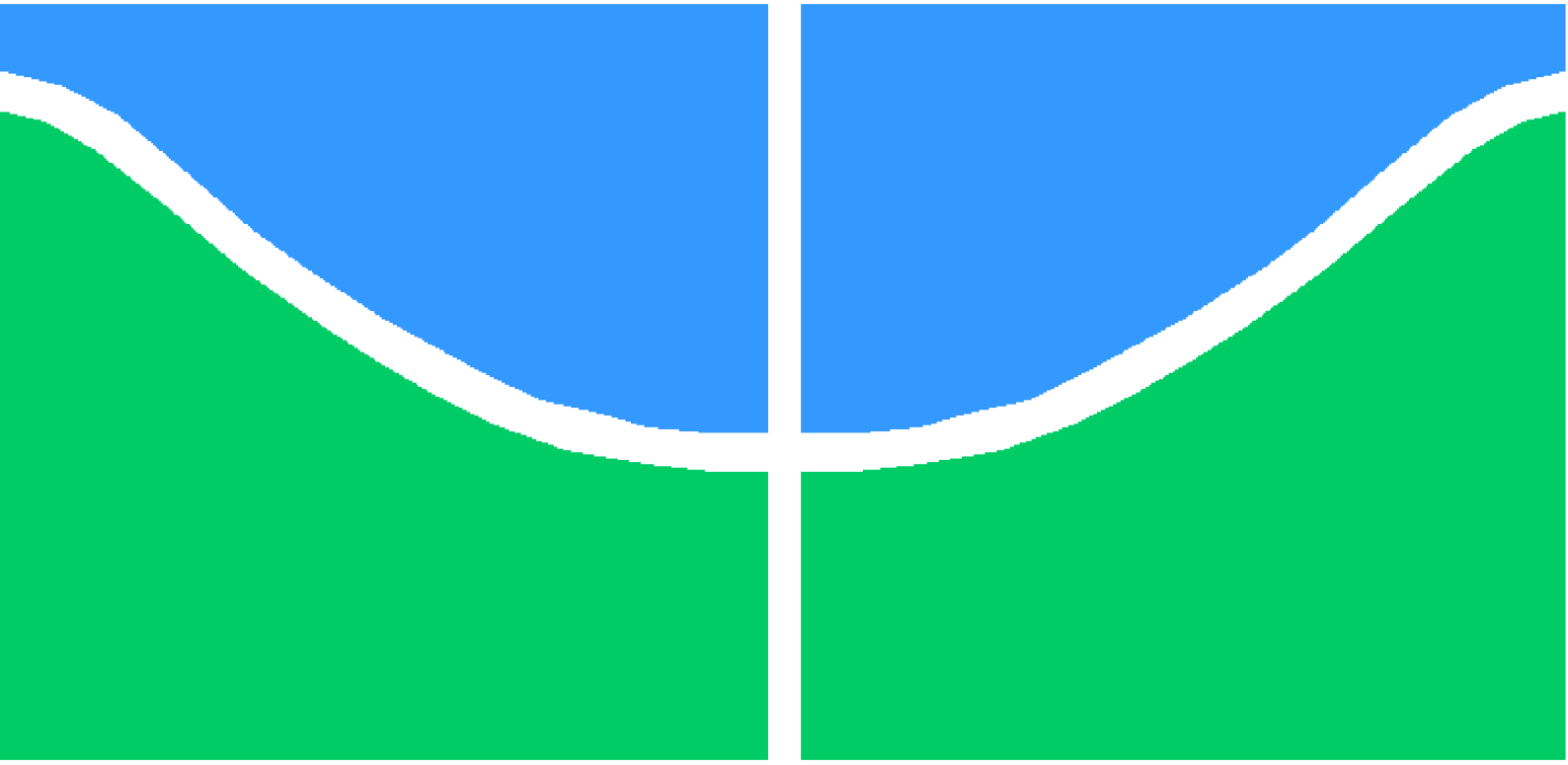

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Framework de Verificação Estática e Dinâmica de Código: Uma Abordagem Baseada em Valor

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

Brasília, DF
2017

Matheus Herlan dos Santos Ferraz

Framework de Verificação Estática e Dinâmica 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

2017

Matheus Herlan dos Santos Ferraz

Framework de Verificação Estática e Dinâmica de Código: Uma Abordagem Baseada em Valor/ Matheus Herlan dos Santos Ferraz. – Brasília, DF, 2017-

106 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª Msc. Elaine Venson

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2017.

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 de Verificação Estática e Dinâmica de Código: Uma Abordagem Baseada em Valor

CDU 02:141:005.6

Matheus Herlan dos Santos Ferraz

Framework de Verificação Estática e Dinâmica 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, 29 de junho de 2017:

Profª Msc. Elaine Venson
Orientador

Profª Msc. Cristiane Soares Ramos
Convidado 1

Prof. Dr. Sérgio Antônio Andrade de Freitas
Convidado 2

Brasília, DF
2017

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. Mediante este cenário, durante a execução da primeira parte do plano metodológico elaborado neste estudo, concebeu-se um *framework* que reúne um conjunto de atividades e práticas que favorecem a implementação de testes unitários e realização de inspeções correlacionado aos conceitos da Engenharia de *Software* Baseada em Valor, que aborda o alinhamento entre a missão do projeto e as atividades técnicas de desenvolvimento de *software*. Neste trabalho, o objetivo é apresentar a avaliação da efetividade do *framework* concebido na primeira parte deste estudo mediante o uso de um procedimento técnico denominado pesquisa-ação. Foram executados dois ciclos de pesquisa-ação para coletar os dados de execução dos projetos adotados para utilização do *framework*. Ao final será possível contemplar um *framework* de avaliação da qualidade de código comprovadamente adequado para uso em qualquer organização que faça uso de metodologias ágeis.

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

Abstract

Code production and implementation of unit tests, as well as code inspections, are insensitively connected. Unit tests and inspections, as complementary software verification practices, were designed with the aim 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. Through this scenario, during the execution of the first part of the methodological plan elaborated in this study, a framework was conceived that brings together a set of activities and practices that favor the implementation of unit tests and conducting inspections correlated to the concepts of Software Engineering Based on Value, which addresses the alignment between the project's mission and the technical activities of software development. In this work, the objective is to present the evaluation of the effectiveness of the framework conceived in the first part of this study through the use of a technical procedure called action research. Two action research cycles were executed to collect the execution data of the projects adopted to use the framework. At the end it will be possible to contemplate a code quality evaluation framework that is proven to be suitable for use in any organization that uses agile methodologies.

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 perspectiva, 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).

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.

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 princípios oriundos da VBSE, na avaliação da qualidade de código no contexto do desenvolvimento ágil?

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 princípios 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 princípios propostos 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 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.
- **Capítulo 5 - Materiais e Métodos**: apresenta uma caracterização detalhada dos projetos adotados para uso do *framework* e descrição de todas as ferramentas de trabalho utilizadas para coleta de informações.
- **Capítulo 6 - Resultados da Aplicação do *Framework***: apresenta os resultados obtidos a partir da execução dos ciclos de pesquisa-ação bem como uma análise dos dados coletados.

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 (1012 - IEEE *Standard for System and Software Verification and Validation*). Pelo CMMI (*Capability Maturity Model Integration*), a verificação é tida como uma confirmação de que produtos de trabalho refletem corretamente os requisitos especificados (CMMI-Dev V1.3 *Model*).

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 defeitos 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 outros itens, 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 desenvolvi-

mento 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 determinadas 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.

Também foi utilizada como fonte para o estabelecimento do *framework* uma revisão sistemática de literatura sobre qualidade de implementação de testes unitários, realizada na disciplina Engenharia de *Software* Experimental do curso de graduação em Engenharia de *Software* da Faculdade UnB Gama (FGA).

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 uso desta revisão sistemática deve-se ao fato de que este trabalho também está voltado para a tentativa de compreender quais têm 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.

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 1 ilustra essa parte do plano metodológico.

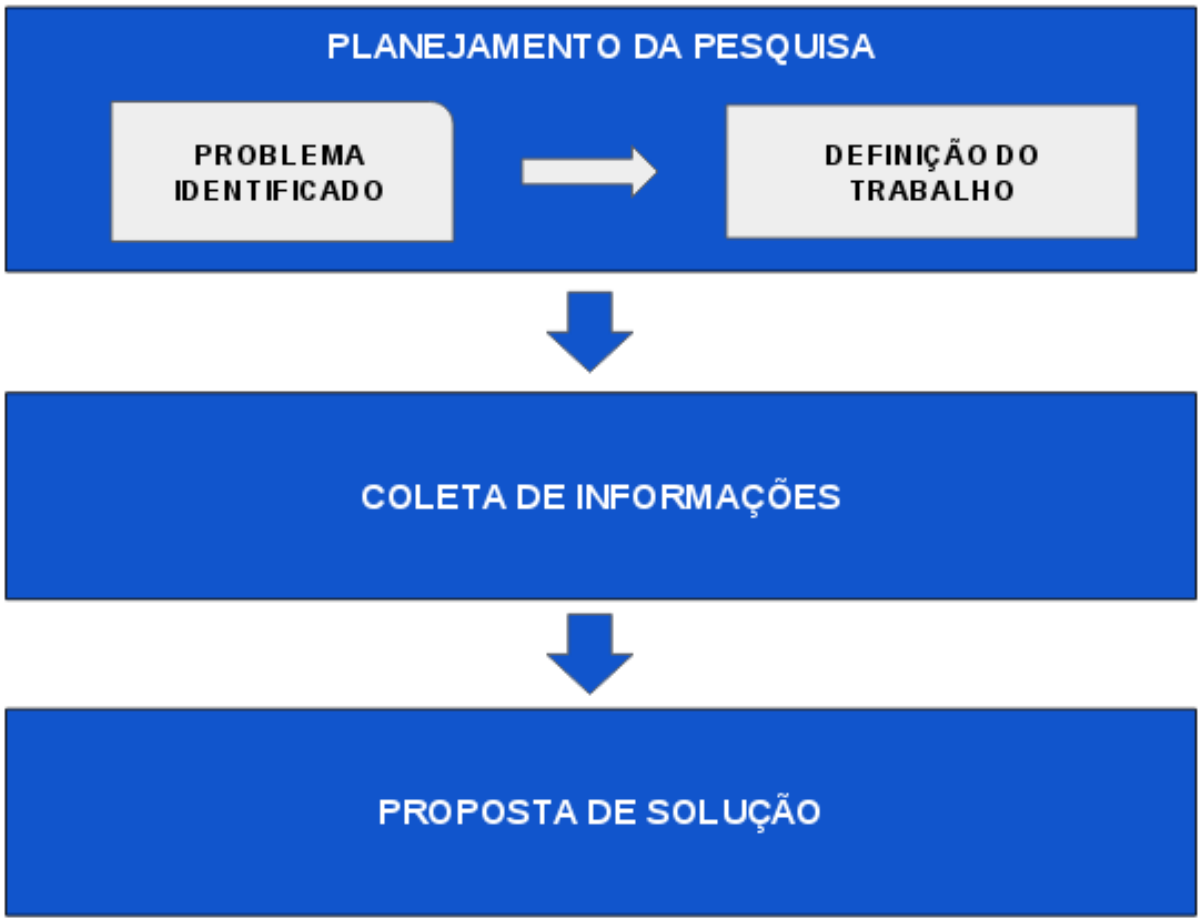

Figura 1: Plano Metodológico - TCC 1

A segunda parte, já vinculada ao TCC 2, e voltada para a implantação e avaliação da proposta, 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 2 ilustra a segunda parte do plano metodológico.

