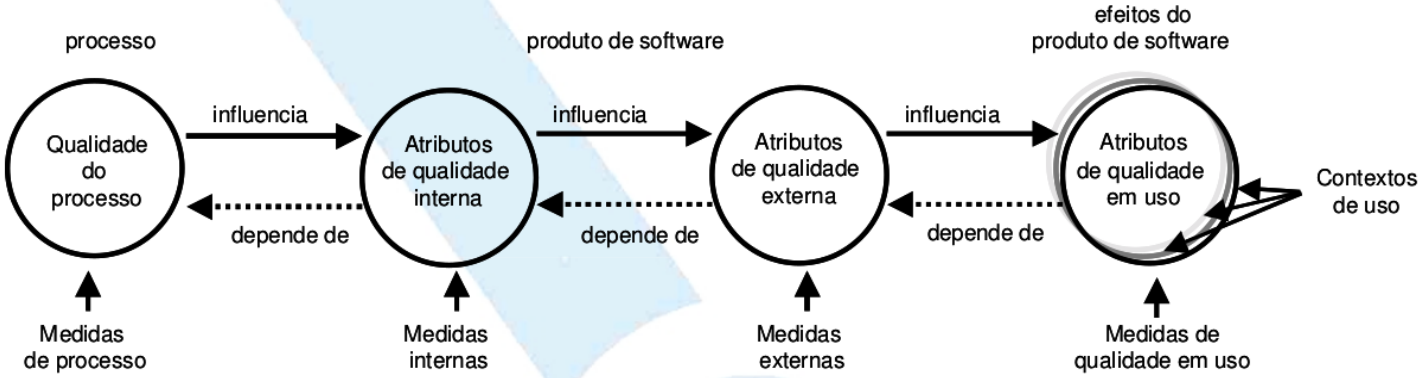


Fonte: International Organization for Standardization (ISO), 2001

Além disso, esta norma também define os conceitos de qualidade interna e externa, os quais podemos expressar suas definições respectivamente como: qualidade do sistema no quesito de código e manutenibilidade, design e arquitetura, sem que seja perceptível através do uso do mesmo; qualidade do sistema no quesito usabilidade, performance e interação com o usuário, assim como segurança e integridade dos dados.

A qualidade interna é por sua vez um fator determinante para a qualidade externa, tendo em vista que a qualidade externa é resultado do código o qual deve ser arquitetado e modelado atendendo os requisitos definidos, e o mesmo é resultado de processos bem definidos, envolvendo atividades de Engenharia de Software como diagramação e padronização de componentes conforme visualizado na Figura 2.

Figura 2 – Impacto da Qualidade no Sistema



Fonte: (International Organization for Standardization (ISO), 2001)

A imagem acima apresenta a cadeia de dependência entre os atributos de qualidade, a qual demonstra a relação entre qualidade interna, qualidade externa e qualidade em uso.

3 REFATORAÇÃO

Se tratando de atividades para manutenção da qualidade interna do software, Fowler (1999) nos apresenta e define refatoração como:

> Processo de mudança de um sistema de software de tal forma que não altere o comportamento externo do código e ainda melhore sua estrutura interna. É uma maneira disciplinada de limpar o código minimizando as chances de introduzir bugs. Ao refatorar você está melhorando o design do código depois do mesmo já ter sido escrito. (Fowler, 1999, p. 9) (Tradução livre pelos autores)

Fowler (1999, p. 49) aponta que a refatoração deve ser aplicada sempre ao realizar uma tarefa atrelada à correção de bugs e manutenção do software, mesmo que em baixa intensidade, mantendo a qualidade do sistema com o decorrer do tempo.

O mesmo também aponta a necessidade de aplicar métodos de refatoração quando o desenvolvedor se deparar com dificuldade em implementar funcionalidades devido a estrutura de código presente. Em outras palavras, um código bem estruturado irá proporcionar ao desenvolvedor, agilidade e velocidade na

implementação de novas funcionalidades, e o oposto também é perceptível.

Fowler (1999, p. 49) afirma que um bom design deve promover velocidade no desenvolvimento do sistema, e que em casos opostos irá retardar o desenvolvimento do mesmo pois o desenvolvedor precisará corrigir bugs e levará mais tempo para implementar alterações por encontrar códigos duplicados.

