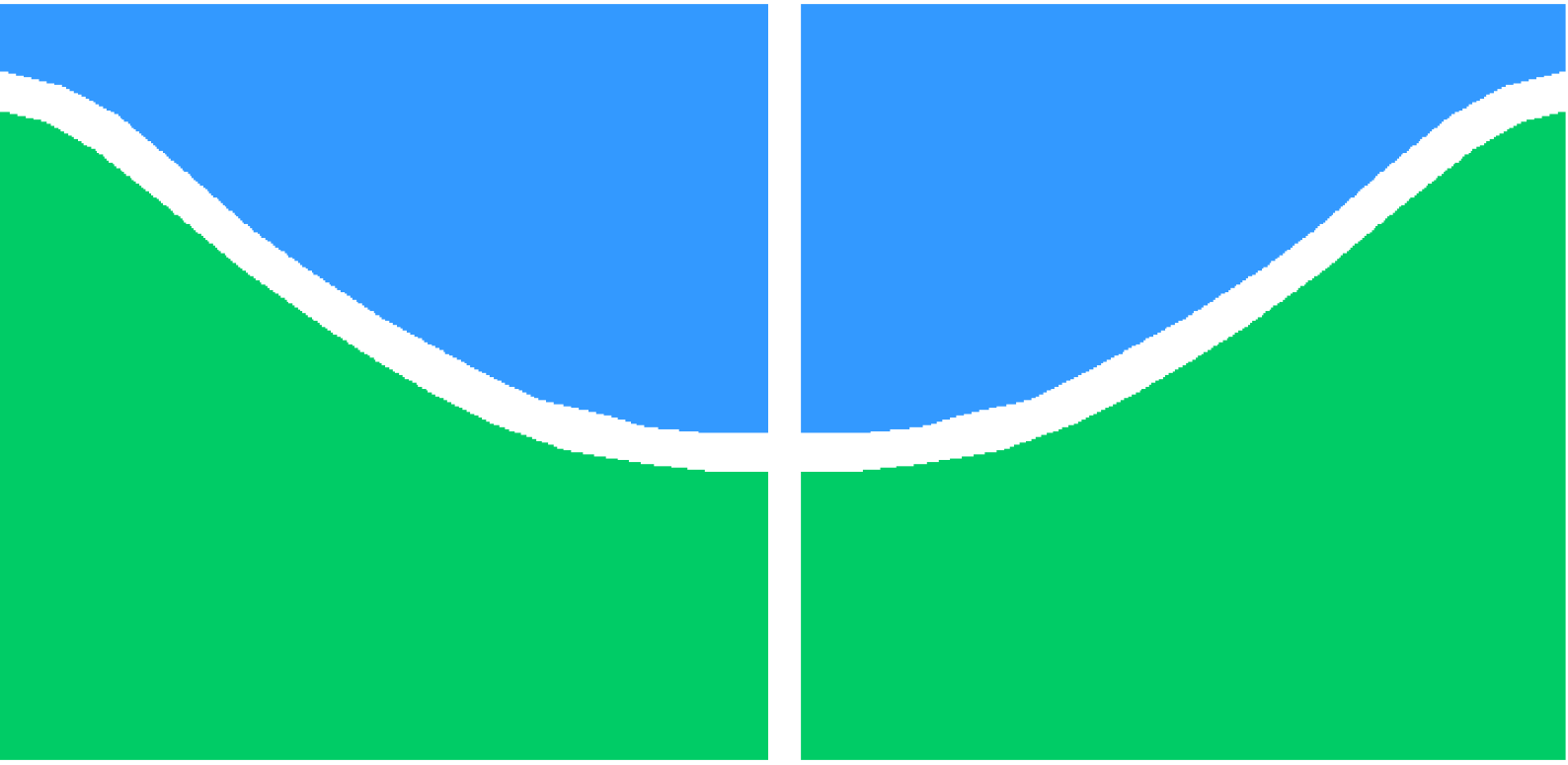

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Framework para Avaliação da Qualidade de Código: Uma Abordagem Baseada em Valor

Autor: Matheus Herlan dos Santos Ferraz
Orientador: Profª Msc. Elaine Venson

Brasília, DF
2016

Matheus Herlan dos Santos Ferraz

Framework para Avaliação da Qualidade de Código: Uma Abordagem Baseada em Valor

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof[a] Msc. Elaine Venson

Brasília, DF

2016

Matheus Herlan dos Santos Ferraz
Framework para Avaliação da Qualidade de Código: Uma Abordagem Baseada em Valor/ Matheus Herlan dos Santos Ferraz. – Brasília, DF, 2016-
60 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª Msc. Elaine Venson

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Qualidade de Código. 2. Engenharia de Software Baseada em Valor. I. Profª Msc. Elaine Venson. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Framework para Avaliação da Qualidade de Código: Uma Abordagem Baseada em Valor

CDU 02:141:005.6

Matheus Herlan dos Santos Ferraz

Framework para Avaliação da Qualidade de Código: Uma Abordagem Baseada em Valor

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 28 de novembro de 2016:

Profª Msc. Elaine Venson
Orientador

Profª Msc. Cristiane Soares Ramos
Convidado 1

Prof. Dr. Sérgio Antônio Andrade de Freitas
Convidado 2

Brasília, DF
2016

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Primeiramente, gostaria de agradecer à Deus por tudo que Ele me possibilitou viver, chegando até este momento. Em segundo lugar, gostaria de agradecer aos meus pais, Célio Ferraz e Aurenice Ferraz, e irmão, Daniel Ferraz, que sempre me apoioram, não somente durante minha jornada de graduação, mas também, durante toda a vida.

Agradeço à professora Elaine, que com paciência e sabedoria, se colocou à disposição para me auxiliar a concluir este trabalho. Por fim, gostaria de agradecer à todos os professores que fizeram parte da minha graduação, pois, sem dúvida alguma, todos contribuíram para o aperfeiçoamento da minha visão intelectual e profissional.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

A produção de código e a implementação de testes unitários, bem como a realização de inspeções de código, estão instrinsecamente conectados. Testes unitários e inspeções, enquanto práticas complementares da verificação de *software*, foram concebidas com a intenção de aprimorar a identificação de defeitos existentes no código fonte do *software*. Nesse sentido, também é válido notar que a qualidade do código fonte influi na qualidade de uso do *software*, sendo esta contemplada pelo usuário. Neste trabalho, o objetivo é propor um *framework* que reúne um conjunto de atividades e práticas que favoreçam a implementação de testes unitários e realização de inspeções de código, levando em consideração os preceitos da Engenharia de Software Baseada em Valor, que aborda o alinhamento entre a missão do projeto e as atividades técnicas de desenvolvimento de *software*. Espera-se, como resultados, que o *framework* esteja adequado ao uso em qualquer organização ao final da aplicação dos procedimentos técnicos de pesquisa.

Palavras-chave: Qualidade de Código; Testes Unitários; Inspeções de Código; Engenharia de Software Baseada em Valor.

Abstract

The production of code and the implementation of unit tests, as well as the code inspections, are closely connected. Unit tests and inspections, as complementary software verification practices, were designed with the intention of improving the identification of defects in the software source code. In this sense, it is also worth noting that the quality of the source code influences the quality of use of the software, which is contemplated by the user. In this work, the objective is to propose a framework that combines a set of activities and practices that favor the implementation of unit tests and code inspections, taking into account the precepts of Value-Based Software Engineering, which addresses the alignment between the project's mission and the technical activities of software development. As results, it is expected that the framework is suitable for use in any organization at the end of the application of technical research procedures.

Key-words: Code Quality; Unit Tests; Code Inspections; Value-Based Software Engineering.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CMMI Capability Maturity Model Integration

IEEE Institute of Electrical and Eletronics Engineers

NASA National Aeronautics and Space Administration

PMBOK Project Management Body of Knowledge

VBSE Value-Based Software Engineering

Sumário

1 Introdução

Este trabalho caracteriza-se como uma monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, Campus Gama. O trabalho está organizado de forma a prover um bom entendimento ao leitor, apresentando inicialmente uma contextualização sobre o assunto tratado, os objetivos do trabalho, bem como um detalhamento acerca da problemática identificada.

1.1 Contextualização

Nas últimas décadas, o controle de qualidade e uso de padrões atraíram muita atenção. Contudo, historicamente, este assunto é muito antigo (KOSCIANSKI; SOARES, 2007). A exemplo dessa afirmação, tem-se os diversos padrões de medida e controle de produção que foram estabelecidos há mais de quatro mil anos pelas antigas civilizações, como os egípcios.

A partir da observação da linha cronológica de evolução da qualidade, um grande marco, sem dúvida alguma, foi a revolução industrial. Nesse período, a expansão industrial fomentou uma situação de concorrência entre as diversas empresas que surgiram e assim, o processo de melhoria contínua passou a ser buscado. A indústria de desenvolvimento de *software*, embora tenha sido criada em um período mais recente, também está inserida nesse contexto.

As empresas que atuam no desenvolvimento de *software* possuem os desafios de lidar com a crescente taxa de mudanças tecnológicas e adicionalmente, com o aumento dos níveis de concorrência em escala global. Mediante esta perpectiva, a fim de permanecer, as empresas buscam constantemente novas estratégias para se diferenciarem de seus concorrentes.