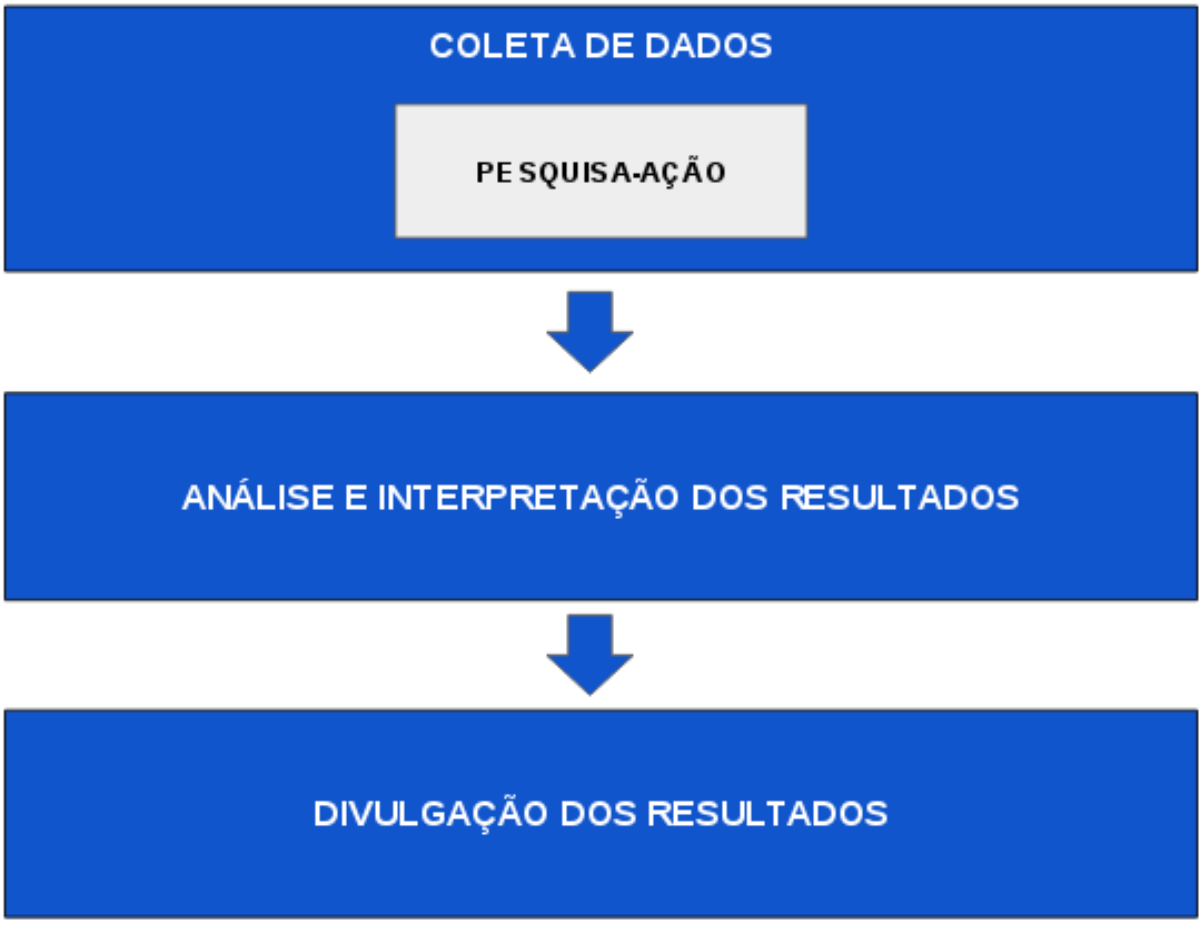

Figura 2: Plano Metodológico - TCC 2

3.2 Elaboração da Proposta do *Framework*

A primeira parte do plano metodológico foi executada durante a realização do TCC 1. Com base no problema identificado e nas informações levantadas por meio da pesquisa bibliográfica e da revisão sistemática, foi possível conceber o *framework* de verificação estática e dinâmica de código. Adicionalmente, já foi definido como a avaliação do *framework* seria feita. As próximas seções descrevem detalhadamente este quesito que, por sua vez, constitui a segunda parte do plano metodológico, executado durante este trabalho (TCC 2).

3.3 Estratégia de Aplicação do *Framework*

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

Adicionalmente, foi construída uma página na wiki dos ambientes dos projetos que dispõe da imagem do processo que constitui o *framework*, bem como o detalhamento deste, de forma que os *stakeholders* dos projetos pudessem efetuar consultas em caso de dúvidas.

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

3.4 Organizações onde o *Framework* foi 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, a COTEC (Coordenação de Administração de Assuntos Tecnológicos), 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*, Java e *Ruby* com auxílio do *framework Rails*. 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 uma análise da efetividade da aplicação do *framework*, elaborou-se um GQM (*Goal Question Metrics*) (BASILI; ROMBACH, 1994). A seguir, tem-se a tabela 1, que resume o objetivo da medição.

Tabela 1: Tabela Resumo do Objetivo de Medição

Analisar	A aplicação 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	Projetos de desenvolvimento e manutenção da Fábrica e da Controladoria

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:** Como a quantidade de defeitos do código fonte evolui ao longo das *sprints*?

  Número de Defeitos encontrados em Inspeções

- **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 em Produção

- **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 dos Desenvolvedores

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

  Índice de Manutenibilidade (Para Java e *CSharp*) e Flog (para *Ruby*)

As métricas relacionadas às questões 4 e 5 são detalhadas a seguir.

A coleta de métricas foi realizada ao final de cada *sprint*, de maneira a avaliar os resultados e verificar a efetividade do *framework*. Caso ajustes fossem necessários, as modificações eram feitas e novamente, todo o ciclo de atividades era executado na *sprint* seguinte.

Adicionalmente, para o correto uso do *framework*, foi proposta uma etapa a mais no kanban de implementações da equipe de desenvolvimento, sendo esta a inspeção de

código. A funcionalidade só seria 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?

Realizou-se o cômputo das respostas atribuídas pelos desenvolvedores aos itens da pesquisa de satisfação usando a escala *Likert*:

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

É importante ressaltar que a pesquisa foi aplicada ao final de cada *sprint*. Dessa maneira, os desenvolvedores puderam assimilar mais experiências para fornecerem respostas mais concisas ao julgar os itens da pesquisa.

Por fim, com relação ao índice de satisfação dos desenvolvedores, nenhuma fórmula específica foi desenvolvida. Considerou-se o percentual das alternativas da Escala *Likert* obtido para cada questão do questionário.

3.7 Índice de Manutenibilidade e Flog

Com relação às métricas adotadas por meio da aplicação do GQM, citadas anteriormente, esta seção traz detalhes acerca do Índice de Manutenibilidade.

O Índice de Manutenibilidade indica um valor entre 0 e 100 e representa a facilidade em manter o código de um determinado módulo. Um valor alto expressa melhor manutenibilidade. De acordo com o portal de métricas de código fonte da *Microsoft*, disponível em (https://msdn.microsoft.com/pt-br/library/bb385914.aspx), valores entre 20 e 100 expressam boa manutenibilidade. Valores entre 10 e 19 indicam manutenibilidade moderada e, por fim, valores entre 0 e 9 indicam pouca manutenibilidade.

A figura 3 expressa a fórmula utilizada para calcular o Índice de Manutenibilidade, que foi concebida por Paul Oman e Jack Hagemeister e apresentada pela primeira vez em 1992 na ICSM (*International Conference on Software Maintenance*).

$$MI = 171 - \left(5{,}2 * \ln(aveV)\right) - \left(0{,}23 * aveV(g')\right) - \left(16{,}2 * \ln(aveLOC)\right) + \left(50 * \sin\left(\sqrt{2{,}4 perCM}\right)\right)$$

Figura 3: Fórmula para o cálculo do Índice de Manutenibilidade

A fórmula considera o Volume de Halstead, que indica a facilidade de entender o código, considerando todos os operadores e operandos, a complexidade ciclomática, que expressa a quantidade de fluxos diferentes que podem ser obtidos, a quantidade de linhas de código e o percentual de linhas de comentário.

Com relação aos projetos em *Ruby*, foi necessário selecionar uma métrica mais apropriada para avaliar a manutenibilidade do código (Flog), pois o recurso de metaprogramação, fortemente presente em código *Ruby*, acrescenta particularidades para a análise de manutenibilidade.

A métrica Flog expressa uma pontuação que é fornecida ao código com base em três quesitos:

- Atribuições: quanto mais atribuições são feitas, maior é a pontuação obtida.
- Ramos (*Branches*): quando o código se ramifica, existem vários caminhos que podem ser seguidos. Esse aspecto aumenta a complexidade do código.
- Chamadas: a quantidade de chamadas efetuadas internamente entre os módulos afeta a complexidade do código, visto que existem muitas interfaces e passagens de parâmetros.

Para cada um dos quesitos citados acima há uma pontuação diferente associada. Por exemplo, uma chamada de método padrão recebe pontuação igual a 2, mas uma chamada *eval* já recebe pontuação igual a 5, mesmo que ambos sejam chamadas. Quanto mais complexo for o entendimento do código, mais alta será a pontuação. Há um site que contem a lista de pontuações categorizada para a métrica disponível em (http://docs.seattlerb.org/flog/). Com relação ao nível de aceitação proposto pelos idealizadores da métrica, um método com boa aceitação possui pontuação total inferior à 40.

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 4:

- **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 4 ilustra o macroprocesso do *framework* proposto.