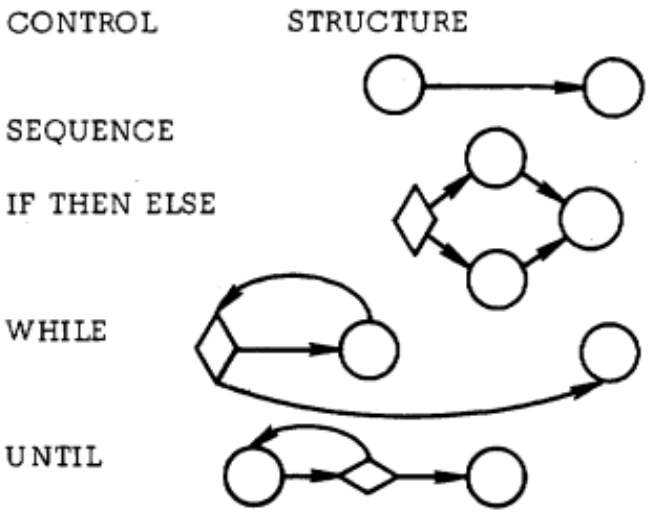
Além da refatoração, Fowler também menciona a importância da aplicação de testes automatizados por parte do desenvolvedor : segundo o mesmo, os testes são essenciais pois mesmo seguindo processos detalhados e estruturados de código, os profissionais são passíveis a cometer erros no desenvolvimento do sistema. A abordagem de Fowler apresenta conformidade com as práticas do TDD (Test Driven Development, (Beck, 2002)), o qual primeiramente implementa os casos de teste a definir o comportamento esperado dos componentes do software e em seguida implementa os métodos do componente, dando sequência à refatoração para promover a qualidade do sistema.

4 COMPLEXIDADE CICLOMÁTICA

A complexidade ciclomática é um relevante parâmetro para ser levado em consideração ao estudar a necessidade de refatorar um sistema. Apresentado por McCabe (1976), seu estudo apresenta a necessidade da análise matemática da complexidade de um componente em um sistema, através da metrificação dos possíveis caminhos e instruções presentes em um software, o qual pode-se utilizar para obter valores quantitativos que descrevem a complexidade do código desenvolvido.

McCabe faz uso de grafos representando a estrutura e fluxograma de um procedimento ou conjunto de instruções de um código. Através da quantidade de instruções e a quantidade de caminhos possíveis obtém-se um número positivo inteiro que representa a complexidade do sistema, assim como os esforços necessários para testar o mesmo. Quanto maior o valor obtido pela equação, mais complexo é o sistema e mais difícil será de obter a cobertura de todas as ramificações possíveis.

Figura 3 – Estrutura de Controle de Fluxo



Fonte: Mccabe (1976)

Conforme podemos visualizar na Figura 3, a complexidade ciclomática utiliza como base o estudo de fluxograma dos sistemas, e faz uso da fórmula de McCabe conforme descrito na equação 1 abaixo, onde 'C' é o valor de complexidade, 'E' é o número de ligações entre os nós e 'N' é a quantidade de nós.

$$C = E - N + 2.$$

Através do estudo apresentado por McCabe podemos calcular e quantificar a complexidade da estrutura dos componentes, mas além de sua estrutura é necessário observar outros fatores minuciosos no código, tais como o quão descritivo o mesmo é, e a dificuldade a qual o desenvolvedor irá enfrentar para compreender cada função envolvida com o sistema. Esses fatores são discutidos no Clean Code (Martin, 2008) o qual apresenta boas práticas para um código legível e de fácil compreensão, abordando técnicas como nomenclatura de variáveis, funções bem descritivas, e a abolição do uso excessivo de comentários.

5 ESTADO DA ARTE

Com relação a refatoração na atualidade, Chávez (2017) realizou um estudo para identificar as relações entre atributos internos de qualidade (International

Organization for Standardization (ISO), 2001) e métodos de refatoração. O mesmo encontrou no material bibliográfico, estudos realizados por Cedrim (2016) apresentando que a refatoração pode ser categorizada pela ocorrência, adição e remoção de estruturas mau formadas. Chávez por sua vez afirma que apenas a melhoria da estrutura interna do *software* não é o bastante, sendo necessário aplicar vários processos de refatoração ao longo dos componentes relacionados para que a estrutura mal formada seja de fato removida e consequentemente atributos de qualidade interna sejam melhorados.