Nesse âmbito, deve-se considerar também a crescente complexidade dos produtos de *software*. Segundo Sommerville (2003), é válido notar que *software* está presente em várias atividades do cotidiano das pessoas e isso favorece ainda mais a demanda por este produto, acarretando na construção de produtos até mesmo inusitados dependendo da necessidade. Por outro lado, para Dijkstra (1972), o aumento de complexidade deve-se também à rápida melhoria e aperfeiçoamento que se obteve na criação de máquinas.

Como resultado da combinação entre os fatores citados no parágrafo anterior e a crescente pressão imposta pelo mercado, as empresas se voltam para investigação das abordagens de validação e das técnicas de verificação para garantir o desenvolvimento de produtos de valor agregado e de alta qualidade (BIFFL et al., 2014).

Embora na contemporaneidade já se tenha metodologias e práticas para a construção de *software* de forma mais rigorosa e controlada, ainda é possível perceber problemas mencionados na década de 70, sendo eles (KOSCIANSKI; SOARES, 2007):

- Cronogramas não observados.
- Projetos com tantas dificuldades que são abandonados.
- Módulos que não operam corretamente quando combinados.
- Programas que não fazem exatamente o que era esperado.
- Programas tão difíceis de usar que são descartados.
- Programas que simplesmente param de funcionar.

Dentre as metodologias concebidas para resolver os problemas enfrentados no desenvolvimento de *software*, tem-se abordagens mais tradicionais (processo unificado por exemplo) e abordagens mais adaptativas (métodos ágeis), sendo estas mais recentes e amplamente adotadas por diversas organizações na atualidade.

Intrínseco à metodologia de desenvolvimento, tem-se as práticas de verificação de *software*. Estas, quando corretamente aplicadas, resolvem boa parte dos problemas identificados no desenvolvimento de *software* citados anteriormente.

Para as empresas fazerem uso conjunto das práticas de verificação e validação de software é necessário que haja uma estratégia de desenvolvimento que favoreça a aplicação de tais práticas. Contudo, durante a concepção desta metodologia de desenvolvimento, muitas organizações não conseguem abstrair as práticas mais básicas e extremamente necessárias para a construção do produto e acabam por negligenciar determinadas atividades. A exemplo dessa afirmação, tem-se a negligência que se pode contemplar em atividades de implementação e execução de testes e também, nas inspeções de código (KANER, 1995). Para atender aos prazos, muitas equipes de projeto decidem protelar essas atividades, tornando a qualidade do produto inferior ao que se poderia obter. Adicionalmente, em muitos casos, as atividades técnicas como as citadas anteriormente não estão alinhadas plenamente aos interesses dos clientes, favorecendo a entrega de um produto de menor valor agregado (RAMLER; BIFFL; GRüNBACHER, 2014).

Adotar uma metodologia de desenvolvimento perfeitamente adequada para as necessidades da organização e que também atenda de forma concisa aos interesses de todos os envolvidos no projeto não é uma tarefa fácil. Porém, a perspectiva da Engenharia de *Software* Baseada em Valor fornece uma boa maneira de olhar para o processo de desenvolvimento do produto. É válido ressaltar que os envolvidos em um projeto de desenvolvimento de *software* (clientes, analistas de negócio, gerentes de projetos, arquitetos

de *software*, desenvolvedores etc) devem possuir um melhor entendimento das implicações provenientes das decisões efetuadas sobre o produto (BIFFL et al., 2014).

1.2 Problema

Na indústria de desenvolvimento de *software*, em geral, há baixo nível de aplicação das principais práticas propostas pela Verificação de *Software* para a construção de produtos de maior qualidade. Na grande maioria dos projetos da indústria de desenvolvimento, essas práticas são negligenciadas, caracterizando-se como atividades de menor importância (KANER, 1995).

Adicionalmente, outro quesito que deve ser ressaltado é que as concepções dos clientes de negócio não são concisamente levadas em conta no momento da execução de atividades mais técnicas como as citadas anteriormente. Para exemplificar esta afirmação, basta analisar o que ocorre durante a elaboração e execução de testes para o sistema que está sendo construído. Testes, muitas vezes não estão organizados para maximizar o valor de negócio e também, não estão alinhados com a missão do projeto (RAMLER; BIFFL; GRüNBACHER, 2014).

Na busca de possíveis soluções para a problemática destacada, do ponto de vista técnico, já existem práticas propostas pela Verificação de *Software*, tais como a elaboração de testes, inspeção de código etc. Nesse sentido, seria necessário reunir esse conjunto de boas práticas em um único guia. Por outro lado, para lidar com a conciliação dos interesses de todos os envolvidos em um projeto de construção de *software*, tem-se as abordagens da Engenharia de *Software* Baseada em Valor (VBSE - *Value-Based Software Engineer*).

A VBSE traz considerações de valor para o primeiro plano, de modo que as decisões em todos os níveis possam ser otimizadas, para atender ou conciliar os objetivos explícitos das partes interessadas, do marketing pessoal e analistas de negócio aos desenvolvedores, arquitetos e especialistas em qualidade (BIFFL et al., 2014).

Dessa forma, este trabalho apresenta a seguinte questão de pesquisa:

- Como utilizar as inspeções e testes unitários, que são práticas complementares da Verificação de *Software*, em conjunto com os conceitos oriundos da VBSE, na avaliação da qualidade de código no contexto do desenvolvimento ágil?

Outro quesito que deve ser evidenciado é que devido às limitações de orçamento e prazos, é inviável inspecionar todo o código do *software* ou em alguns casos, obter um percentual de cobertura de testes de 100% (cem por cento). Nesse contexto, a VBSE se torna ainda mais importante, pois minimamente, as partes mais críticas e que mais agregam valor para o cliente devem ter a qualidade assegurada.

1.3 Objetivo

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é propor um conjunto de atividades, práticas e ferramentas que favoreçam as atividades de inspeção e elaboração de testes unitários, levando em consideração conceitos da VBSE. Por meio deste conjunto, busca-se alcançar bons níveis de manutenibilidade de código no contexto de desenvolvimento ágil.

Este trabalho, como mencionado, possui foco em metodologias ágeis, mais especificamente no *Scrum* e nas práticas mais básicas e complementares da verificação de *software*, sendo elas a implementação de testes unitários e inspeções de código. O *framework* aqui elaborado, reúne essas práticas, a partir de uma ótica de análise de efetividade das mesmas, com atividades do *Scrum*. Naturalmente, exercendo um controle maior sobre a escrita do código que concretiza o *software*, alinhando esta atividade ao propósito do projeto, será possível obter produtos de maior qualidade.

1.3.2 Objetivos Específicos

Por objetivos específicos para este trabalho, tem-se:

- Consolidar o entendimento acerca dos elementos necessários à construção do *framework* de avaliação da qualidade de código.
- Pesquisar abordagens inerentes às inspeções e aos testes unitários para a elaboração do *framework*.
- Definir detalhes acerca da execução das atividades propostas pelo *framework* junto às atividades presentes no *Scrum*.
- Incorporar diretrizes propostas pela Engenharia de *Software* Baseada em Valor ao *framework*.

1.4 Organização do Documento

- **Capítulo 2 - Referencial Teórico**: apresenta conceitos e abordagens relacionados ao tema deste trabalho, explicitando as informações obtidas a partir da pesquisa bibliográfica e também, a partir da realização da revisão sistemática.
- **Capítulo 3 - Metodologia**: especifica a metodologia adotada para a pesquisa e desenvolvimento deste trabalho, elencando também como a efetividade do *framework* será avaliada e também, quais instituições serão utilizadas como caso durante a aplicação do *framework*.

- **Capítulo 4 - *Framework* de Avaliação de Código**: apresenta os *checklists* elaborados, correlacionando estes ao detalhamento do *framework* proposto.

2 Referencial Teórico

Primeiramente, é válido ressaltar que o *framework* proposto neste trabalho está voltado para o contexto de desenvolvimento que adota metodologias ágeis. Adicionalmente, as atividades constituintes do *framework* visam o alinhamento com os aspectos centrais e complementares propostos pela Verificação de *Software*, que são as inspeções de código e implementação de testes, mais especificamente testes unitários para o âmbito do *framework* concebido neste trabalho.

Neste capítulo, busca-se apresentar conceitos que fundamentam a concepção do *framework*. Na seção 2.1, é possível contemplar uma breve explanação com relação à Verificação de *Software*. Já na seção 2.2, serão explanados aspectos principais inerentes às inspeções, bem como a base para o *checklist* de inspeção de código elaborado para o *framework*. Ao longo da seção 2.3, serão apresentadas definições elucidativas para a atividade de teste a partir de uma perspectiva genérica, aprofundando paulatinamente a análise para aspectos pontuais dos testes unitários, exibindo-se os resultados da revisão sistemática para a elaboração do *checklist* de implementação de testes unitários contido no *framework*. Por fim, na seção 2.4, haverá uma apresentação mais detalhada acerca da perspectiva da VBSE.

Os conceitos aqui apresentados são importantes pelo fato de demonstrarem plenamente o propósito das atividades existentes no *framework*.