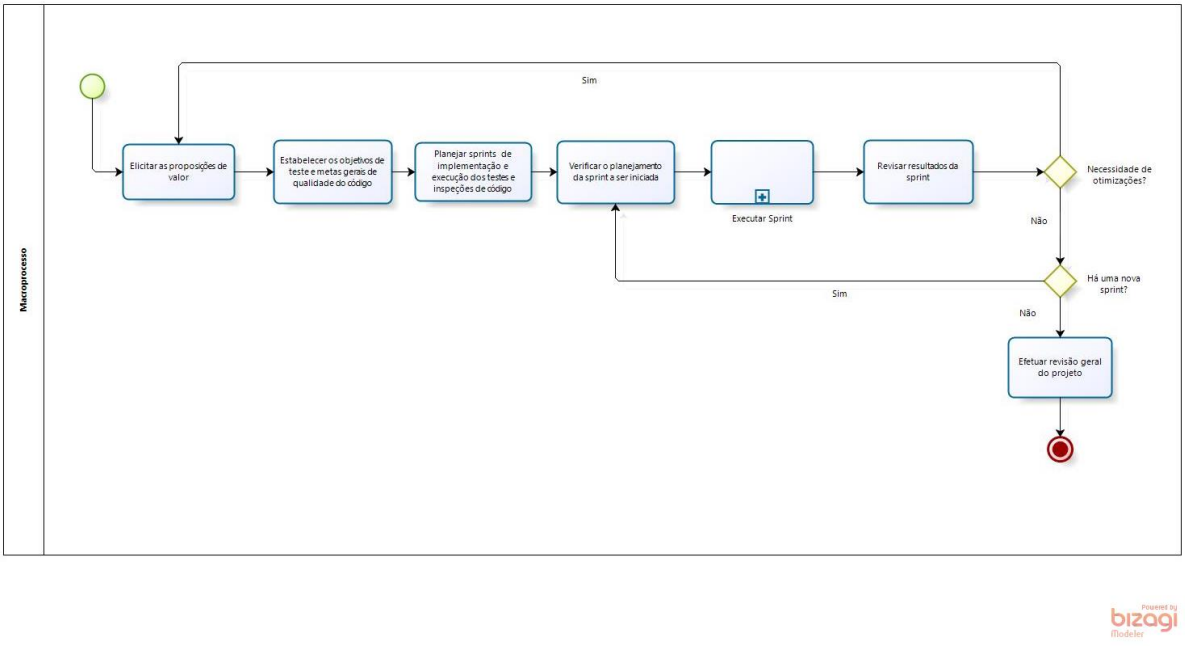

Figura 4: Macroprocesso - *Framework de Avaliação de Código*

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 5:

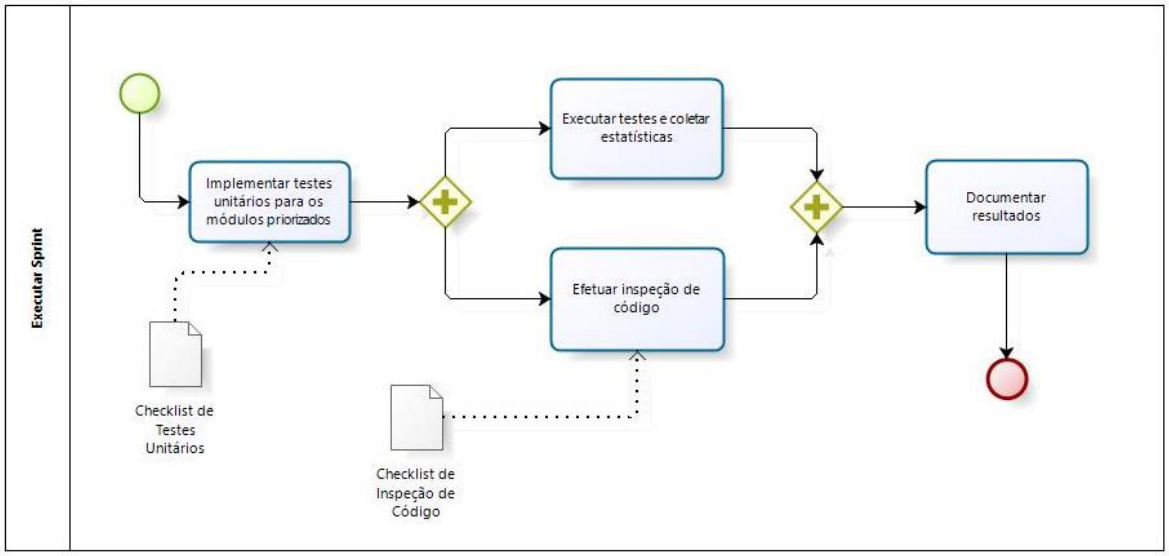

Figura 5: Subprocesso - *Executar Sprint*

- **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 Materiais e Métodos

Neste capítulo são descritos os procedimentos e materiais adotados no trabalho. Inicialmente, tem-se uma breve caracterização dos projetos selecionados nas organizações citadas na seção 3.4 inerente ao capítulo sobre metodologia. Posteriormente, são descritas todas as ferramentas utilizadas para desenvolvimento, gerenciamento e coleta de dados. Por fim, são apontadas as atividades executadas neste estudo de acordo com o procedimento técnico pesquisa-ação.

5.1 Projetos selecionados na CGDF

Pelo fato de o presente trabalho propor um *framework* que reúne as práticas complementares e fundamentais da verificação de *software* (testes e inspeções), bem como conceitos da VBSE, procurou-se selecionar produtos que possuem impacto significativo na missão da CGDF e que não possuíam tais práticas sendo utilizadas no seu processo de desenvolvimento.

5.1.1 Portal da Transparência do Distrito Federal

O primeiro produto selecionado foi o Portal da Transparência do Distrito Federal (www.transparencia.df.gov.br). Esta é uma ferramenta de participação da sociedade no controle das ações do Governo. Dentre as informações disponibilizadas pelo portal, destacam-se informações sobre a contabilidade do Governo do Distrito Federal, tais como: instrumentos de planejamento, receitas públicas arrecadadas pelo Governo, despesas públicas realizadas pelo Estado, licitações dos órgãos do Governo do Distrito Federal e remuneração dos servidores.

O Portal da Transparência do Distrito Federal já se encontra em sua terceira versão. De acordo com a CGDF, a reformulação do Portal, culminando em sua terceira versão, fez com que os acessos aumentassem em cerca de 34% em dezembro de 2016 e janeiro de 2017 em relação ao mesmo período no ano passado. A quantidade de usuários únicos no site, nesse intervalo, passou de 174.232 para 234.653 de acordo com o último levantamento.

A terceira versão do Portal foi concebida de forma a facilitar a navegação do usuário, bem como a compreensão dos dados disponibilizados. Adicionalmente, procurou-se desenvolver uma solução escalável, de forma que os dados pudessem ser utilizados por outros sistemas.

Desde o início da concepção arquitetural da terceira versão do Portal foram adotadas tecnologias modernas no âmbito do desenvolvimento de aplicações *web*. Basicamente, adotou-se o *framework AngularJS* para elaboração da aplicação cliente e o *Spring Framework*, desenvolvido em linguagem Java, para construção da API *REST (Representational State Transfer)* que é utilizada como *backend*. A seguir, tem-se a figura 6, que exibe a página inicial do Portal e a figura 7, que exibe um esquema da arquitetura do Portal.

Figura 6: Página inicial - Portal da Transparência do Distrito Federal

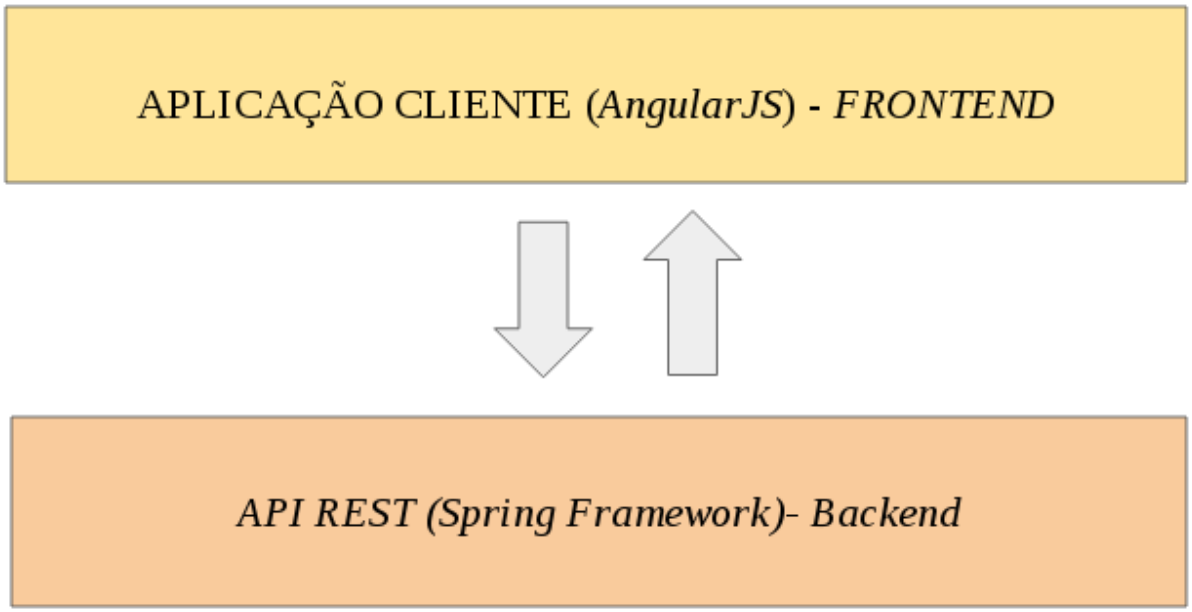

Figura 7: Esquema Arquitetural - Portal da Transparência do Distrito Federal

O Portal passa por constantes evoluções. Todos os dias dados são atualizados e novas funcionalidades são incorporadas em uma periodicidade de 3 semanas, tempo de

duração de uma *sprint* no processo de desenvolvimento da COTEC.

A partir de uma coleta inicial de métricas, o Portal apresentou índice de manutenibilidade médio igual a 55 no código constituinte da camada de *backend*. A cobertura de código era 0%, devido à inexistência de uma suíte de testes unitários. Adicionalmente, realizando um levantamento dos relatórios de falhas reportadas pela área de negócio acerca das *sprints* passadas, foi possível identificar um número médio de falhas igual a 5 a cada implantação em produção. Assim, percebeu-se a importância de utilizar concisamente práticas da verificação de *software* para melhorar a qualidade do Portal.

5.1.2 Sistema de Ouvidoria do Distrito Federal

O segundo produto selecionado foi o Sistema de Ouvidoria do Distrito Federal (www.ouv.df.gov.br). A Ouvidoria é um espaço que organiza a comunicação entre o cidadão e o governo, favorecendo a participação popular, a transparência e o aperfeiçoamento da prestação dos serviços públicos.

Como parte da estrutura da CGDF, existe a Subcontroladoria denominada Ouvidoria-Geral do DF. Esta é responsável pela coordenação dos trabalhos das Ouvidorias Especializadas - localizadas em cada órgão do Governo do Distrito Federal.

O sistema de ouvidoria também passou por uma reformulação, sendo que já se encontra em sua segunda versão. Dentre as melhorias realizadas na navegação, foram incluídas novas funcionalidades que fornecem maior transparência ao cidadão durante o atendimento de sua manifestação.

Assim como no Portal da Transparência, durante a concepção arquitetural da segunda versão do sistema de ouvidoria foram adotadas tecnologias modernas para o desenvolvimento. A camada de apresentação também utiliza o *framework AngularJS* e a camada *backend* foi construída utlizando a linguagem *CSharp* seguindo as diretrizes do estilo arquitetural MVC (*Model, View, Controller*) e também, do estilo de desenvolvimento DDD (*Domain Driven Design*). Posteriormente, a equipe do projeto optou por refatorar a camada de *backend*, passando a utilizar *Web API*.

O DDD, por sua vez, é uma abordagem que se alinha concisamente às ideias apresentadas pela VBSE, pois basicamente, um dos aspectos mais destacados nesta abordagem é que o código produzido deve estar bem alinhado ao negócio. Este conceito, se utilizado corretamente, favorece a reutilização pois os blocos de construção facilitam o aproveitamento de um mesmo conceito de domínio ou um mesmo código em vários lugares (EVANS, 2003). Adicionalmente, com um modelo bem feito e organizado, as diversas partes de um sistema interagem sem que haja muita dependência entre os módulos ou classes de objetos de conceitos distintos, acarretando em um quadro mínimo de acoplamento.

Com relação ao projeto da Ouvidoria, o cenário relativo à implementação de tes-

tes unitários era idêntico ao do Portal da Transparência, ou seja, cobertura de código em 0%. No âmbito do número médio de falhas reportadas pela área de negócio, o projeto da Ouvidoria possuía cerca de 4 falhas reportadas ao final de cada *sprint* (falhas reportadas a cada implantação em produção). O código, contudo, apresentava um índice de manutenibilidade médio igual a 65.