Em pesquisas realizadas por Silva (2016), foi identificado que a motivação da refatoração pelos desenvolvedores, está atrelada principalmente a alteração de requisitos do projeto e em menor grau estruturas mal definidas. Segundo Chávez, as pesquisas de Silvia não apontam diretamente uma relação entre refatoração e qualidades internas, mas suas conclusões sugerem um grau de associação.

Chávez também afirma existir um consenso entre desenvolvedores na aplicação da técnica de extração de métodos duplicados no código fonte, e conclui sua seção de trabalhos relacionados com refatoração e qualidade de software, de que apesar de não se tratar de um objetivo concreto e final dos desenvolvedores, os mesmos apresentam preocupação com atributos de qualidade interna do código.

O estudo de Chávez, além de apresentar material bibliográfico com artigos atuais, discussões recentes e uma fundamentação sólida sobre refatoração e atributos de qualidade interna, este também realizou um levantamento buscando identificar e quantificar os impactos nos atributos de qualidade perante técnicas de refatoração, apresentando em sua conclusão valores os quais reforçam a tese do presente trabalho: Chávez fez seus estudos com 23 projetos open-source em Java, apresentando 29.303 operações de refatoração, concluindo que 94% das operações de refatoração estavam atreladas a pelo menos um atributo de qualidade interna, além disso em 65% dos casos o atributo de qualidade interna apresentava uma melhora e os 35% restantes apresentavam uma estagnação, ou seja, não apresentava uma degradação do atributo. Em outras situações as quais a refatoração se faz presente como atividade secundária (a atividade primária por sua vez, sendo a implementação de um caso de uso ou consertar uma falha), sua eficácia se fez presente em 55% dos casos com apenas 10% apresentando uma degradação de atributo de qualidade

interna.

Na Figura 4 é apresentado as métricas identificadas por Chávez nos estudos levantados, assim como suas descrições e em qual atributo de qualidade a métrica se encontra direcionada.

Figura 4 – Métricas Utilizadas para Analisar Atributos de Qualidade Interna

Attribute	Software Metric	Metric Description
Cohesion	Lack of Cohesion of Methods 2 (LCOM2) [10]	Number of pairs of methods that do not share attributes, minus the number of pairs of methods that share attributes
	Lack of Cohesion of Methods 3 (LCOM3) [23]	Number of disjoint components in the graph that represents each method as a node and the sharing of at least one attribute as an edge
	Tight Class Cohesion (TCC) [5]	Ratio of number of similar method pairs to total number of method pairs in the class
Coupling	Coupling Between Objects (CBO) [10]	The number of classes to which a class is coupled
	Fan-in (FANIN) [17]	The number of other classes that reference a class
	Fan-out (FANOUT) [17]	The number of other classes referenced by a class
	Coupling Intensity (CINT) [22]	The number of distinct operations called by the measured operation
	Coupling Dispersion (CDISP) [22]	The number of classes in wich the operations called from the measured operation are defined by
Complexity	Cyclomatic Complexity (CC) [27]	Measure of the complexity of a module's decision structure.
	Weighted Method Count (WMC) [10]	The sum of Cyclomatic Complexity [27] of all methods declared in the given class
	Essential Complexity (Evg) [27]	Measure of the degree to which a module contains unstructured constructs
	Paths (NPATH) [32]	Number of unique paths though a body of code, not counting abnormal exits or gotos
	Nesting (MaxNest)[25]	Maximum nesting level of control constructs
Inheritance	Depth of Inheritance Tree (DIT) [10]	The depth of a class as the number of its ancestor classes
	Number Of Children (NOC) [10]	The number of direct descendants (subclasses) of a class
	Base Classes (IFANIN) [12]	Number of immediate base classes
	Override Ratio (OR) [22]	The number of methods of the measured class that override methods from the base class, divided by the total number of methods in the class
Size	Lines of Code (LOC) [25]	The number of lines of code excluding white spaces and comments
	Lines with Comments (CLOC) [25]	Number of lines containing comment
	Statements (STMTC) [25]	Number of statements
	Classes (CDL) [25]	Number of classes
	Instance Variables (NIV) [25]	Number of instance variables
	Instance Methods (NIM) [25]	Number of instance methods
	Weight of Class (WOC) [22]	The sum of "funtional" public methods divided by the total number of public members
	Number of Public Attributes (NOPA) [22]	The number of publics attributes of a class

Fonte: Chávez *et al.* (2017)

Entre as métricas de complexidade apresentadas no trabalho de Chávez, se faz presente a Complexidade Ciclomática de McCabe (1976) e o Acoplamento Entre Objetos (Coupling Between Objects, CBO), que será discutido posteriormente na seção de Experimentação em Análise Pré-Refatoração.