2.1 Verificação de *Software*

A verificação é uma das principais disciplinas da Engenharia de *Software*. A verificação, segundo definição do IEEE (*Institute of Electrical and Eletronics Engineers*), caracteriza-se como o processo de avaliar um sistema, produto ou componente para determinar se os resultados de um passo do respectivo processo de desenvolvimento satisfazem as condições impostas no início do passo. Pelo CMMI (*Capability Maturity Model Integration*), a verificação é tida como uma confirmação de que produtos de trabalho refletem corretamente os requisitos especificados.

As verificações incluem análises estáticas, testes de desenvolvimento (testes de unidade e integração) e revisões (FILHO, 2009). Nesse âmbito, uma verificação efetiva aumenta a visibilidade do processo de desenvolvimento e reduz os riscos do projeto (SINGH; BAWA, 1995).

Como mencionado em seções anteriores, o presente trabalho propõe um *framework* que agrega práticas da verificação de *software*, especificamente implementação de testes

unitários e inspeções de código. Assim sendo, como a verificação evidencia a sistemática de desenvolvimento do produto, naturalmente, esta favorece a agregação de valor ao produto, alinhando as atividades técnicas da execução do projeto com a missão do mesmo.

2.2 Inspeção

Segundo a definição do Glossário do IEEE, uma inspeção (*inspection*) caracteriza-se como um exame visual de produtos de trabalho para detectar e identificar anomalias. Adicionalmente, é possível contemplar uma definição ainda mais detalhada que consta na norma IEEE-1028, que acrescenta o fato de que o exame realizado na atividade de inspeção inclui, ao identificar anomalias, erros e desvios em relação a padrões e especificações.

A inspeção é o tipo mais formal de revisão. O objetivo principal é a identificação e remoção de defeitos (FILHO, 2009). Nesse contexto, é válido ressaltar que o IEEE define anomalia como qualquer coisa observada na documentação ou operação de um produto de *software* que se desvie de expectativas baseadas em outros produtos já verificados, ou em materiais de referência. Adicionalmente, defeito, segundo o PMBOK (*Project Management Body of Knowledge*) é uma imperfeição ou deficiência em um componente do projeto na qual esse componente não atende aos seus requisitos ou especificações, fazendo-se necessário o reparo ou substituição.

2.2.1 Itens da Inspeção de Código

É válido ressaltar que inspeções formais de *software* objetivam a detecção e eliminação de erros em produtos desenvolvidos durante o seu ciclo de desenvolvimento. Nesse sentido, inspeções formais são aplicáveis à qualquer parte do produto de *software*, incluindo requisitos, especificação, *design* e código (JúNIOR et al., 2003).

Segundo (JúNIOR et al., 2003), um *checklist* para inspeção formal de código deve contemplar, dentre outras técnicas, os principais itens:

- **Retorno de métodos**: Retorno de métodos e/ou rotinas muitas vezes pode provocar a interrupção do fluxo do programa, descontinuidade, corrupção de pilha, estouro de memória ou valor incorreto. Assim, verificando este quesito, é possível constatar se todas as rotinas ou métodos retornam valores de maneira correta.
- **Tratamento de interrupção e regiões críticas**: Esta verificação atesta a manutenção de interrupções pelas rotinas correspondentes e por rotinas que dependem dos serviços de interrupção em regiões críticas. É válido ressaltar que este aspecto deve ser profundamente investigado principalmente quando se trata do desenvolvimento de sistemas críticos. Nesse sentido, a inspeção deve localizar todas as rotinas de interrupção de serviços e rotinas que são chamadas por serviços de interrupção.

- **Controle de *loops***: Este item verifica os loops para garantir que eles possuem fim (exceto quando intencionalmente nunca terminam), evitando ciclos infinitos no programa.
- **Teste de I/O**: Este quesito verifica o I/O de rotinas importantes, especialmente àquelas onde a reentrada deve ser prevenida. Entrada e saída de dados em um programa é um aspecto que deve ser bem avaliado, justamente pelo fato de possuir implicações diretas no uso da CPU (analisando a perspectiva de escalonamento dos processos).
- **Controle de fluxo do programa**: Este item verifica se a sequência do programa está correta. A incorreta utilização de estruturas de controle pode resultar em uma execução inesperada, possibilitando situações de risco no funcionamento do *software*.
- **Código inutilizado**: Esta verificação atesta se não há nenhum código entre marcas de comentário e, portanto, não utilizado (em geral rotinas que serviram para o desenvolvimento e que não são mais utilizadas). Este fator pode ter influência negativa na manutenção futura do código.
- **Variáveis e constantes**: O uso de variáveis e constantes é outro aspecto que deve ser verificado de forma concisa. Deve-se atentar para a correta atribuição dos valores, bem como sua atualização.
- **Comentários de código**: Deve-se verificar se os comentários de fato melhoram a compreensão do código, melhorando a manutenibilidade. Adicionalmente, é importante destacar que os comentários não devem ser ambíguos, podendo resultar em interpretações incorretas.
- **Legibilidade de código**: A legibilidade é fundamental para a manutenção do código. Quanto menor a complexidade na escrita do código, menor o esforço na compreensão.
- **Diretivas ao pré-processador**: Deve-se evitar a utilização indiscriminada das diretivas ao pré-processador no código fonte. Este aspecto evita erros durante a manutenção.
- **Otimização de código**: Deve-se evitar deterinadas otimizações durante a compilação, o que pode gerar um código objeto de maneira inesperada. Em geral, linguagens de programação de alto nível têm aspectos ambíguos que podendo levar a uma dupla interpretação dependendo do nível de otimização selecionado.

O *framework* proposto neste trabalho centraliza a prática da inspeção no exame do código fonte do produto de software, incluindo o código dos testes unitários implementados.

2.3 Teste de *Software*

Segundo o IEEE, um teste é uma atividade na qual um produto, sistema ou componente é executado sob condições especificadas. A partir dessa execução controlada, há uma observação e registro dos resultados e também, avaliação de um ou mais aspectos.

Mediante essa abordagem, os testes são mais do que apenas um meio de detecção e correção de erros, mas se caracterizam também como indicadores da qualidade do produto. Em geral, quanto maior o número de defeitos detectados em um *software*, infere-se que o número de defeitos não detectados também é grande. É importante ressaltar também que a contemplação de uma quantidade exorbitante de defeitos em testes indica a provável necessidade de redesenho dos itens testados.

Existe uma variedade de tipos de teste nos processos de desenvolvimento de *software*. Contudo, a corrente pesquisa se propõe a avaliar mais prontamente aspectos associados aos testes unitários ou de unidade.

2.3.1 Testes Unitários

De maneira geral, como soluções em *software* são elaboradas a partir de uma necessidade de um cliente real, muitas regras de negócio são implementadas e assim, alguns sistemas tornam-se razoavelmente complexos.

Por outro lado, é importante destacar que devido às boas práticas propostas pela Engenharia de *Software*, as regras de negócio não são implementadas em um único arquivo. Em um sistema orientado a objetos, por exemplo, existem diversas classes, cada uma exercendo um papel específico.

Dessa forma, um teste de unidade não se preocupa com todo o sistema, mas apenas com uma pequena parte do mesmo. Geralmente, em sistemas orientados a objetos, uma unidade do sistema é uma classe. Contudo, levando em consideração outros paradigmas de programação, uma unidade também pode ser um procedimento.

Além dos aspectos citados anteriormente, considerando o conceito de unidade adotado para a implementação dos testes unitários, faz-se necessária a construção de códigos auxiliares (*Test Harness*) (BIASI, 2006). O código auxiliar é constituído de *drivers* e *stubs* de teste.

Um *driver*, basicamente, é uma unidade que implementa chamadas às funcionalidades testadas. Os *stubs*, por sua vez, são utilizados para substituir funcionalidades que ainda não foram implementadas ou que estão subordinadas ao módulo que está sendo testado.

A elaboração de *drivers* e *stubs* é importante pelo fato de que um determinado método de teste deve, de fato, testar uma unidade de maneira isolada. São mecanismos

que auxiliam no tratamento do código como uma composição de várias unidades.

É válido ressaltar que testes unitários estão inseridos no âmbito do primeiro nível da estratégia de teste de *software* (GANESAN et al., 2013). A veracidade desta afirmação é comprovada pelo fato de que são os primeiros testes elaborados para um *software* em construção, utilizando o conhecimento que se tem do código fonte.

Como mencionado em seções anteriores, a corrente pesquisa foca a vertente dos testes em nível unitário. Sabe-se que os testes unitários, como qualquer outro teste e com suas particularidades, auxiliam na detecção de defeitos e indicam qualidade do *software*. Contudo, também é necessário avaliar se os testes unitários são efetivos e se são portadores de qualidade, ou seja, se foram bem elaborados.

Assim, o *framework* proposto, além de avaliar a qualidade do código de uma maneira geral, avalia também a qualidade do código inerente aos testes unitários.

2.3.2 Revisão Sistemática - Qualidade dos Testes Unitários

A partir da leitura dos artigos selecionados durante a realização da revisão sistemática, foi possível contemplar abordagens quanto:

- Ao pensamento que os desenvolvedores devem ter quando se discute implementação de testes unitários.
- À existência de práticas e técnicas que devem estar presentes no processo de desenvolvimento de *software* para que os testes unitários sejam efetivos e de qualidade.
- Às ferramentas que apoiam a construção de testes unitários.