Embora tenham sido feitas boas escolhas quanto às tecnologias e arquitetura, o sistema de ouvidoria também necessitava de um controle de qualidade mais rigoroso, visto que assim como o Portal, também passa por constantes evoluções. A figura 8 exibe a página inicial do Sistema de Ouvidoria do Distrito Federal.

Figura 8: Página inicial - Sistema de Ouvidoria do Distrito Federal

5.1.3 Sistema de Correição - SICOR

Interna à CGDF existe a SUCOR (Subcontroladoria de Correição Administrativa). A SUCOR é uma das áreas que alicerçam a atuação da CGDF e tem como finalidade prevenir e apurar irregularidades no âmbito da Administração Pública, por meio da instauração e condução de procedimentos correcionais. Adicionalmente, exerce o papel de coordenação e supervisão das atividades correcionais dos órgãos e entidades do Governo do Distrito Federal.

Para o acordo de resultados de 2017, foi estabelecida a criação de um sistema para automatizar algumas atividades do fluxo de trabalho da SUCOR. Assim sendo, iniciou-se o projeto de desenvolvimento do SICOR.

Para desenvolvimento do sistema, a equipe de tecnologia optou por adotar a linguagem de programação *Ruby* juntamente com o apoio do *framework Rails*. Quanto à arquitetura, o SICOR possui organização similar ao Portal, portando a camada de *frontend* utilizando o *framework AngularJS* e a camada de *backend* como uma API *REST*, desenvolvida em *Ruby*.

A seleção deste projeto denota o desejo compartilhado entre a alta administração de TI e a área de negócio em se obter um produto sendo construído com qualidade desde o início. Adicionalmente, as práticas propostas pela VBSE, integrantes do *framework* concebido neste trabalho, apresentam-se como essenciais para o êxito do projto. Desde o início o planejamento do projeto foi realizado com base nos itens mais críticos considerados pela área de negócio, tais como trilhas de auditorias, sigilo das informações e criptografia das informações trafegadas pela aplicação.

No capítulo 6 será possível contemplar os dados obtidos a partir do uso do *framework* de avaliação da qualidade de código no projeto de desenvolvimento do SICOR.

5.2 Projeto selecionado no Laboratório Fábrica de *Software*

5.2.1 Sistema de Perícia Médica

O Laboratório Fábrica de *Software* possui atualmente uma parceria com uma instituição privada que, por motivos de políticas de sigilo, não terá sua identidade revelada neste trabalho. Contudo, a mesma será tratada como Instituição X.

A Instituição X é uma empresa que promove processos seletivos e aplica diversos tipos de exames no Brasil. Determinados processos seletivos organizados pela empresa necessitam de uma etapa adicional para a realização da perícia médica nos candidatos classificados para as fases posteriores. Atualmente, todo este processo é manual, envolvendo o preenchimento de fichas e um exaustivo trabalho para passar todos estes dados para o meio digital.

Tendo em vista esta oportunidade de negócio, a Instituição X solicitou o desenvolvimento de um sistema para automatizar o registro e compilação dos dados inerentes à perícia médica ao Laboratório Fábrica de *Software*.

A solução proposta pelo Laboratório Fábrica de *Software* envolve a elaboração de duas aplicações que funcionam de forma conjunta. Há uma aplicação *web*, sendo desenvolvida em linguagem de programação *CSharp*, seguindo o estilo arquitetural MVC. Nesta aplicação, a área de negócio pode montar fichas personalizadas de perícia médica para os diversos editais que são publicados. Adicionalmente, a aplicação *web* recebe todos os dados enviados pela segunda aplicação, que é *mobile* e esta foi projetada para ser utilizada nos locais de realização da perícia médica. A aplicação *mobile* também está sendo desenvolvida em linguagem *CSharp* e também segue as diretrizes do estilo arquitetural MVC.

A equipe de desenvolvimento do sistema de perícia médica também relatou a necessidade que possuíam de implementar um controle mais rigoroso da qualidade. O índice de manutenibilidade médio obtido a partir da primeira coleta foi igual a 55 e o número médio de falhas reportados pela área de negócio ao final de cada *sprint* era igual a 5. Adicionalmente, a equipe também mencionou dificuldades na elaboração de testes unitários para a aplicação *mobile*. Assim, o projeto caracterizou-se como uma excelente oportunidade para aplicação do *framework* proposto por este trabalho.

5.3 Ferramentas para Desenvolvimento e Coleta de Dados

No contexto da CGDF, para o desenvolvimento do Portal da Transparência, utiliza-se a IDE (*Integrated Development Environment*) *Intellij*.

A IDE *Intellij* fornece suporte nativo para as tecnologias utilizadas no desenvolvimento do Portal. Adicionalmente, possui compatibilidade com diversos *plugins*, dentre eles, o *Metrics Reloaded*, o *plugin* adotado neste estudo para coleta das métricas de código fonte do projeto Portal.

Por fim, é válido destacar que a IDE *Intellij* já possui integração nativa com a biblioteca JaCoCo (*Java Code Coverage*), que fornece uma análise detalhada acerca da cobertura do código a partir da coloração apresentada no código fonte. As cores apresentadas pela biblioteca possuem significados específicos:

- **Linhas Verdes:** Indicam que a suíte de testes cobriu o referido trecho de código tanto pela abordagem de *statement coverage* quanto pela abordagem de *branch coverage*.
- **Linhas Amarelas:** Indicam que a suíte de testes cobriu parcialmente o referido trecho de código. Neste cenário, faz-se necessária a construção de mais um método de teste para exercitar o trecho faltante.
- **Linhas Vermelhas:** Indicam que a suíte de testes não foi capaz de exercitar o referido trecho de código. Faz-se necessária a construção de métodos de teste para fazê-lo.

As figuras 9 e 10 exibem, respectivamente, a tela de customização de métricas do *Metrics Reloaded Plugin* e um trecho de código analisado pela biblioteca JaCoCo após execução da suíte de testes.

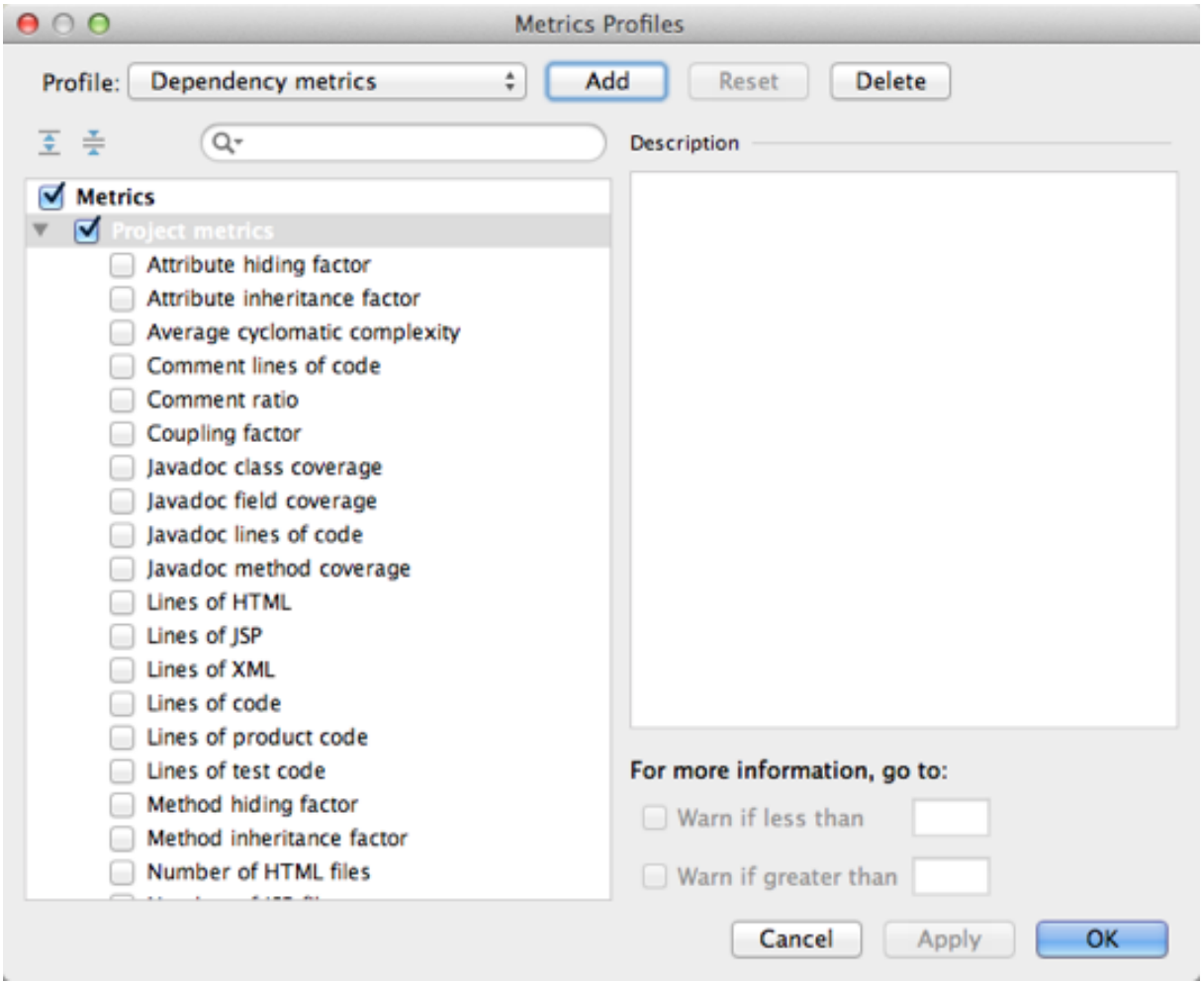

Figura 9: Tela de Customização de Métricas - *Metrics Reloaded Plugin*

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```

Figura 10: Análise de Cobertura de Código - Jacoco

Com relação ao Sistema de Ouvidoria do Distrito Federal e ao Sistema de Perícia Médica, pelo fato de serem sistemas desenvolvidos em linguagem *CSharp*, utiliza-se a IDE *Visual Studio*.

A IDE *Visual Studio*, em sua versão *Ultimate*, possui suporte para análise de cobertura de código e coleta de métricas.

De forma semelhante à biblioteca JaCoCo, a IDE *Visual Studio* exibe uma coloração específica para a análise de cobertura após execução da suíte de testes. Uma coloração azul claro é exibida nos trechos de código exercitados de forma efetiva pela suíte de testes (*statement coverage* e *branch coverage*). As linhas de código que apresentam uma coloração vermelha expressam insuficiência de métodos de teste na suíte para exercitá-las.

Por fim, com relação ao projeto SICOR, adotou-se o *Simplecov* como ferramenta de análise de cobertura e a ferramenta *Metric fu* para coleta de métricas de código fonte.

As figuras 11 e 12 exibem, respectivamente, uma análise de cobertura de código feita pela IDE *Visual Studio* após execução de uma suíte de testes e o resultado da coleta de métricas.