O estudo realizado por Guilherme Lacerda (2020) apresenta informações as quais são muito próximas do tema abordado nessa pesquisa. O mesmo tem como objetivo realizar um levantamento sobre as técnicas e ferramentas de refatoração, code smells e a relação de qualidade de software com code smells. Pode-se notar que na pesquisa de Lacerda, as técnicas de refatoração que foram destacadas pelo uso frequente em seus estudos foram "Move Method"e "Extract Method", sendo esta última também aplicada na experimentação deste trabalho.

A pesquisa de Lacerda também identificou desafios relacionados com o tema, sendo estes a identificação eficiente dos smells, a escolha da técnica de refatoração

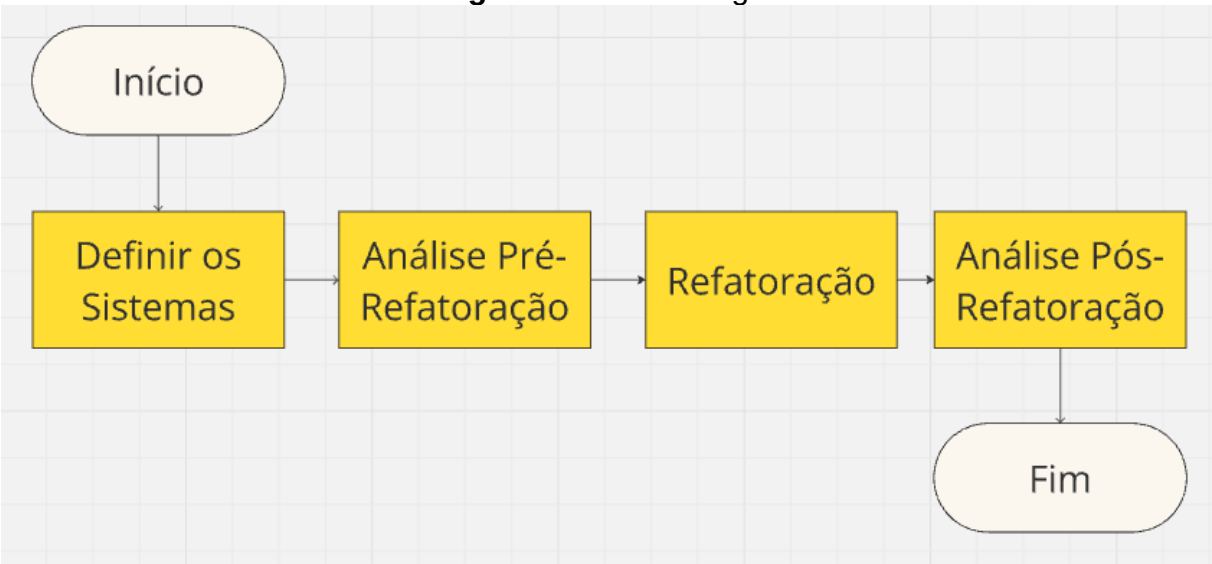
ideal para cada cenário e também a automação da refatoração e medição dos atributos de qualidade, a fim de proporcionar processos mais robustos e eficazes promovendo a prática.

Sobre os estudos relatados, o levantamento destes simbolizam uma relevância do tema à área acadêmica, envolvendo discussões ativas até o presente momento, resultando em um avanço do conhecimento técnico e empregado no mercado de trabalho. Além disso, os mesmos apresentam possíveis técnicas a serem aplicadas em várias etapas de nossa análise, tais como na análise da manutenibilidade em ambos os momentos e também na aplicação da refatoração.

6 METODOLOGIA

Como este trabalho visa realizar uma análise sobre o impacto da refatoração na manutenibilidade, a metodologia consistirá na análise anteriormente e posteriormente a aplicação da refatoração conforme descrito na figura 5.

Figura 5 – Metodologia



Fonte: Os Autores

Nas seções abaixo, é descrito a importância e impacto de cada passo definido na metodologia para que a mesma seja replicada em futuras pesquisas.