2.3.3 Abordagens de pensamento para elaboração de testes unitários

Com relação às abordagens de pensamento para elaboração de testes unitários, (GANESAN et al., 2013) e (ANICHE; OLIVA; GEROSA, 2013) trazem duas abordagens fundamentais para a ideologia de desenvolvedores:

- Testes unitários são parte de um *software*, sendo também entregáveis.
- A quantidade de assertivas utilizadas em código de teste unitário favorecem a percepção de aspectos intrínsecos à qualidade de código.

Primeiramente, deve-se notar que a cultura existente no desenvolvimento de *software* quanto à elaboração de testes unitários deve mudar. Esta é uma atividade fortemente

negligenciada na indústria de desenvolvimento. Assim, instituições de prestígio na comunidade de desenvolvimento científico e tecnológico tem apresentado perspectivas importantes a serem consideradas com relação à atividade de implementação de testes unitários. A exemplo disso, tem-se a NASA (*National Aeronautics and Space Administration*), que por desenvolver sistemas críticos, concebeu um pensamento extremamente diferenciado e que atribui a devida importância às atividades de verificação, mais especificamente, a implementação de testes.

É importante destacar que testes unitários são parte integral de um produto e assim, as diferentes versões dos testes unitários também devem ser controladas e gerenciadas como qualquer outra parte do código fonte do produto (GANESAN et al., 2013). Nesse sentido, artefatos de teste também são entregáveis. Portanto, os testes unitários bem como os *stubs* e os resultados da execução destes podem ser considerados entregáveis para o cliente como uma maneira de relatar a qualidade presente na construção do produto.

Adicionalmente, as assertivas utilizadas em testes unitários emitem alertas aos desenvolvedores com relação à qualidade do código como um todo, sendo evidenciados aspectos da complexidade ciclomática, número exarcebado de linhas de código em um módulo e invocações de métodos (ANICHE; OLIVA; GEROSA, 2013).

Considerando as colocações feitas anteriormente, tem-se um alinhamento entre atividades técnicas e a missão de um projeto, o que é fortemente ministrado pela VBSE. Para se obter êxito na construção de um *software*, a prática de implementação de testes unitários deve ser considerada relevante e extremamente significativa.

2.3.4 Práticas e técnicas para elaboração de testes unitários

Além das abordagens voltadas para o pensamento dos desenvolvedores citadas anteriormente, é importante evidenciar práticas e técnicas que podem e devem ser empregadas para elaboração de testes unitários mais concisos. Segundo (GANESAN et al., 2013), são elas:

- Deve-se criar muitos métodos de teste pequenos ao invés de se criar poucos métodos de teste grandes.
- Código de teste deve possuir convenções de nomenclatura.
- A ordem de execução dos testes não deve ser fator decisivo.
- Testes devem ser auto verificáveis.
- A estrutura hierárquica dos testes unitários facilita a compreensão destes.

- Deve-se criar estratégias para testar funções mais internas.
- *Stubs* devem ser simples e pequenos.
- *Stubs* não devem depender de outros *Stubs* que simulam comportamentos de outros módulos.
- A arquitetura do *software* deve comportar *stubs*.
- A arquitetura do *software* deve abstrair aspectos pertinentes ao *hardware* e ao sistema operacional.
- Utilização de ferramentas de análise de cobertura auxiliam a desenvolver novos cenários de teste.
- Gráficos e métricas são úteis para analisar a qualidade de testes.

É válido ressaltar que as práticas e técnicas apresentadas anteriormente foram derivadas a partir do sucesso contemplado na atividade de implementação de testes unitários promovida pela equipe da NASA.

Outro aspecto do ponto de vista técnico que deve ser verificado em testes unitários é o quesito adequação (ZHU; HALL; MAY, 1997). Dentro desta temática, é válido ressaltar os quesitos que os testes unitários precisam atender em termos de noção de adequação:

- Cobertura de linha, pois espera-se que os métodos de teste unitário elaborados para testar uma determinada unidade exercitem todas as linhas de código da unidade sob teste.
- Cobertura de caminho (*path*), pois espera-se que os métodos de teste unitário elaborados exercitem minimamente uma vez cada caminho contemplado na unidade sob teste. Este tipo de cobertura é tratado quando se tem pontos de decisão no código da unidade sob teste.

2.3.5 Utilização de ferramentas de apoio para elaboração de testes unitários

Outro aspecto importante na construção e verificação da qualidade dos testes unitários é o uso de ferramentas de apoio (PERSCHEID; CASSOU; HIRSCHFELD, 2012).

Primeiramente, para que os testes unitários sejam eficazes na identificação de comportamento inadequado de uma determinada unidade, os métodos de teste unitário devem abranger o tanto quanto possível o código do sistema, conforme comentado anteriormente sobre a adequação centralizada nas coberturas de linha e de caminho. Em segundo lugar, os desenvolvedores precisam executar os métodos de teste unitário tão frequentemente quanto possível e assim, a execução deve ser automatizada e rápida.

Existem diversos *frameworks* para implementação de testes unitários para as mais variadas linguagens de programação. A exemplo disso, tem-se o *JUnit* para Java, *CUnit* para linguagem C e o *Rspec* para a linguagem Ruby. Todas estas ferramentas provêem suporte para elaboração de métodos de teste unitário bem como para a execução automatizada da suíte de testes construída.

Adicionalmente, é importante visualizar a cobertura de código (BERGEL; PEñA, 2012). Não basta apenas elaborar uma suíte de testes unitários e executá-la de forma automática, também é necessário avaliar o quanto os métodos de teste unitário estão exercitando o código da unidade sob teste.

A partir do uso de ferramentas de análise de cobertura, é possível identificar mais cenários de teste a serem elaborados e assim, a suíte de testes se torna ainda mais eficaz. Todos os aspectos listados até aqui elevam a qualidade do produto e assim, tem-se a entrega de maior valor para o cliente.

2.4 Engenharia de *Software* Baseada em Valor

O objetivo da Engenharia de *Software* é criar produtos, serviços e processos que agreguem valor (BIFFL et al., 2014). Mas, o que é valor de fato? O dicionário moderno de sociologia define valor como o princípio generalizado de comportamento em que os membros de um grupo sentem um forte compromisso em prover padrão para julgar atos e objetivos específicos. A definição é aplicável nas mais diversas áreas, inclusive no desenvolvimento de *software*.

O primeiro texto significativo que trouxe considerações sobre valor no âmbito do desenvolvimento de *software* foi de Boehm, em 1981. Na obra *Software Engineering Economics*, Boehm enfatiza o aspecto de que as equipes de projetos de desenvolvimento de *software* sempre irão se deparar com recursos limitados. Não haverá tempo ou dinheiro suficientes para cobrir todas as funcionalidades pretendidas.

Como mencionado na Introdução, a VBSE traz considerações de valor para o primeiro plano. Caso as perspectivas de valor não sejam explicitadas e conciliadas entre os envolvidos em um projeto, todos perdem ao final.

O produto de *software* reconhecidamente possui características internas e externas particulares, sendo portador de uma natureza altamente flexível e volátil. Nesse sentido, há uma forte dependência da colaboração entre pessoas, com níveis de criatividade e qualificações diferenciadas. Faz-se necessárias então, uma construção e gestão mais rigorosas.

Para compreender melhor o que a VBSE propõe, pode-se considerar o exemplo da elaboração de um novo *software* para ser utilizado pelos clientes de uma determinada instituição bancária, citado em (BIFFL et al., 2014). Inicialmente, a equipe de negócio

do banco explica para a equipe de projeto quais são as funcionalidades que o *software* deverá contemplar e dentre elas, elegem as mais significativas. Neste ponto, a equipe de negócio explicitou suas proposições de valor, ou seja, o que é mais importante para ela. Posteriormente, a equipe de projeto planeja todas as iterações do projeto visando a construção e entrega dos componentes mais críticos do *software* para as primeiras iterações. Assim, caso haja alguma restrição de orçamento ou recursos, as funcionalidades mais importantes já terão sido entregues, levando em consideração que estas também já terão sido efetivamente testadas e inspecionadas. Por fim, caso a equipe de negócio decida solicitar uma nova funcionalidade devido ao fato de uma instituição concorrente ter lançado um *software* que seja portador desta funcionalidade, o planejamento do projeto poderá ser redesenhado, atendendo, novamente, às proposições de valor da equipe de negócio.

Assim, pode-se exemplificar a relação entre VBSE e a atividade de teste de *software*, por exemplo. O desafio em teste baseado em valor consiste em integrar as duas dimensões (a interna, que abrange custos e benefícios do teste e a externa, que foca nas oportunidades e riscos). Adicionalmente, deve-se alinhar o processo interno de teste com os objetivos de valor oriundos dos clientes e mercado (BIFFL et al., 2014).

Considerando este raciocínio, não é diferente no caso da inspeção de código. As atividades mais técnicas, internas de um projeto de desenvolvimento de *software*, devem estar alinhadas à missão do projeto como um todo, corroborando expectativas inerentes às proposições de valor externalizadas pelos clientes.

3 Metodologia

A metodologia concebida para a realização deste trabalho foi classificada quanto à sua natureza, abordagem, objetivos e aos procedimentos técnicos.

Quanto à natureza, a pesquisa possui uma caracterização aplicada, visto que se objetiva gerar conhecimentos de efeito prático e destinados à resolução de problemas específicos. Quanto à abordagem, a pesquisa possui um enfoque qualitativo, devido ao fato de não requerer a utilização de métodos e técnicas estatísticas (MORESI, 2003). Quanto aos objetivos, a pesquisa se caracteriza como descritiva, pois pretende definir um *framework*.