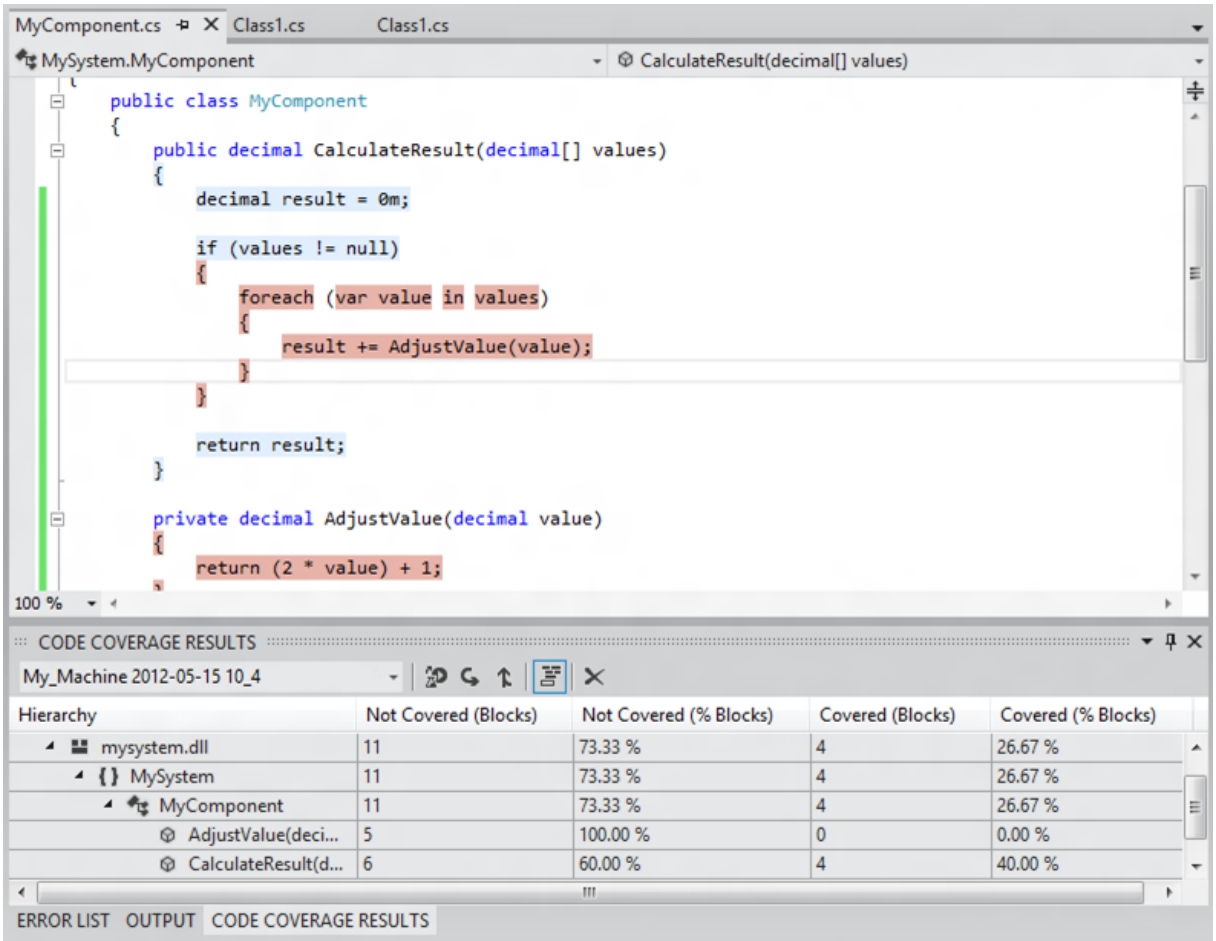

Figura 11: Análise de Cobertura de Código - *Visual Studio*

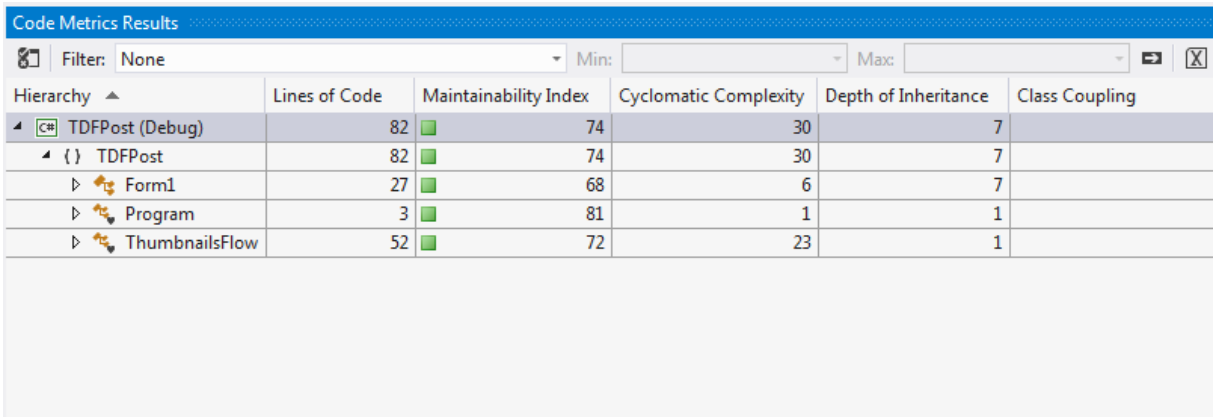

Figura 12: Coleta de Métricas - *Visual Studio*

5.4 Ferramenta para Gerenciamento e Inspeções de Código

Tanto no contexto da CGDF como também do Laboratório Fábrica de *Software*, utiliza-se a ferramenta de gerenciamento TFS (*Team Foundation Server*). A ferramenta já apresenta suporte para gerenciamento de projetos que adotam metodologias ágeis, especificamente o *Scrum*. A figura 13 exibe a página inicial da ferramenta.

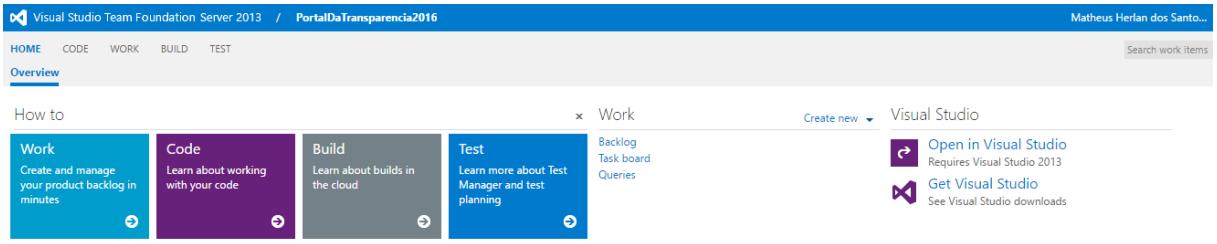

Figura 13: Página inicial - *Team Foundation Server*

É válido ressaltar que a ferramenta TFS possui suporte para a realização de *Pull Requests*, a forma encontrada neste estudo para que as inspeções fossem devidamente realizadas. O fluxo básico para realização das inspeções, tanto na CGDF como também no Laboratório Fábrica de *Software*, apresenta os seguintes passos:

1. O desenvolvedor, após finalizar a implementação da sua funcionalidade, submeterá um *Pull Request* da sua *branch* para a *branch approval*. Durante este processo, o desenvolvedor deverá delegar esta inspeção a qualquer outro desenvolvedor membro do projeto.

2. O desenvolvedor que for designado para efetuar a inspeção deverá aplicar os *checklists* propostos pelo *framework* concebido neste estudo e efetuar os devidos apontamentos para o desenvolvedor que submeteu o *Pull Request*.

3. Após correções e ajustes, caso necessário, o desenvolvedor que realizou a inspeção deverá efetuar a mesclagem do código na *branch approval*.

4. Ao final da *sprint*, todo o código da *branch approval* será mesclado na *branch* de homologação para entrega das funcionalidades desenvolvidas na *sprint* à área de negócio.

5. Após homologação da *release* entregue, será feita a contabilização do número de falhas inerentes às novas funcionalidades entregues, caso sejam notadas. Por fim, ocorrerá mesclagem do código na *branch* master.

5.5 Ferramenta para Coleta do Índice de Satisfação dos Desenvolvedores

Para coletar a percepção dos desenvolvedores com relação ao uso do *framework* proposto neste estudo, foi elaborado um formulário por meio da ferramenta *Google Forms*. Basicamente, foram colocadas as questões e alternativas, segundo escala *Likert*, já apresentados no capítulo sobre metodologia. A figura 14 exibe a página inicial do formulário eletrônico.

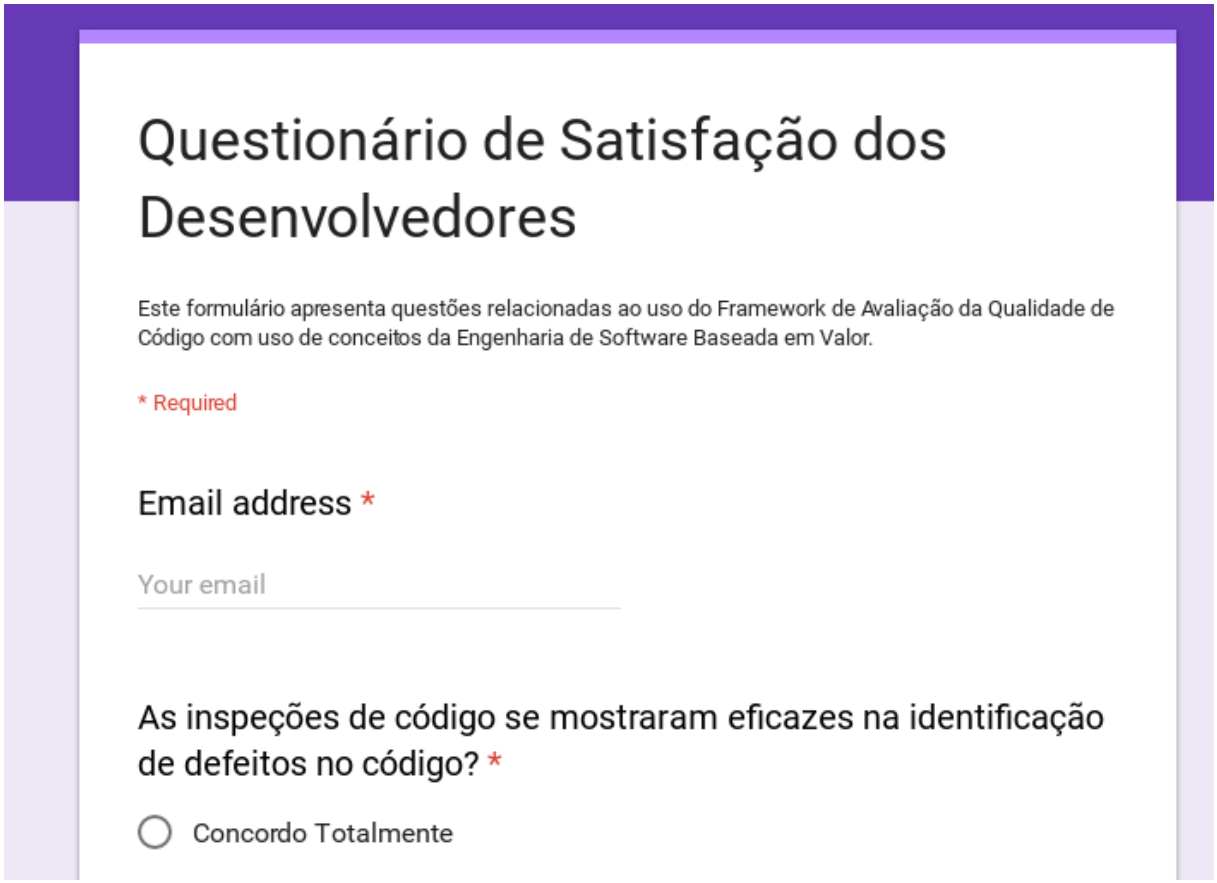

Figura 14: Formulário Eletrônico - Índice de Satisfação dos Desenvolvedores

5.6 Planejamento da Aplicação do *Framework*

Inicialmente, planejou-se um período de treinamento nas organizações. Na CGDF, a cultura de implementação de testes unitários e realização de inspeções ainda estava em seu estágio inicial. Portanto, foi preparado um treinamento contemplando modelos de implementação de testes unitários para todas as tecnologias utilizadas no desenvolvimento dos projetos selecionados. Adicionalmente, explicitou-se mais conceitos inerentes às inspeções para que os *checklists* propostos pelo *framework* aplicado neste estudo ficassem mais elucidativos.