6.1 DEFINIR OS SISTEMAS

Neste passo é realizado a definição dos sistemas a serem analisados, que por sua vez apresenta um impacto significativo na pesquisa pois também será a definição do escopo de estudo. Outras pesquisas as quais venham a aplicar semelhante metodologia, poderá apresentar resultados variados de acordo com este passo. Alguns dos fatores que podem intensificar a variância dos resultados são: tecnologia a ser empregada, nicho ou área a qual o sistema é utilizado e o grau de complexidade anterior à refatoração.

6.2 ANÁLISE PRÉ-REFATORAÇÃO

Nesta etapa será coletado e analisado as métricas dos sistemas definidos no passo anterior. Também se faz necessário a definição das métricas a serem utilizadas, variando de acordo com a perspectiva que se propõe ser investigada.

6.3 REFATORAÇÃO

As técnicas de refatoração também podem variar, cada uma normalmente é desenvolvida visando resolver um problema (ou "code smell") e por isso cada técnica que for utilizada, impactará diretamente a pesquisa realizada assim como seu resultado. Algumas técnicas são limitadas para sistemas Orientado a Objetos, enquanto outras são mais eficazes ao paradigma funcional.

Neste estudo foi aplicado a técnica "Extract Method", o qual consiste na extração de linhas de código já presentes e a criação de um novo método utilizando as mesmas, sem que o comportamento do método original seja alterado. Isso proporciona alguns benefícios, tais como a redução do tamanho do método original que por vezes apresenta muitas linhas de código e muitas ramificações do fluxo de código (podendo implicar em uma complexidade maior), e quando aplicado seguindo determinada padronização, desenvolve uma camada de abstração ao longo do sistema o qual facilita na manutenção do mesmo devido sua legibilidade.

No entanto, em casos os quais o trecho extraído consiste em um baixa

ramificação de fluxos ou uma sequência linear, a aplicação do mesmo pode não apresentar grande impacto, sendo este um fator a ser considerado e analisado ao aplicar a técnica.

6.4 ANÁLISE PÓS-REFATORAÇÃO

As informações obtidas nessa etapa consistem nos resultados e consequência da pesquisa aplicada, que por sua vez podem apresentar insights valiosos com relação aos sistemas analisados e as técnicas aplicadas. Nesta etapa os resultados serão discutidos e observações serão realizadas.

7 EXPERIMENTAÇÃO

Nesta seção será relatado a aplicação da metodologia e dos resultados obtidos através da mesma neste trabalho, contendo também por sua vez a análise dos dados obtidos.

7.1 DEFINIR OS SISTEMAS

Os sistemas a serem analisados consistem em 3 módulos de *e-commerce* da plataforma Magento versão 1, por 2 motivos: familiaridade do autor com sistema dessa natureza; os sistemas têm como propósito apresentar uma natureza modular, ou seja, serem aptos a serem instalados e removidos sem dificuldade de sistemas de *e-commerce*, sendo a manutenibilidade um fator relacionado.

Para cada módulo presente, foi analisado com prioridade os componentes relacionados à manipulação e busca de dados, pois tais funções comumente apresentam complexidade por estarem relacionadas com regras de negócio e manipulação de dados. Também foram priorizadas, mas não restringidas, funções que apresentam *a priori* uma quantidade de linhas maior do que outras funções, assim como estruturas de controle e de repetição, o que sinalizam maior complexidade.

7.2 ANÁLISE PRÉ-REFATORAÇÃO

7.2.1 Acoplamento entre objetos

Durante o desenvolvimento deste trabalho foi realizada a identificação de alguns problemas com a aplicação da métrica de CBO.

Com relação à métrica em si, ela tem como propósito medir o quão dependente determinado objeto está com relação à outro. No paradigma Orientado à Objetos, a dependência é algo comum e natural pois as responsabilidades do sistema são distribuídas entre os objetos de forma que a comunicação entre eles se torna necessário para o funcionamento do sistema e a arquitetura do software. No entanto, o fator a ser medido na CBO se trata de dependências diretas, quando não há uma camada de abstração entre uma classe e outra, o que ocasionaria em erros e eventuais problemas para o caso do objeto requisitado ter seus métodos alterados, ou mesmo para futuras extensões e desenvolvimento de novas funcionalidades.

A injeção de dependência é uma técnica a qual permite que o desenvolvedor determine uma classe abstrata no construtor da classe requisitante, e ao instanciar o objeto dessa classe, seja passado uma instância válida da classe requisitada ao construtor de forma que a classe requisitante jamais faça acesso direto ou exclusivo a determinado objeto.