Para concretizar a realização do trabalho, os seguintes procedimentos técnicos foram adotados:

- Pesquisa Bibliográfica
- Revisão Sistemática
- Pesquisa-Ação

Pelo fato de o presente trabalho estar centrado na proposição de um *framework* que reúne práticas propostas pela verificação de *software* e integra estas às diretrizes fornecidas pela VBSE, fez-se necessária a seleção de mais de um procedimento técnico de pesquisa para a construção do trabalho.

O arcabouço teórico inerente à VBSE e às inspeções de código foi obtido a partir da pesquisa bibliográfica.

Para Fonseca (2002), a pesquisa bibliográfica é realizada a partir do levantamento de referências teóricas já analisadas, e publicadas por meios escritos e eletrônicos. A pesquisa bibliográfica possibilita que o pesquisador conheça o que já se estudou sobre a temática.

A revisão sistemática, por sua vez, se estabelece como uma metodologia bem definida para identificar, analisar e interpretar as melhores abordagens e práticas relacionadas a uma determinada questão de pesquisa (KITCHENHAM, 2007). Além desses aspectos, é uma metodologia que possibilita uma repetição concisa.

A opção pelo procedimento da revisão sistemática deve-se ao fato de que este trabalho também está voltado para a tentativa de compreender quais tem sido as abordagens empregadas na elaboração de testes unitários e como se tem avaliado a efetividade destes. O *framework*, naturalmente, compreende a prática de elaboração de testes unitários.

Outro aspecto interessante da revisão sistemática é que esta auxilia de maneira precisa na localização dos principais trabalhos publicados para uma determinada problemática, favorecendo a construção de uma linha cronológica que demonstra ao pesquisador quais linhas têm sido defendidas e quais são os campos mais promissores para uma futura exploração.

Neste trabalho, para a execução da revisão sistemática, considerou-se a proposta de Kitchenham para a revisão no âmbito da Engenharia de *Software*. A proposta é composta por três fases:

- Planejamento da revisão.
- Condução da revisão.
- Relato da revisão.

A fase de planejamento consiste na identificação da necessidade de se desenvolver uma revisão sistemática, bem como elaborar um protocolo de revisão. Logo em seguida, na fase de condução, os objetivos mais pontuais de pesquisa são identificados e, adicionalmente, estudos são selecionados e seus dados são extraídos e analisados. Por fim, na fase de relato, os dados extraídos e analisados na fase anterior são externalisados e elabora-se uma discussão. Maiores detalhes acerca da revisão sistemática executada neste trabalho podem ser encontrados no Apêndice A.

3.1 Detalhamento do Plano Metodológico

O plano metodológico concebido para a formulação e aplicação do *framework* aqui proposto se subdivide em duas partes, sendo a primeira atrelada ao TCC 1 (Trabalho de Conclusão de Curso 1) e a segunda, ao TCC 2.

A primeira parte compreende 3 fases: *Planejamento da Pesquisa*; *Coleta de Informações* e *Proposta de Solução*.

A fase inerente ao *Planejamento da Pesquisa* envolve a definição de objetivos, pergunta de pesquisa, classificação metodológica e estabelecimento dos procedimentos técnicos de pesquisa. É válido evidenciar que essas definições são realizadas a partir do problema que foi identificado.

A fase de *Coleta de Informações* compreende a aplicação dos procedimentos técnicos adotados para a pesquisa, que para este trabalho foram a pesquisa bibliográfica e a revisão sistemática.

Por fim, na fase *Proposta de Solução*, há a concepção de uma possível solução para a problemática identificada. Esta solução, por sua vez, está embasada nas informações coletadas na fase anterior.

A figura a seguir ilustra essa parte do plano metodológico.

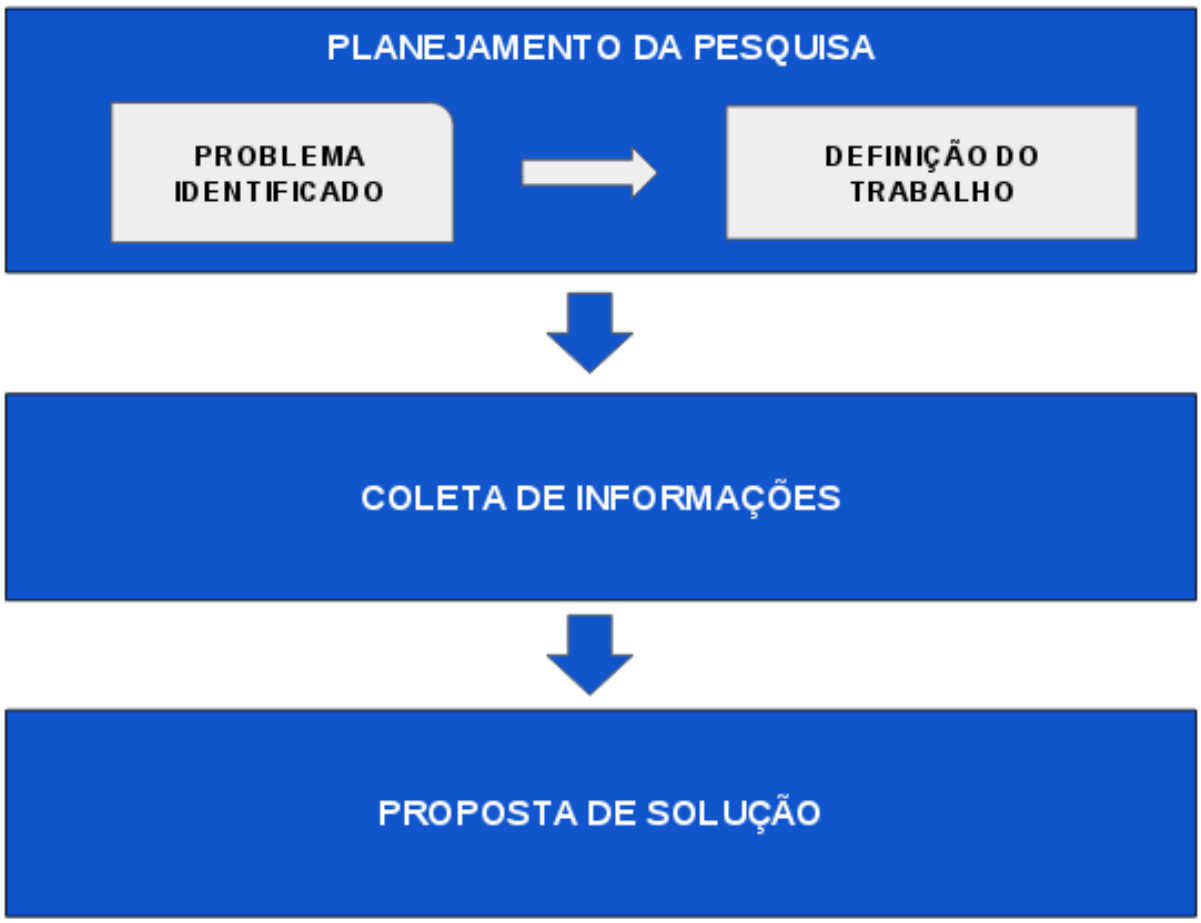

Figura 1: Plano Metodológico - TCC 1

A segunda parte, já vinculada ao TCC 2, envolve outras 3 fases: *Coleta de Dados*; *Análise e Interpretação dos Resultados* e *Divulgação dos Resultados*, conforme ilustrado na Figura 2. A fase *Coleta de Dados* envolve a aplicação de um procedimento técnico denominado pesquisa-ação. Esse procedimento possibilita que o pesquisador intervenha dentro da problemática, mobilizando os participantes e construindo novos conhecimentos (ANDALOUSSI, 2004). Assim, a cada ciclo novos quesitos poderão ser melhor tratados no *framework*.

Na fase *Análise e Interpretação dos Resultados*, todos os dados coletados a partir da utilização do *framework* serão analisados, de forma que todas as percepções possíveis possam ser obtidas. Por fim, na fase *Divulgação dos Resultados*, haverá uma estruturação de todos os dados obtidos e analisados, de forma que possa ser evidenciado a efetividade do uso do *framework*.

A figura a seguir ilustra a segunda parte do plano metodológico.