No Laboratório Fábrica de *Software*, de forma semelhante, também foi promovido um treinamento inicial. Contudo, o foco deste treinamento esteve centralizado na implementação de testes unitários para aplicações *mobile*, uma das principais dificuldades relatadas pela equipe de desenvolvimento do projeto da perícia médica.

Como o *framework* utilizado neste estudo já se caracteriza como um processo alinhado às atividades e práticas do *Scrum*, os ciclos de pesquisa-ação, basicamente, são as *sprints* dos projetos de desenvolvimento. É válido notar que ao final de cada *sprint*, serão realizadas coletas de percepções de forma a aprimorar as atividades propostas pelo *framework*. A seguir, tem-se a figura 15, que exibe o planejamento inicial dos ciclos de coleta de dados.

Planejamento - Ciclos de Coleta de Dados	
Projeto - Portal da Transparência - CGDF	
Sprint	**Período**
Treinamento	16/01/2017 - 24/02/2017
1	06/03/2017 - 27/03/2017
2	27/03/2017 - 17/04/2017
3	17/04/2017 - 08/05/2017
Planejamento - Ciclos de Coleta de Dados	
Projeto - Sistema de Ouvidoria - CGDF	
Sprint	**Período**
Treinamento	16/01/2017 - 24/02/2017
1	13/03/2017 - 03/04/2017
2	03/04/2017 - 24/04/2017
Planejamento - Ciclos de Coleta de Dados	
Projeto - Sistema de Perícia Médica - Laboratório Fábrica de Software	
Sprint	**Período**
Treinamento	24/03/2017 - 06/04/2017
2	10/04/2017 -25/04/2017
3	27/04/2017 - 11/05/2017

Figura 15: Planejamento inicial - Ciclos de Coleta de Dados

Ao longo da execução dos ciclos de coleta de dados alguns ajustes foram necessários.

Na CGDF, por causa do surgimento de uma demanda emergencial, a execução do projeto do Sistema de Ouvidoria foi temporariamente suspensa e assim, não houve tempo hábil para coletar os dados da segunda *sprint* do mesmo.

O projeto do Portal, devido também à demanda emergencial da CGDF, teve sua execução suspensa temporariamente. As equipes de desenvolvimento destes projetos foram realocadas para atendimento da demanda. Adicionalmente, como o projeto de desenvolvimento do SICOR surgiu em tempo oportuno, resolveu-se adotar o mesmo como caso para aplicação do *framework* proposto neste trabalho.

Com relação ao Laboratório Fábrica de *Software* também foi necessário efetuar um ajuste no planejamento, visto que a equipe apresentou dificuldades em realizar as inspeções após treinamento, comprometendo o que havia sido planejado inicialmente. Assim, optou-se por fazer mais uma reunião para retratar como a coleta deveria ser feita e o planejamento passou a contemplar as *sprints* subsequentes.

A figura 16 exibe o planejamento final deste trabalho.

Planejamento - Ciclos de Coleta de Dados	
Projeto - Portal da Transparência - CGDF	
Sprint	**Período**
Treinamento	16/01/2017 - 24/02/2017
1	06/03/2017 - 27/03/2017
2	10/04/2017 - 04/05/2017
Projeto - SICOR (Sistema de Correição) - CGDF	
Sprint	**Período**
1	17/04/2017 - 09/05/2017
2	17/05/2017 - 06/06/2017
Projeto - Sistema de Ouvidoria - CGDF	
Sprint	**Período**
Treinamento	16/01/2017 - 24/02/2017
1	13/03/2017 - 03/04/2017
Projeto - Sistema de Perícia Médica - Laboratório Fábrica de Software	
Sprint	**Período**
Treinamento	24/03/2017 - 06/04/2017
1	27/04/2017 - 15/05/2017
2	16/05/2017 - 29/05/2017

Figura 16: Planejamento final - Ciclos de Coleta de Dados

5.7 Diagnóstico Inicial dos Projetos

Na CGDF, como nenhum projeto era adepto de uma cultura centralizada em verificação, antes da aplicação do *framework*, não havia implementação de testes unitários e também, realização de inspeções. Em muitas ocasiões, modificações efetuadas em um determinado componente do código fonte alterava o comportamento de outro componente.

Este aspecto era percebido posteriormente pelos desenvolvedores durante atividades de *debug*.

No Laboratório Fábrica de Software já ocorria implementação de testes unitários, porém, somente as classes do pacote *controller* possuíam testes implementados. A partir de uma rápida inspeção nos códigos de teste, identificou-se a necessidade de refatorar trechos de código repetidos, bem como implementar mais cenários de testes de forma que o código fonte dos componentes sob teste fosse concisamente exercitado.

5.8 Restrições existentes na Coleta de Dados

Com base nas tecnologias utilizadas para construção dos sistemas, ferramentas existentes para coleta de métricas e disponibilidade das equipes para executar as atividades de verificação, foram delimitados escopos de coletas.

No projeto do Portal, foi acordado realização de inspeções e implementação de testes unitários tanto para o código da camada de *frontend* quanto da camada *backend*. Com relação à análise do índice de manutenibilidade, somente a camada de *backend* foi utilizada. A camada de *frontend*, por se feita em *Java Script*, não possui um suporte conciso em termos de ferramentas para realizar a análise desta métrica.

Quanto ao projeto da Ouvidoria, foi acordado realização de inspeções e implementação de testes somente para a camada de *backend*. A equipe do projeto, por não possuir experiência em verificação, considerou melhor aplicar o *framework* inicialmente desta maneira. Contudo, a equipe do projeto pretende futuramente incluir a camada de *frontend* nas análises.

Por fim, no Laboratório Fábrica de Software, optou-se também por efetuar as inspeções e implementação de testes unitários somente na camada de *backend*.

É válido ressaltar que tanto no projeto da Ouvidoria quanto no projeto da Perícia Médica, foram acordadas inspeções rápidas do código da camada de *frontend* antes da mesclagem nas *branches* oficiais. Porém, seus resultados não foram considerados formalmente para os fins da análise efetuada neste trabalho.

6 Resultados da Aplicação do *Framework*

Neste capítulo são apresentados os resultados de cada *sprint* executada nos projetos selecionados para análise. Neste trabalho, cada *sprint* representa um ciclo de pesquisa-ação e, como etapas de um ciclo, tem-se o planejamento, a ação e a avaliação. O relato da etapa de planejamento detalha tudo o que foi concebido entre a equipe de pesquisa, equipe de TI da instituição e área de negócio do projeto. A etapa ação evidencia como as atividades planejadas na etapa anterior se desenvolveram. Por fim, na etapa de avaliação, os dados coletados são exibidos bem como uma breve análise. Será possível contemplar gráficos comparativos ao final para evidenciar a evolução dos ciclos em cada projeto.

6.1 Inicialização dos Ciclos - Treinamento

Como demonstrado no planejamento da execução dos ciclos de pesquisa-ação, no capítulo sobre Materiais e Métodos, foi realizado um treinamento em todas as instituições com o propósito de esclarecer conceitos e práticas propostas pelo *framework*. Nesse âmbito, foram evidenciados os aspectos inerentes à verificação de *software* e utilização de ferramentas para as equipes de TI e, por outro lado, procurou-se dialogar com as áreas de negócio para tratar sobre os aspectos formais de como os projetos seriam executados a partir daquele momento, considerando princípios da VBSE.

Para familiarizar as equipes de TI com as práticas de implementação de testes unitários e inspeção de código, foram organizadas dinâmicas denominadas *Coding Dojo*. Nesse tipo de dinâmica todas as pessoas atuam na construção de uma solução proposta pelo idealizador, alternando a posição reflexiva da platéia com as posições mais ativas dos pilotos. Assim, a construção sempre continua a partir do que os pilotos anteriores já desenvolveram.

As dinâmicas foram projetadas considerando as tecnologias e ferramentas utilizadas pelas equipes de TI. No caso da CGDF, para o projeto do Portal, considerou-se uma dinâmica de desenvolvimento de uma API *REST* utilizando o *framework Spring*. Adicionalmente, os participantes desenvolveram testes unitários e os demais desenvolvedores desempenhavam ações de inspeção. De forma semelhante, para o projeto de construção do Sistema de Ouvidoria foi organizada uma dinâmica para desenvolvimento de um simples exemplo com testes e inspeções. Para a equipe de desenvolvimento do Sistema de Correição não foi necessário promover outro treinamento, visto que os integrantes já haviam participado dos dojos anteriores.

No Laboratório Fábrica de *Software* foi feito um dojo que envolvia a criação de

uma aplicação *mobile* na plataforma *Android* juntamente com a implementação de testes unitários e inspeções.

Com relação às áreas de negócio, foi necessário enfatizar o motivo pelo qual a quantidade de *story points* passou a ser menor: aumento da qualidade das entregas. Para executar as atividades de verificação de *software*, as equipes de TI optaram, a priori, pela redução da carga de trabalho.

6.2 Ciclo 1

O principal objetivo do Ciclo 1 foi avaliar o primeiro uso do *framework* de avaliação da qualidade de código nos contextos selecionados e, a partir dos resultados obtidos, caso necessário, efetuar ajustes nas atividades constituintes do mesmo. A seções a seguir descrevem de forma detalhada os acontecimentos e dados coletados durante a execução da *sprint* 1 dos projetos selecionados.

6.2.1 Portal da Transparência

6.2.1.1 Planejamento

A partir de uma reunião com a área de negócio, tendo em vista o acordo de resultados com o governador do Distrito Federal, foi realizada uma priorização de itens do *backlog*. Dessa forma, as seguintes funcionalidades foram acordadas:

- Ajustes na consulta de Empresas Punidas
- Reformulação de filtros nos *dashboards* de dados sobre Servidores e Remuneração dos Servidores
- Ajustes na consulta de Licitações e Contratos

Antes do início da *Sprint* 1, os desenvolvedores configuraram o ambiente de testes, visto que o projeto do Portal não compreendia essas atividades anteriormente.

6.2.1.2 Ação

Com base nos itens do *backlog* priorizados, todas as tarefas foram criadas na ferramenta de gerenciamento TFS. A partir de então, as atividades corriqueiras do *Scrum* foram executadas em conjunto com as práticas propostas pelo *framework* concebido neste trabalho.