No entanto, essa técnica se mostrou pouco eficaz na análise dos módulos presentes neste trabalho, uma vez que os mesmos fazem uso de inúmeros objetos do framework, estes que são armazenados em um diretório denominado "Core", os quais não são alterados nem personalizados diretamente devido uma convenção e boa prática da comunidade deste framework.

A aplicação da injeção de dependência nestes módulos, apresenta sinais de *overengineering*, sendo a mesma válida para casos os quais o sistema por completo foi desenvolvido pela equipe envolvida no projeto, e não um sistema que faz uso de um produto de terceiros. Por estes motivos a métrica não pôde ser utilizada e este caso permanecerá como um aprendizado de que cada projeto deve ser analisado a fim de compreender quais informações são viáveis e válidas de serem analisadas.

7.2.2 Complexidade ciclomática

A complexidade ciclomática é uma medida de análise de complexidade que analisa os fluxos possíveis da execução de determinado código, por isso é levado em consideração estruturas de decisão, estruturas de controle e fluxo de dados.

A métrica é citada em estudos científicos, desde pesquisas atreladas à metrificação de sistemas Orientado a Objetos (Dallal; Abdin, 2017), desenvolvimento de algoritmos metaheurísticos para resolução de problemas (Osaba *et al.,* 2021) e até mesmo detecção de vulnerabilidade utilizando redes de aprendizado profundo (Lin *et al*., 2020).

Sobre os módulos utilizados neste estudo, foram selecionados *a priori* por familiaridade e identificados como C, OR e CJ (siglas para identificação, utilizando como base seus nomes originais). Analisando a tabela 1, é perceptível que os dados coletados apresentam certa semelhança entre o módulo C e o módulo OR, diferenciando apenas o módulo CJ o qual representa um indício de módulo com código de baixa manutenibilidade e alta complexidade.

Tabela 1 – Complexidade Ciclomática para cada módulo

Módulo	CJ	C	OR	Total
Quantidade de Métodos Analisados	5	4	5	14
Complexidade Média	5	1.2	1.4	2.5
Maior Valor de Complexidade Identificado	8	2	2	8
Complexidade Contabilizada	25	6	7	38

Fonte: Os Autores

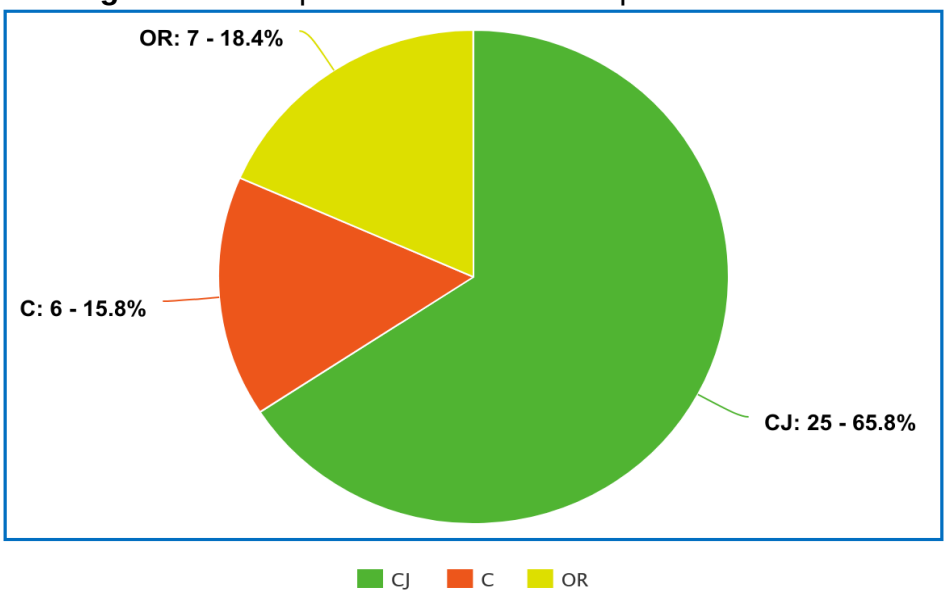
Em geral os valores coletados não significam altas taxas de complexidade, isso porque muitas funções se tratam de procedimentos breves e rápidos, não envolvendo muitas validações e variações no fluxo de dados.

Outro ponto a se levantar é a reusabilidade de funções que promovam um código normalizado, evitando duplicidade e mantendo um certo nível de abstração o qual colabora com a legibilidade e complexidade do sistema. Tal comportamento se fez presente em grande parte dos componentes analisados.