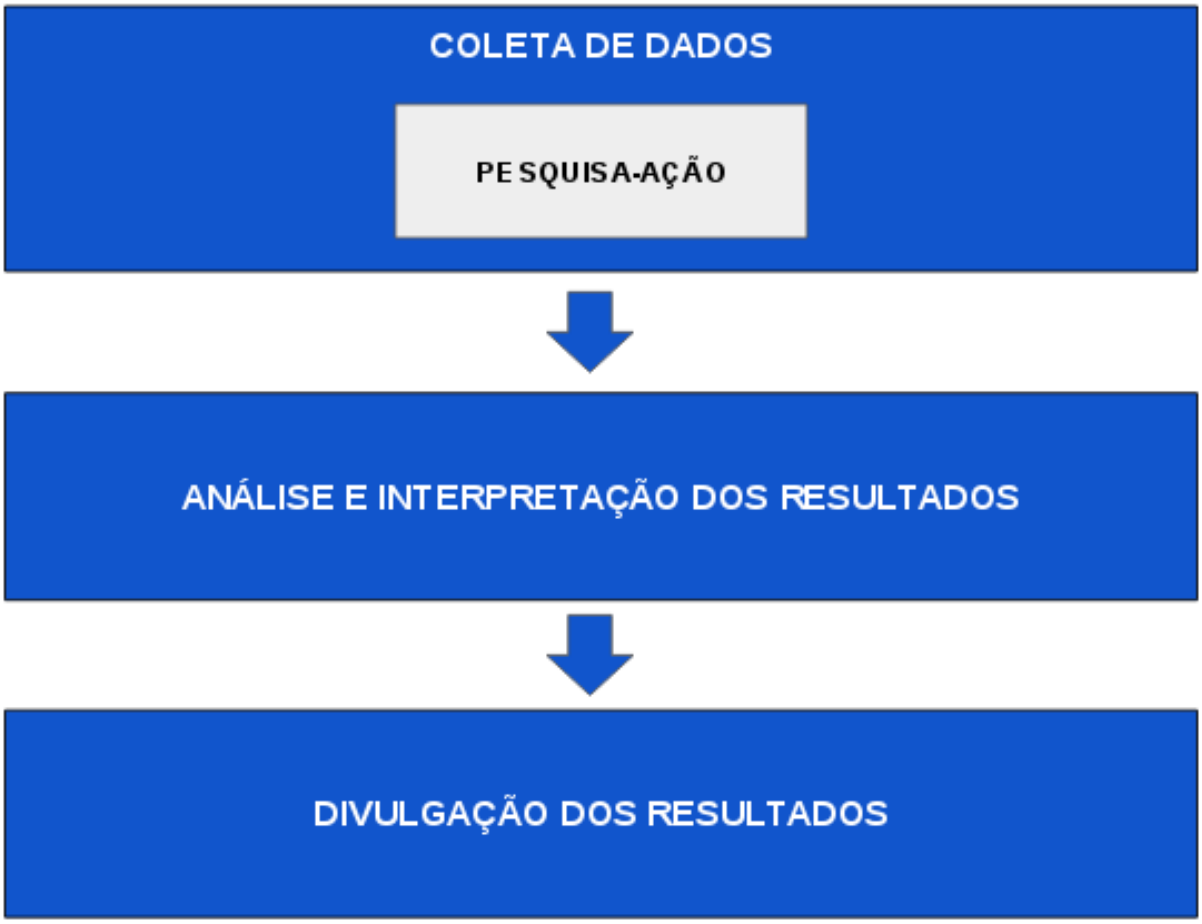

Figura 2: Plano Metodológico - TCC 2

3.2 Elaboração da Proposta do *Framework*

A primeira parte do plano metodológico já foi executada durante a realização deste trabalho. Com base no problema identificado e nas informações levantadas por meio da pesquisa bibliográfica e da revisão sistemática, foi possível conceber o *framework* de avaliação da qualidade de código. Adicionalmente, já foi definido como a avaliação do *framework* será feita. As próximas seções descrevem detalhadamente este quesito que, por sua vez, constitui a segunda parte do plano metodológico, a ser executado no TCC 2.

3.3 Estratégia de Aplicação do *Framework*

Antes da incorporação das atividades propostas pelo *framework* à rotina das organizações que serão utilizadas como casos, será feita uma reunião com os principais envolvidos de cada projeto. Assim, serão explicitados todos os aspectos do *framework* e a importância dessas atividades para que haja aumento na qualidade do *software*.

Adicionalmente, será construída uma página na wiki ou até mesmo uma simples página web nos ambientes dos projetos que irá dispor a imagem do processo que constitui o *framework*, bem como o detalhamento deste, de forma que os *stakeholders* dos projetos possam efetuar consultas em caso de dúvidas.

Como mencionado em seção anterior, a pesquisa-ação será o procedimento técnico empregado para analisar o uso do *framework*.

3.4 Organizações onde o *Framework* será aplicado

3.4.1 Controladoria Geral do Distrito Federal

A Controladoria Geral do Distrito Federal (CGDF) teve sua estrutura criada por meio do Decreto nº 36.236, de 1º de janeiro de 2015, pelo Governador Rodrigo Rollemberg. Esta Unidade Gestora possui como missão orientar e controlar a gestão pública, com transparência e participação da sociedade.

Interna à CGDF, há uma coordenação de assuntos tecnológicos, a COTEC, que é responsável por desenvolver e manter sistemas para todas as demais coordenações da CGDF. Além dos sistemas internos, tem-se aplicações que são desenvolvidas por empresas privadas ao governo e entregues à COTEC para serem mantidas.

A equipe da COTEC é pequena, contando com um total de 10 servidores. São 3 servidores na frente de infraestrutura, 3 na frente de administração de banco de dados e apenas 4 na área de desenvolvimento. O *Scrum* e XP foram adotados recentemente na COTEC.

A maior parte dos sistemas desenvolvidos e mantidos pela COTEC são aplicações *Web*, construídas com as linguagens de programação *CSharp* e Java. Adicionalmente, faz-se uso do *framework AngularJS* para elaboração da camada de apresentação das aplicações.

Embora já se tenha o mínimo de atividades para controlar o desenvolvimento, a COTEC ainda necessita incorporar uma série de outras práticas. As atividades inerentes à verificação de *software* ainda são incipientes na COTEC, tornando-a um local propício para a aplicação do *framework*.

3.4.2 Laboratório Fábrica de *Software* - Campus UnB Gama

O Laboratório Fábrica de *Software* foi concebido com fins de pesquisa e promoção de excelência no desenvolvimento de *software*. Atualmente, o laboratório já incorpora metodologias ágeis para o desenvolvimento, sendo o *Scrum* e o XP (*Extreme Programming*).

Adicionalmente, o laboratório atua de forma a unir o aprendizado das disciplinas

do curso de Engenharia de *Software* do Campus Gama da UnB, e a atuação prática de projetos reais.

O laboratório possui parceria com instituições públicas e privadas, as quais subsidiam pesquisas e projetos utilizando tecnologia *CSharp* e *Android*. Quando necessário, o laboratório também modela processos de negócio utilizando notação BPMN (*Business Process Model and Notation*).

Por se tratar de uma instituição aberta ao âmbito de pesquisa na área de Engenharia de *Software*, o laboratório é um local apropriado para uso e experimentação do *framework*, justamente pelo fato de conciliar interesses práticos do mercado com os fins acadêmicos.

3.5 Avaliação da Efetividade do *Framework*

Para realizar futuramente uma análise da efetividade da aplicação do *framework*, elaborou-se um GQM (*Goal Question Metrics*). A seguir, tem-se uma tabela que resume o objetivo da medição.

Analisar	A efetividade do *framework*
Com o propósito de	Melhorar
Com respeito a	Efetividade do *framework*
Sob o ponto de vista de	*Stakeholders* do projeto
No contexto de	Aumento da qualidade do *software*

Tabela 1: Tabela Resumo do Objetivo de Medição

A partir do quadro resumo que descreve o objetivo de medição, foram elaboradas questões a serem respondidas por meio das medições, bem como as métricas associadas às mesmas. A seguir, tem-se a apresentação das questões e métricas.

- **Questão 1:** A densidade de defeitos diminui ao longo das *sprints*?

  Densidade de Defeitos por Unidade do Código

- **Questão 2:** Qual o nível de eficácia dos testes unitários implementados a partir do uso do guia de implementação do *framework*?

  Taxa de acertos por linha de código (Métrica fornecida pelas ferramentas de análise de cobertura)

- **Questão 3:** Qual o número de falhas reportados pelos usuários após homologação da *release* entregue?

  Número de Falhas

- **Questão 4:** Qual o grau de satisfação dos desenvolvedores ao utilizarem os guias de inspeção e de implementação de testes unitários propostos pelo *framework*?

  Índice de Satisfação

- **Questão 5:** Qual o grau de facilidade percebido quando vai ser feita uma manutenção no código?

  Índice de Manutenibilidade

A coleta de métricas será realizada ao final de cada *sprint*, exceto o índice de satisfação e também, número de falhas (esta métrica só será coletada ao final das entregas de *release*), de maneira a avaliar os resultados e verificar a efetividade do *framework*. Caso ajustes sejam necessários, as modificações serão feitas e novamente, todo o ciclo de atividades será executado na *sprint* seguinte.

Adicionalmente, para o correto uso do *framework*, propõe-se uma etapa a mais no kanban de implementações da equipe de desenvolvimento, sendo esta a inspeção de código. A funcionalidade só será considerada finalizada, após inspecionada e aprovada por outro integrante da equipe.

3.6 Pesquisa de Satisfação dos Desenvolvedores

Para verificar o nível de satisfação dos desenvolvedores, concebeu-se uma pequena lista com itens a serem julgados pelos desenvolvedores de acordo com as opções disponíveis, citadas como componentes da métrica que responde à questão 4. A seguir, tem-se os itens de pesquisa.

- As inspeções de código se mostraram eficazes na identificação de defeitos no código?
- Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?
- As inspeções de código efetuadas segundo o guia e as implementações de testes unitários também feitas de acordo com o guia demonstraram-se onerosos ao processo de desenvolvimento?
- O número de falhas percebidos pelo usuário passou a ser menor a cada *release* entregue?
- A manutenibilidade do código ficou melhor após as inspeções de código?