6.2.1.3 Avaliação

A *Sprint* 1 do projeto do portal foi executada com êxito. Todas as funcionalidades foram entregues. É válido ressaltar também que os testes unitários foram devidamente implementados para todos os componentes que sofreram alterações e também, as inspeções foram realizadas. Contudo, a equipe do projeto protelou as atividades de verificação, tornando a finalização da *sprint* mais onerosa.

As tabelas 2 e 3 exibem um resumo das métricas coletadas para as camadas de *frontend* e *backend*, respectivamente.

Tabela 2: Tabela Resumo - Métricas *Sprint* 1 - Portal da Transparência (*Frontend*)

Número de Defeitos	4
Taxa de Acertos por Linha de Código	3

Tabela 3: Tabela Resumo - Métricas *Sprint* 1 - Portal da Transparência (*Backend*)

Número de Defeitos	3
Taxa de Acertos por Linha de Código	2

A partir dos dados exibidos pelas tabelas acima, considerando somente os novos trechos de códigos produzidos e os módulos alterados, as inspeções foram capazes de identificar 4 defeitos na camada *frontend* e 3 defeitos na camada *backend*. De fato, por mais que os desenvolvedores da equipe estivessem implementando de forma concisa, ainda foi possível contemplar defeitos no código. Ao final da *sprint*, todos os defeitos foram corrigidos antes da mesclagem final.

Com relação à taxa de acertos por linha de código, a suíte de testes elaborada foi capaz de exercitar, em média, 3 vezes os trechos de código sob teste na camada *frontend* e 2 vezes os trechos de código sob teste na camada *backend*. Ao final da *sprint* foi possível contemplar um percentual total de cobertura igual a 15,43% na camada *frontend* e 22,10% na camada *backend*.

Com relação à camada *backend*, calculou-se o índice de manutenibilidade para todos os componentes alterados. A tabela 4 exibe estes dados, indicando a classe com o seu respectivo índice calculado.

Tabela 4: Índice de Manutenibilidade *Sprint* 1 - Portal da Transparência (*Backend*)

Empresa Punida - Modelo	95,12
Empresa Punida - *Controller*	94,99
Servidores - *Controller*	49,31
Remuneração - *Controller*	74,51
Empresa Punida Relatório - *Service*	84,82

Foi possível perceber um valor baixo do índice para a classe *ServidoresController* tendo em vista os valores que as demais classes atingiram. Contudo, é um valor aceitável de acordo com os parâmetros da *Microsoft* e indica a direção para futuros esforços em refatoração.

Com relação ao número de falhas identificadas pela área de negócio, nada foi reportado. Este aspecto indica que uma verificação concisa possui forte impacto na qualidade do produto entregue.

Por fim, tem-se o percentual obtido para as respostas relacionadas ao questionário de verificação da satisfação dos desenvolvedores ao utilizarem o *framework*. Foi possível perceber que a equipe de desenvolvimento do portal apresentou boa aceitação quanto ao uso do *framework*. Em linhas gerais, as inspeções e a forma de implementar testes unitários foram bem vistas pelos desenvolvedores. Ainda assim, as práticas propostas, segundo os desenvolvedores da equipe, se mostratam onerosas em contextos de equipes pequenas (até 3 desenvolvedores). A figura 17 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 1 - Portal da Transparência					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?	2				
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?	1	1			
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?		2			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?	1		1		
A manutenibilidade do código ficou melhor após as inspeções de código?		2			

Figura 17: Índice de Satisfação dos Desenvolvedores - *Sprint* 1 do Portal

6.2.2 Sistema de Ouvidoria

6.2.2.1 Planejamento

Assim como o projeto do Portal, o Sistema de Ouvidoria é um projeto que possui metas no acordo de resultados com o governador do Distrito Federal. Por se tratar de um sistema que recebe diretamente manifestações por parte dos cidadãos e por armazenar tantos dados diariamente, as seguintes funcionalidades foram acordadas:

- Reformulação do filtro de manifestações
- Disponibilização de um questionário de pesquisa de satisfação dos cidadãos quanto ao atendimento das manifestações

Como o projeto não compreendia práticas de verificação de *software*, a equipe de TI também teve que configurar o ambiente de testes antes do início da *sprint*.

6.2.2.2 Ação

As atividades padrão do *Scrum* e do *framework* aqui proposto foram desenvolvidas durante a *sprint*. Adicionalmente, todas as tarefas relacionadas às funcionalidades foram criadas no TFS para maior controle do andamento da iteração de desenvolvimento.

6.2.2.3 Avaliação

Assim como no projeto do portal, a primeira *sprint* do projeto do sistema de ouvidoria também foi executada com êxito. A equipe de desenvolvimento também protelou as atividades de verificação, alocando-as para o final da *sprint*.

A tabela 5 exibe um resumo das métricas coletadas para as camadas de *backend*.

Tabela 5: Tabela Resumo - Métricas *Sprint* 1 - Sistema de Ouvidoria (*Backend*)

Número de Defeitos	1
Taxa de Acertos por Linha de Código	3

A partir dos dados exibidos pela tabela, constata-se que o código foi bem implementado, tendo em vista o baixo número de defeitos identificados durante a inspeção. Adicionalmente, a suíte de testes foi capaz de exercitar, em média, 3 vezes o trecho de código sob teste. A cobertura final contemplada na camada de *backend* foi igual a 23,88%.

Com relação ao índice de manutenibilidade, os valores calculados são expressos na tabela 6.

Tabela 6: Índice de Manutenibilidade *Sprint* 1 - Sistema de Ouvidoria (*Backend*)

CoreAppService - OuvCidadania.AppService	61
Resposta - *Domain.Entity*	90
Resposta Questionário - *Domain.Entity*	90
Resposta Exception - *Domain.Exceptions*	98
IRespostaRepository - *Domain.Repository*	100
IRespostaQuestionarioService - *Domain.Service*	100
IRespostaService - *Domain.Service*	100
Resposta Questionario Service - *Domain.Service*	65
Resposta Service - *Domain.Service*	64

Considerando os parâmetros da *Microsoft*, todas as classes e interfaces apresentaram índices de manutenibilidade aceitáveis. Contudo, deve-se atentar para as classes na camada de serviço, que indicam necessidade de refatorações para que o índice de manutenibilidade possa aumentar.

Assim como no projeto do portal, não houve registro de falhas por parte da área de negócio. Todas as funcionalidades entregues ao final da *sprint* foram estabelecidas de forma consistente.

Por fim, com relação ao índice de satisfação dos desenvolvedores, também contemplou-se uma boa percepção acerca da utilização do *framework*. A equipe também considerou as práticas propostas pelo *framework* onerosas para contextos de equipes pequenas e sugeriu que fosse feita uma adequação das práticas para equipes pequenas. A figura 18 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 1 - Sistema de Ouvidoria					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?	1	1			
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?	1	1			
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?	1	1			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?			2		
A manutenibilidade do código ficou melhor após as inspeções de código?	1	1			

Figura 18: Índice de Satisfação dos Desenvolvedores - *Sprint* 1 do Sistema de Ouvidoria

6.2.3 SICOR - Sistema de Correição

6.2.3.1 Planejamento

Pelo fato de o SICOR ser um novo projeto na CGDF, a equipe de TI realizou um diálogo prévio com a área de negócio para obter uma visão geral acerca de todas as funcionalidades que o sistema deveria possuir até a data da primeira entrega prevista no acordo de resultados com o governador.

Foi possível perceber a necessidade de agrupar as funcionalidades do sistema em 4 *features* principais, sendo:

- Módulo Administrativo
- Módulo Correcional
- Módulo Disciplinar
- Módulo de Responsabilização

Considerando os aspectos da VBSE, durante as conversas com a área de negócio, percebeu-se que todas as informações presentes no sistema deveriam ser tratadas como sigilosas. Adicionalmente, a área de negócio solicitou uma trilha de auditoria complexa para a aplicação, sendo que todos os eventos ocorridos deveriam ser registrados e sempre vinculados ao usuário responsável pelo mesmo.

Dessa forma, a área de negócio priorizou o módulo adminstrativo, caracterizando-o como crítico e fundamental para o sistema. Adicionalmente, a equipe de TI ponderou que este módulo seria fundamental para inicializar a construção da arquitetura do sistema, visto que todos os mecanismos de segurança e registro de eventos deveriam ser projetados e implementados logo no início.

Para a primeira *sprint* do projeto, as funcionalidades acordadas foram:

- Criação da arquitetura de registro de eventos (Logs)
- Cadastro de Usuários
- Cadastro de Unidades
- Login
- Cadastro de Perfis

Após priorização, a equipe de desenvolvimento iniciou as configurações de ambiente, banco de dados etc. O mais interessante é que a reestruturação efetuada nos outros

projetos contribuiu de forma significativa para o início da contrução do SICOR. Além disso, a equipe de desenvolvimento já havia participado previamente das iterações de desenvolvimento de outros projetos e, portanto, já estava familiarizada com as práticas propostas pelo *framework* de avaliação da qualidade de código.

6.2.3.2 Ação

Antes que a *sprint* fosse iniciada, todas as histórias foram registradas no TFS. Adicionalmente, todas as tarefas vinculadas também foram cadastradas. Ao final da execução da *sprint*, somente uma funcionalidade não foi entregue, sendo o cadastro de unidades.

6.2.3.3 Avaliação

Apesar de uma funcionalidade não ter sido entregue, a primeira *sprint* do SICOR foi considerada um sucesso. As tabelas 7 e 8 exibem um resumo das métricas coletadas para as camadas de *frontend* e *backend*, respectivamente.

Tabela 7: Tabela Resumo - Métricas *Sprint* 1 - SICOR (*Frontend*)

Número de Defeitos	1
Taxa de Acertos por Linha de Código	1

Tabela 8: Tabela Resumo - Métricas *Sprint* 1 - SICOR (*Backend*)

Número de Defeitos	1
Taxa de Acertos por Linha de Código	2,46

A partir dos dados exibidos nas tabelas acima é possível perceber uma baixa quantidade de defeitos, que evidencia uma maior precisão por parte dos desenvolvedores nos momentos de implementação. Adicionalmente, a suíte de testes unitários implementada foi capaz de exercitar, em média, 1 vez o trecho de código sob teste na camada de *frontend* e 2,46 vezes o trecho de código sob teste na camada de *backend*. Ao final da *sprint*, percebeu-se um percentual total de cobetura igual a 100% na camada de *frontend* e 95% na camada de *backend*. É válido ressaltar que não foi reportado nenhuma percepção de falha do sistema por parte da área de negócio durante a homologação.

Com relação ao quesito manutenibilidade e complexidade do código, como mencionado no capítulo sobre metodologia, adotou-se a métrica Flog (devido às particularidades da linguagem Ruby e do *framework Rails*). A seguir, tem-se a tabela 9, que exibe a pontuação total da métrica para as principais classes implementadas.

Tabela 9: Flog *Sprint* 1 - SICOR (*Backend*)

Auditoria (Log)	14
Application - Controller	12
Usuário - *Controller*	38
Usuário - Modelo	11,6
Órgão - *Controller*	3,2
Auditoria Base - Módulo Log	3
Handler - Módulo Usuário	2,3

Embora a trilha de auditoria realmente tenha sido considerada uma funcionalidade crítica do sistema, as escolhas de tecnologias se caracterizaram como apropriadas para o contexto, visto que o processamento é dividido entre o cliente e o servidor (API *REST* com cliente *AngularJS*). Esse aspecto favoreceu a diminuição da complexidade do código na camada de *backend*.