A partir da figura 6 podemos obter algumas informações interessantes como por exemplo o impacto da complexidade contabilizada no módulo CJ com relação aos

outros módulos. O módulo CJ representa 65% da complexidade contabilizada nos métodos analisados, um valor muito significativo o qual demonstra alta complexidade do módulo em comparação com os demais.

Figura 6 – Complexidade Ciclomática para cada módulo



Fonte: Os Autores

Evidenciando a diferença do CJ para os outros módulos, a tabela 1 apresenta a média de complexidade entre as funções de cada módulo e destaca a diferença entre o módulo CJ e os demais.

Investigando minuciosamente os métodos de maior complexidade de cada módulo, podemos perceber que o módulo CJ possui uma função em específico o qual sua complexidade é 8, sendo essa a maior complexidade contabilizada.

Para contabilizar a complexidade ciclomática dos sistemas foi realizada a diagramação dos sistemas nos momentos anteriores à refatoração, com base nos estudos de McCabe.

Os diagramas que apresentam formato de grafo, auxiliaram na visualização das informações e coleta dos dados, pois a partir dos mesmos foi mais eficaz e assertivo a contabilização das variáveis utilizadas na fórmula de McCabe conforme demonstrado nos grafos em anexo.

7.3 REFATORAÇÃO

Com o método de refatoração definido, o mesmo foi aplicado nos métodos de maior complexidade de cada módulo (método de complexidade 8 para módulo CJ, e complexidade 2 para os módulos C e OR).

Além disso, como o módulo CJ apresentava uma média de complexidade por método maior que os outros, a técnica foi replicada em outros 2 métodos do mesmo módulo, sendo essas de complexidade 6 e 5, pois acredita-se que dessa forma podemos ter um aproveitamento melhor das técnicas de refatoração.

7.4 ANÁLISE PÓS-REFATORAÇÃO

Com relação às métricas obtidas, importante apresentar como as mesmas serão reportadas. Se tratando das novas funções as quais são consequência da aplicação de "Extract Method", considerar uma das seguintes formatações:

- Quantidade de Funções (valor de complexidade de cada uma) Exemplo:
  - 4(1): quatro funções cada uma com complexidade de valor um.
  - 2(1,2): duas funções cada uma respectivamente com valor um e dois de complexidade
- Quantidade de Funções (quantidade de funções agrupadas:valor de complexidade a qual cada uma das funções agrupadas possui), exemplo:
  - 4(2:1,2:3): Quatro funções com duas apresentando complexidade valor um e outras duas apresentando complexidade valor três.

A tabela 2 apresenta uma relação entre cada função metrificada e refatorada, assim como as novas funções geradas a partir desta. A tabela em questão sinaliza um cenário positivo para a aplicação da técnica de extração de método para redução de complexidade, conforme visualizado nos métodos C e CJs. Interessante perceber que o método de maior complexidade, CJ(3), não obteve uma redução grande da complexidade em si, mesmo utilizando de uma quantidade expressiva de novas funções como resultado da extração de método, podendo este ser um sinal de uma possível ineficiente da utilização do método de forma isolada.

Tabela 2 – Comparativo antes e depois da refatoração

Funções	Complexidade Antes	Complexidade Depois	Novas Funções
OR	2	2	4(1)
C	2	1	2(1,2)
CJ (1)	6	2	3(2,1,3)
CJ (2)	5	3	2(1,2)
CJ (3)	8	7	4(2:1,2:3)

Fonte: Os Autores

Outra situação que simboliza ineficiência do método em questão é a função OR o qual sua complexidade se manteve e quatro novas funções foram geradas, sendo um cenário o qual cada trecho extraído tem a mesma complexidade que a chamada da nova função. Isso representa uma situação a qual não é necessário ou mesmo eficaz a aplicação da extração de método, inclusive porque a complexidade da função antes da refatoração já era um valor muito baixo, logo não haviam sinais de necessidade da refatoração ou mesmo possibilidade de uma melhora significativa.

8 CONCLUSÃO

Com base nos pontos discutidos na análise pós-refatoração, podemos concluir que a técnica empregada apresenta sinais de impacto positivo na complexidade dos sistemas, no entanto a mesma deve ser utilizada com cautela pois existem sinais que não são em todos os cenários que apresentará eficácia.