Será feito o cômputo das respostas atribuídas pelos desenvolvedores aos itens da pesquisa de satisfação usando a escala *Likert*:

- 1 - Concordo Totalmente.
- 2 - Concordo Parcialmente.
- 3 - Discordo Parcialmente.
- 4 - Discordo Totalmente.

É importante ressaltar que a pesquisa será aplicada ao final das entregas de *release* e não ao final de cada *sprint*. Dessa maneira, os desenvolvedores poderão assimilar mais experiências para fornecerem respostas mais concisas ao julgar os itens da pesquisa.

4 *Framework* de Avaliação da Qualidade do Código

Este capítulo apresenta, inicialmente, o detalhamento das atividades do *framework*. Por fim, apresenta-se os *checklists* de implementação de testes unitários e inspeção de código obtidos a partir do referencial teórico.

4.1 Detalhamento do *Framework*

Com base em todos os conceitos assimilados a partir da realização da revisão sistemática de literatura e também, da pesquisa bibliográfica, foi possível conceber um grupo de atividades que em sua totalidade compõem o *framework* de avaliação de código.

Iniciando a descrição das atividades pelo macroprocesso, tem-se, conforme ilustrado na figura 3:

- **Elicitar as proposições de valor:** Caracteriza-se como uma reunião entre todos os envolvidos no projeto (área de negócio e área técnica) para alinhamento da missão do projeto. Nesta reunião, a área de negócio irá descrever quais seriam as funcionalidades mais críticas e, portanto, que mais agregam valor em seu contexto, para o *software* a ser construído, de forma que a equipe técnica possa efetuar um planejamento embasado neste aspecto.
- **Estabelecer os objetivos de teste e metas gerais de qualidade de código:** Nesta atividade, a equipe técnica irá conceber uma estratégia para testar e assegurar a qualidade do código do *software* levando em consideração os módulos priorizados a partir da elicitação das proposições de valor.
- **Planejar sprints de implementação e execução dos testes e inspeções de código:** Nesta atividade, o gestor do projeto, juntamente com os representantes dos demais grupos técnicos existentes no processo de desenvolvimento irão planejar as sprints, definindo início e término das mesmas, bem como os módulos do código que serão testados e inspecionados de acordo com uma priorização obtida a partir das proposições de valor.
- **Verificar o planejamento da sprint a ser iniciada:** Nesta atividade, a equipe do projeto irá verificar as funcionalidades a serem desenvolvidas e o que deverá ser plenamente testado e inspecionado. Assim, fará a alocação apropriada dos recursos (humanos e financeiros) para atendimento dos objetivos da sprint.

- **Executar Sprint:** Subprocesso que compreende as atividades corriqueiras de uma sprint segundo o Scrum, contudo, acrescentando as atividades agregadas pelo *framework*.
- **Revisar resultados da sprint:** Nesta atividade, a equipe técnica irá verificar se todos os objetivos estabelecidos para a sprint foram alcançados, ou seja, se de fato o que foi planejado foi executado. Caso seja necessário efetuar otimizações nos planejamentos, devido também à mudança de alguma proposição de valor, a primeira atividade deverá ser executada novamente. É válido ressaltar que a interação com o cliente deve ser intensa para promover a agregação de valor em um grau satisfatório. Adicionalmente, caso algum item fique pendente, este entrará imediatamente como algo a ser concluído na sprint subsequente.
- **Efetuar revisão geral do projeto:** Nesta atividade, haverá uma reunião entre as áreas de negócio e técnica e todas as proposições de valor serão apresentadas de forma breve, explicitando o planejamento e a execução. Caso alguma proposição fique pendente, será avaliada a possibilidade de prorrogação de prazo para término do projeto.

A figura a seguir ilustra o macroprocesso do *framework* proposto.

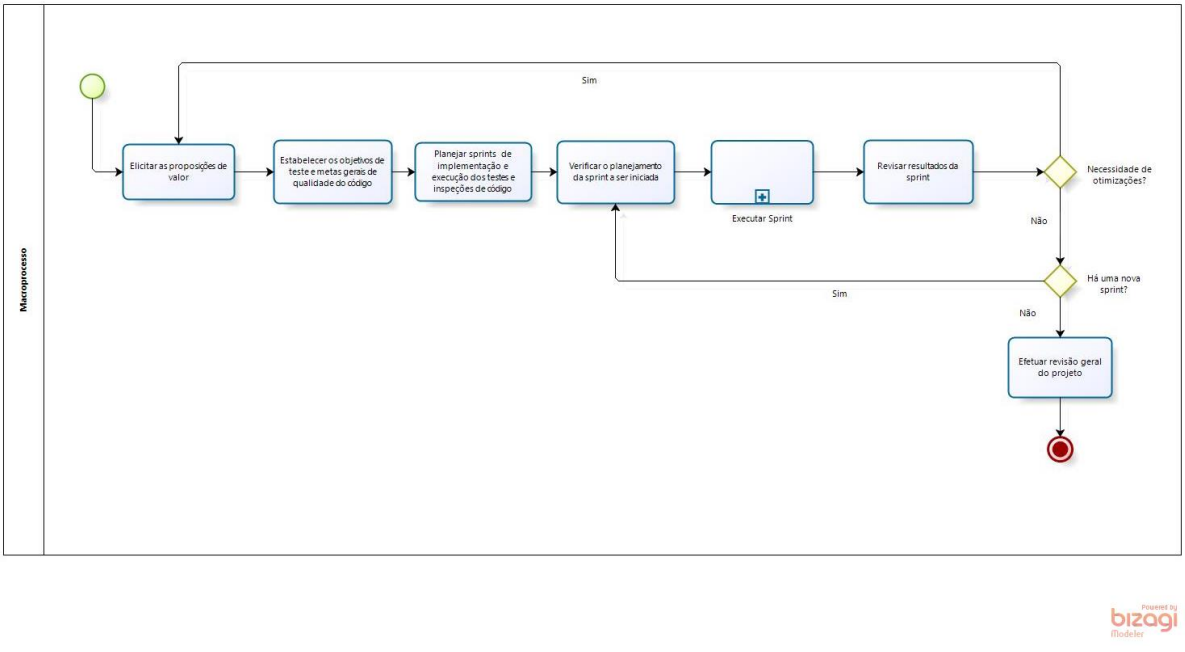

Figura 3: Macroprocesso - *Framework de Avaliação de Código*

A partir da descrição das atividades, é possível perceber o quão importante é fazer um bom planejamento e priorização. Caso haja mudanças no escopo e também nos prazos, a princípio, as funcionalidades que mais agregam valor já terão sido cobertas pelos instrumentos de garantia de qualidade.

Quanto à descrição das atividades constituintes do subprocesso Executar Sprint, tem-se, conforme ilustrado pela Figura 4:

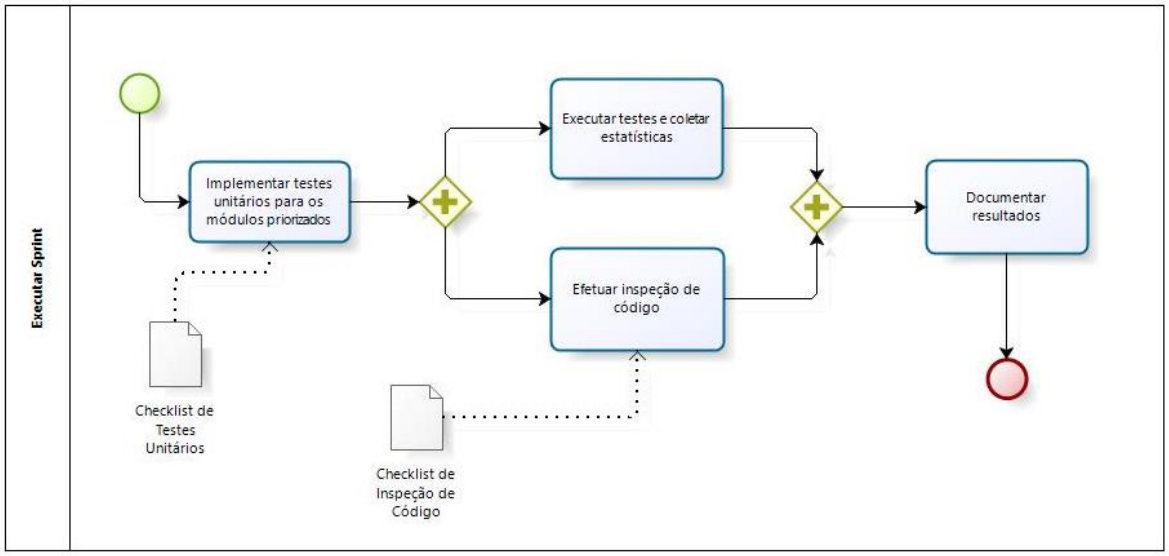

Figura 4: Subprocesso - *Executar Sprint*

- **Implementar testes unitários para os módulos priorizados:** Esta atividade possui como entrada um *checklist* de testes unitários, construído a partir da constatação de práticas utilizadas na indústria de desenvolvimento e que tem sido eficiente neste sentido. Assim, para a implementação dos testes unitários, espera-se o completo atendimento dos itens constantes neste *checklist*.

- **Executar testes e coletar estatísticas:** Nesta atividade, a suíte de testes unitários será executada e aspectos pertinentes à execução desta serão coletados, de forma a verificar performance, número de defeitos detectados etc.

- **Efetuar inspeção de código:** Nesta atividade, deverá ser feita uma inspeção do código tanto dos módulos sob teste, como também do código dos próprios métodos de teste unitário, de forma a verificar a completude e atendimento aos itens constantes no *checklist* de inspeção de código.

- **Documentar resultados:** Além da anotação das estatísticas coletadas durante a execução dos testes, será documentado o resultado da inspeção, explicitando para cada módulo avaliado o grau de qualidade. Em caso de detecção de erros e defeitos severos, o módulo não será aprovado, ou seja, considerado finalizado e assim, deverá ser feito o aperfeiçoamento do mesmo na sprint subsequente.

4.2 *Checklist* para Implementação de Testes Unitários

O *checklist* de implementação de testes unitários é composto dos seguintes itens, conforme a seção 2.3:

1. Foram criados métodos de teste pequenos (máximo de 8 linhas de código)?
2. Foram criadas convenções de nomenclatura para o código de teste (as classes, bem como os métodos de testes possuem nome condizente com o módulo que está sendo testado)?
3. A suíte de testes é executada com êxito independentemente da ordem dos métodos de teste em execução?
4. Os testes são auto verificáveis (já são utilizados mecanismos como assertivas e *expected* de forma que a própria ferramenta de teste seja capaz de dizer se o teste passou ou não)?
5. O código de teste segue uma estruturação hierárquica, sendo a mesma observada nas classes que constituem os componentes do *software*?
6. As funções mais internas aos métodos públicos das classes sob teste estão sendo devidamente testados pela suíte?
7. Os *stubs* são simples, pequenos e independentes de outros *stubs* que simulam comportamentos de outros módulos?
8. Foi utilizada ferramenta de análise de cobertura de código para verificar as linhas de código de fato exercitadas pela suíte de testes?
9. Os métodos de teste elaborados foram capazes de exercitar todas as linhas e caminhos dos métodos da unidade sob teste?
10. Foi analisada a *performance* da suíte de testes quando esta foi executada, de forma a verificar consumo de memória e processamento?

4.3 *Checklist* para Inspeção de Código

O *checklist* para inspeção de código é composto dos seguintes itens, conforme seção 2.2:

1. Foi verificado o retorno de cada método?

2. Foi verificado como são feitos os tratamentos de interrupções, bem como gerenciamento de regiões críticas (algumas aplicações exigem sincronização entre processos, visto que utilizam recursos compartilhados)?

3. Foi verificado o comportamento de todos os trechos de código que são portadores de estruturas de repetição (*for*, *while* etc.)?

4. Foi verificado como estão sendo tratadas as operações de entrada e saída de dados nos métodos das classes?

5. Foi verificado o fluxo do programa, analisando as estruturas de controle(*if*, *else* etc.)?

6. Foram verificados todos os trechos de código inutilizados?

7. As atribuições de valores às constantes e às variáveis foram verificados?

8. Os comentários do código foram verificados, assim como a legibilidade do mesmo (tal como nome de variáveis, métodos etc.)?

9. Foram verificados os *imports* e qualquer outro tipo de inclusão de código de outras bibliotecas?

5 Considerações Finais

Durante a realização deste trabalho, foi possível executar a primeira parte do plano metodológico, onde identificou-se o problema e planejou-se a pesquisa. Posteriormente, coletou-se informações e uma solução foi proposta para a problemática identificada com base nas informações coletadas.

De fato, a revisão sistemática e a pesquisa bibliográfica evidenciaram a importância de se aplicar práticas da verificação de *software* para que seja alcançado êxito em um projeto. Além disso, a VBSE mostra-se como uma boa abordagem para alinhar atividades técnicas relacionadas ao desenvolvimento à área de negócio.

Como próximos trabalhos, espera-se aplicar a segunda parte do plano metodológico, executando uma pesquisa-ação para refinar o *framework* proposto neste trabalho.

Referências

ANDALOUSSI, K. E. *Pesquisas-ações: ciências, desenvolvimento, democracia*. [S.l.]: Edufscar, 2004. Citado na página 41.

ANICHE, M. F.; OLIVA, G. A.; GEROSA, M. A. What do the asserts in a unit test tell us about code quality? a study on open source and industrial projects. 2013. Citado 2 vezes nas páginas 33 e 34.

BERGEL, A.; PEñA, V. Increasing test coverage with hapao. 2012. Citado na página 36.

BIASI, L. B. Geração automatizada de drivers e stubs de teste para junit a partir de especificações u2tp. 2006. Citado na página 32.

BIFFL, S. et al. *Value-Based Software Engineering*. [S.l.]: Springer, 2014. Citado 5 vezes nas páginas 23, 24, 25, 36 e 37.

FILHO, W. de P. P. *Engenharia de Software - Fundamentos, Métodos e Padrões*. [S.l.]: LTC, 2009. Citado 2 vezes nas páginas 29 e 30.

GANESAN, D. et al. An analysis of unit tests of a flight software product line. 2013. Citado 2 vezes nas páginas 33 e 34.

JúNIOR, J. R. de A. et al. Best practices in code inspection for safety-critical software. 2003. Citado na página 30.

KANER, C. Software negligence and testing coverage. Dept of Computer Sciences, Florida Tech, 1995. Citado 2 vezes nas páginas 24 e 25.

KITCHENHAM. Guidelines for performing systematic literature reviews in software engineering. 2007. Citado na página 39.

KOSCIANSKI, A.; SOARES, M. dos S. *Qualidade de Software - Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software*. [S.l.]: Novatec, 2007. Citado 2 vezes nas páginas 23 e 24.

MORESI, E. *Metodologia de pesquisa*. [S.l.]: Universidade Católica de Brasília, 2003. Citado na página 39.

PERSCHEID, M.; CASSOU, D.; HIRSCHFELD, R. Test quality feedback improving effectivity and efficiency of unit testing. 2012. Citado na página 35.

RAMLER, R.; BIFFL, S.; GRüNBACHER, P. Value-based management of software testing. Software Competence Center Hagenberg and Vienna University of Technology and Johannes Kepler University Linz, 2014. Citado 2 vezes nas páginas 24 e 25.

SINGH, H.; BAWA, H. S. Management of effective verification and validation. 1995. Citado na página 29.

ZHU, H.; HALL, P. A. V.; MAY, J. H. R. Software unit test coverage and adequacy. 1997. Citado na página 35.

Apêndices

APÊNDICE A – Detalhamento da Revisão Sistemática

Este apêndice tem por objetivo detalhar como a revisão sistemática foi conduzida para obter informações acerca da implementação de testes unitários.

A.1 Questões de Pesquisa

Para a revisão sistemática, as seguintes questões de pesquisa foram concebidas:

- **Questão de Pesquisa 1:** Quais tem sido as abordagens empregadas para elaboração de testes unitários de qualidade?
- **Questão de Pesquisa 2:** Como se tem avaliado a qualidade dos testes unitários?

Para verificar a qualidade dos testes unitários implementados é necessário coletar e compreender as principais abordagens utilizadas na prática e que, por sua vez, se mostraram efetivas na tarefa de construção de testes unitários.

Nesta revisão sistemática, buscou-se, em primeiro lugar, identificar as abordagens empregadas na elaboração de testes unitários de qualidade. Como segundo aspecto, procurou-se verificar como se tem avaliado a qualidade dos testes unitários.

A.2 Protocolo de Revisão

As buscas foram feitas nas seguintes bibliotecas digitais:

- IEEE *Xplore Digital Library*
- ACM *Digital Library*
- *ScienceDirect*
- Biblioteca Digital Brasileira de Computação

Inicialmente, procurou-se definir uma string para a busca nas bases. Após três ciclos de refinamento, a string final estabelecida foi *Quality of unit tests* ***AND*** *unit test coverage* ***OR*** *unit test adequacy.*

A string de busca foi refinada conforme artigos que condiziam com a problemática eram encontrados e assim, suas palavras-chave eram utilizadas. É importante ressaltar que na base brasileira a mesma string foi utilizada, contudo, traduzida para a língua portuguesa.

Com relação à seleção dos artigos, os seguintes critérios foram estabelecidos:

- Artigos escritos em língua inglesa ou portuguesa.
- Artigos que contemplassem um relato de experiência da indústria, elencando práticas e técnicas, bem como uso de ferramentas.

Para a análise dos dados, os seguintes passos foram definidos:

- **Primeiro:** Extração das abordagens relatadas
- **Segundo:** Classificação das abordagens quanto à sua natureza

A.3 Condução da Revisão

Durante as buscas iniciais nas bases citadas, foram encontrados muitos artigos que retratavam testes unitários, mas não especificamente o quesito qualidade dos mesmos. Após inserção do termo *"unit test coverage"*, foi possível encontrar 2 (dois) artigos que abordavam o quesito qualidade dos testes unitários como aspecto central.

A partir destes artigos, o termo *"unit test adequacy"* foi inserido e assim, outros artigos que também tratavam do quesito qualidade de testes unitários foram encontrados. Levando em consideração as bases citadas anteriormente, a busca retornou um total de 65 (sessenta e cinco) artigos. Ao final do processo de busca e seleção, 8 (oito) artigos foram selecionados.