Existem determinados fatores a serem considerados e que por sua vez podem ser ameaças à validade do trabalho em questão. Devemos destacar que a refatoração foi empregada através de uma única técnica, sendo que a prática de refatorar envolve a aplicação de uma variância de técnicas. A técnica aplicada neste e em qualquer outra pesquisa pode implicar em uma variância dos resultados obtidos.

Por fim, a complexidade ciclomática não leva em consideração algumas melhorias de legibilidade obtidas através da técnica de refatoração empregada, por isso é importante observar que o resultado obtido com a métrica mencionada, se trata da observação da manutenibilidade sobre uma perspectiva específica, e futuros

trabalhos podem explorar outras perspectivas. Além disso, estudos e aplicações da mesma, consideram valores abaixo de 10 como "Satisfatórios", o que pode significar que não houve método passível de utilizar a técnica aplicada em total eficácia. A utilização de sistemas de maior complexidade podem resultar em variações significativas.

Considerando as ameaças de validade, futuros trabalhos podem envolver sistemas de complexidade maior na aplicação da metodologia presente neste estudo. Outro trabalho futuro seria a análise de outros atributos de qualidade interna do código após a aplicação da técnica de refatoração citada anteriormente, visando analisar se a técnica de refatoração aplicada apresentou uma melhoria significativa em algum outro aspecto em específico.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **NBR ISO 9000: Sistemas de gestão da qualidade - Fundamentos e vocabulário**. Rio de Janeiro, Brasil: [s.n.], 2015. Norma Técnica. NBR ISO 9000:2015.

BECK, K. **Test Driven Development: By Example**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321146530.

CEDRIM, D. et al. Does refactoring improve software structural quality? A longitudinal study of 25 projects. *In*: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 30., 2016, New York, NY, USA. **Proceedings** [...]. New York, NY, USA: Association for Computing Machinery, 2016. p. 73–82. ISBN 9781450342018. Disponível em: https://doi.org/10.1145/2973839.2973848.

CHÁVEZ, A. et al. How does refactoring affect internal quality attributes? A multi-project study. *In*: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 31., 2017, New York, NY, USA. **Proceedings** [...]. New York, NY, USA: Association for Computing Machinery, 2017. p. 74–83. ISBN 9781450353267. Disponível em: https://doi.org/10.1145/3131151.3131171.

DALLAL, J. A.; ABDIN, A. Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review. **IEEE Transactions on Software Engineering**, p. 1–1, jan. 2017.

DUC, A. N. The impact of software complexity on cost and quality - A comparative analysis between open source and proprietary software. **International Journal of Software Engineering Applications**, v. 8, p. 17–31, mar. 2017.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. [S.l.]: Novatec Editora, 1999.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). **ISO/IEC 9126-1:2001(E) Software engineering – Product quality – Part 1**: Quality model. Geneva, Switzerland, 2001.

JAMIL, A. *et al*. Software testing techniques: A literature review. *In*: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY FOR THE MUSLIM WORLD, 6., 2016, Indonésia. **Proceedings** [...]. Indonésia: IEEE, 2016. p. 177–182.

LACERDA, G. *et al.* Code smells and refactoring: A tertiary systematic review of challenges and observations. **The Journal of Systems and Software, Elsevier,** v. 167, p. 110610, 2020. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0164121220300732.

LIN, G. *et al*. Software vulnerability detection using deep neural networks: A survey. **Proceedings of the IEEE**, v. 108, n. 10, p. 1825–1848, 2020.

MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. 1. ed. USA: Prentice Hall PTR, 2008. ISBN 0132350882.

MCCABE, T. A complexity measure. **IEEE Transactions on Software Engineering**, SE-2, n. 4, p. 308–320, 1976.

OSABA, E. *et al.* A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. **Swarm and Evolutionary Computation**, v. 64, p. 100888, 2021. ISSN 2210-6502. Disponível em: https://www.sciencedirect.com/science/article/pii/S2210650221000493.

PAN, J. Software testing. **Carnegie Mellon University** 18-849b Dependable Embedded Systems, 1999. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=28abbfdcd695f6ffc18c5041f8208dcfc8810aaf.

SILVA, D.; TSANTALIS, N.; VALENTE, M. T. Why we refactor? Confessions of GitHub contributors. *In*: SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 24., 2016. **Proceedings** [...]. [*S.l.*]: ACM, 2016. p. 858-870. Disponível em: http://dx.doi.org/10.1145/2950290.2950305.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo, SP: Pearson, 2018. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 29 mar. 2024.