Por fim, é válido ressaltar que embora os desenvolvedores do projeto já tivessem familiaridade com o *framework* de avaliação da qualidade de código, ainda consideraram as práticas onerosas para o processo de desenvolvimento em contextos de equipes pequenas. Contudo, afirmaram que se a equipe possuir disciplina no seu fluxo de atividades, é possível conciliar o alto número de atividades com os prazos. A figura 19 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 1 - Sistema de Correição					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?	2				
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?		2			
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?		2			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?			2		
A manutenibilidade do código ficou melhor após as inspeções de código?		2			

Figura 19: Índice de Satisfação dos Desenvolvedores - *Sprint* 1 do SICOR

6.2.4 Sistema de Perícia Médica

6.2.4.1 Planejamento

Durante os diálogos realizados entre a equipe de desenvolvimento e a área de negócio, o projeto de desenvolvimento do Sistema de Perícia Médica esteve voltado, durante a primeira *sprint* avaliada por este trabalho, para aperfeiçoamentos e correções de *bugs* encontrados nas funcionalidades desenvolvidas nas iterações precedentes.

Basicamente, os seguintes itens foram acordados com a área de negócio:

- Melhorias no *layout* da Aplicação *Mobile*
- Aperfeiçoamentos no menu de operações disponíveis na Aplicação *Mobile*
- Correção de bugs no cadastro de colaboradores
- Melhorias no cadastro de tablets na Aplicação *Web*

6.2.4.2 Ação

Ao final da priorização, a equipe de desenvolvimento registrou as descrições dos itens a serem implementados bem como suas respectivas tarefas na ferramenta TFS. A *sprint* contemplou atrasos, dessa forma, as atividades de verificação foram proteladas para o final da mesma, assim como as mesclagens de código considerando o fluxo e política de *branches*.

6.2.4.3 Avaliação

Embora atrasos tenham sido contemplados, todas os itens tiveram sua implementação finalizada. A suíte de testes unitários implementada não contemplou todos os cenários mais básicos considerando a estrutura do código das classes. Adicionalmente, a equipe apresentou muita dificuldade para implementar testes de *Fragments* (um dos elementos constituintes da arquitetura de uma aplicação para plataforma *Android*) e assim, somente as classes da camada *controller* foram finalizadas com suítes de testes consistentes. Quanto à aplicação *Web*, as implementações ocorreram sem registro de dificuldades.

Considerando o código das aplicações, após as inspeções, foi contemplado um total de 5 defeitos. Pelo fato de ter ocorrido atrasos na execução da *sprint*, os defeitos foram corrigidos após finalização da mesma e antes de ocorrer a entrega formal dos resultados para a área de negócio. Quanto a suíte de testes, cada método de teste foi capaz de exercitar somente 1 vez o trecho de código sob teste. Adicionalmente, a inexistência de uma ferramenta de análise de cobertura prejudicou a percepção de mais cenários de teste por parte da equipe de desenvolvimento.

Embora a suíte de testes não tenha sido implementada da melhor forma possível, a área de negócio não reportou nenhuma falha quanto ao que havia sido implementado.

Por fim, coletou-se o índice de manutenibilidade das classes alteradas durante a execução da *sprint*. A tabela 10 exibe os resultados da coleta.

Tabela 10: Índice de Manutenibilidade *Sprint* 1 - Sistema de Perícia Médica (*Backend*)

AlterarFuncaoFragment - *Mobile*	59
MenuCoordenadorFragment - *Mobile*	64
ColaboradorController - *Mobile*	73
ColaboradorDAO - *Mobile*	69
CadastrarColaboradorFragment - *Mobile*	53
MainActivity - *Mobile*	73
ViewUtils - *Mobile*	59
ListarEquipeContratanteFragment - *Mobile*	63
TabletController - *Web*	74
SessaoController - *Web*	65
LiberaQRCodeController - *Web*	72
RegistrarTabletEvento - *Web Service*	71

É possível perceber com os dados da tabela acima que as classes apresentaram índice de manutenibilidade aceitável dentro dos parâmetros da *Microsoft*. Como mencionado anteriormente, mesmo que a suíte de testes não tenha contemplado todos os cenários básicos possíveis, as inspeções auxiliaram na identificação de defeitos e na visualização de formas melhores de se organizar o código, possibilitando altos índices de manutenibilidade e número reduzido de falhas.

Com relação ao uso do *framework*, a equipe do Laboratório Fábrica de *Software* apresentou menos receptividade do que as equipes da CGDF. Contudo, ainda assim, ressaltou que as inspeções auxiliaram significativamente na melhoria do índice de manutenibilidade do código. A figura 20 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 1 - Sistema de Perícia Médica					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?			2		
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?			2		
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?		2			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?		2			
A manutenibilidade do código ficou melhor após as inspeções de código?	2				

Figura 20: Índice de Satisfação dos Desenvolvedores - *Sprint* 1 do Sistema de Perícia Médica

6.3 Ciclo 2

O principal objetivo do Ciclo 2 foi efetuar os ajustes necessários na execução das atividades propostas pelo *framework* de avaliação da qualidade de código, bem como fornecer uma orientação quanto às dúvidas remanescentes por parte das equipes de desenvolvimento das instituições selecionadas para este trabalho.

Com base no relato do Ciclo 1, foi possível perceber que todas as equipes mencionaram que as práticas propostas pelo *framework* se mostraram parcialmente onerosas ao processo de desenvolvimento em âmbito de equipes pequenas. Assim, procurou-se obter por parte dos desenvolvedores sugestões de melhoria para o fluxo proposto pelo *framework*. Chegou-se a conclusão de que não poderia ocorrer exclusão de atividades mesmo para contextos menores, visto que o diferencial do *framework* é justamente o agrupamento das melhores práticas relativas à implementação de testes unitários e realização de inspeções de código, bem como a incorporação dos conceitos da VBSE, possibilitando com que o projeto seja executado de forma a atingir sua missão.

As demais atividades do *framework* apenas refletem o que já é proposto pelo *Scrum*. Quanto à primeira atividade, relacionada à VBSE (Elicitação das Propostas de Valor), não percebeu-se nenhuma carga extra de trabalho. Inclusive, as áreas de negócio passaram a fornecer *feedbacks* que expressavam maior satisfação quanto às atividades desempenhadas pelas equipes de desenvolvimento.

Considerando os fatos citados acima, as próprias equipes de desenvolvimento reconheceram que protelavam as atividades de verificação e que, pelo fato de não terem a cultura de realizar tais atividades, obtiveram certa dificuldade. Dessa maneira, nenhum ajuste foi efetuado no *framework* e as equipes procuraram realizar as atividades com o mínimo de protelação durante o Ciclo 2.

6.3.1 Portal da Transparência

6.3.1.1 Planejamento

Em reunião com a área de negócio, considerando o acordo de resultados com o governador, as seguintes funcionalidades foram acordadas:

- Reformulação da Consulta de Empresas Punidas (Compatibilização com os dados da CGU - Controladoria Geral da União)
- Inclusão da Consulta de Órgãos Deliberativos
- Aperfeiçoamentos na Consulta Dinâmica de Despesas

Antes do início da *sprint*, a equipe não precisou dedicar tempo à configuração de ambiente de testes, visto que tudo foi feito durante a *sprint* 1. Adicionalmente, a equipe de desenvolvimento acordou que nenhuma classe deveria ficar com índice de manutenibilidade inferior à 45 após finalização das implementações.

6.3.1.2 Ação

Com base nos itens do *backlog* priorizados, todas as tarefas foram criadas na ferramenta de gerenciamento TFS. A partir de então a equipe de desenvolvimento passou a fornecer maior atenção para as atividades de verificação e sempre que uma funcionalidade era finalizada, imediatamente após ocorria a inspeção da mesma. Dessa forma, a *sprint* foi executada com êxito dentro do prazo estabelecido.

6.3.1.3 Avaliação

A *Sprint* 2 do projeto do portal, como mencionado na seção anterior, foi executada com êxito. Todas as funcionalidades foram entregues. É válido ressaltar também que os testes unitários foram devidamente implementados para todos os componentes que sofreram alterações e também, as inspeções foram realizadas.

As tabelas 11 e 12 exibem um resumo das métricas coletadas para as camadas de *frontend* e *backend*, respectivamente.

Tabela 11: Tabela Resumo - Métricas *Sprint* 2 - Portal da Transparência (*Frontend*)

Número de Defeitos	1
Taxa de Acertos por Linha de Código	4

Tabela 12: Tabela Resumo - Métricas *Sprint* 2 - Portal da Transparência (*Backend*)

Número de Defeitos	2
Taxa de Acertos por Linha de Código	2

A partir dos dados exibidos pelas tabelas acima, considerando somente os novos trechos de códigos produzidos e os módulos alterados, as inspeções foram capazes de identificar 1 defeito na camada *frontend* e 2 defeitos na camada *backend*. Ao final da *sprint*, todos os defeitos foram corrigidos antes da mesclagem final.

Quanto à taxa de acertos por linha de código, a suíte de testes elaborada foi capaz de exercitar, em média, 4 vezes os trechos de código sob teste na camada *frontend* e 2 vezes os trechos de código sob teste na camada *backend*. Ao final da *sprint* foi possível contemplar um percentual total de cobertura igual a 23,00% na camada *frontend* e 27,10% na camada *backend*.

Analisando de forma rápida os percentuais de cobertura, percebe-se que o aumento foi pouco expressivo. Contudo, a suíte de testes elaborada caracterizou-se como consistente ao final da *sprint* 2, sendo que todos os cenários mais básicos para os módulos sob teste foram implementados.

Com relação à camada *backend*, calculou-se o índice de manutenibilidade para todos os componentes alterados. A tabela 13 exibe estes dados, indicando a classe com o seu respectivo índice calculado.

Tabela 13: Índice de Manutenibilidade *Sprint* 2 - Portal da Transparência (*Backend*)

Empresa Punida - Modelo	77,32
Empresa Punida - *Controller*	85,20
Empresa Punida - *Service*	78,38
Empresa Punida Relatório - *Service*	77,45
Órgãos Deliberativos - Modelo	80,46
Prestando Contas - *Controller*	56,71
Prestando Contas - *Service*	45,66
Prestando Contas Relatório - *Controller*	84,72
Prestando Contas Relatório - *Service*	77,17
Consulta Dinâmica Despesa - Modelo	68,62
Despesa - *Controller*	49,71

Foi possível perceber índices de manutenibilidade dentro dos padrões estabelecidos pela *Microsoft* e também, dentro da meta estabelecida pela equipe de desenvolvimento antes do início da *sprint* 2. Algumas classes tiveram uma redução no índice de manutenibilidade pelo fato de que código foi acrescentado. Contudo, essa redução não se caracterizou como impactante para a qualidade em um primeiro momento. O que deve ser observado é a evolução futura desse quadro. Se o índice de manutenibilidade de uma determinada classe diminuir a cada *sprint*, deverá ser feita uma refatoração.

Com relação ao número de falhas identificadas pela área de negócio, nada foi reportado. Este aspecto indica que uma verificação concisa possui forte impacto na qualidade do produto entregue.

Por fim, tem-se as respostas relacionadas ao questionário de verificação da satisfação dos desenvolvedores ao utilizarem o *framework*. Percebeu-se maior aceitação e entendimento da importância das práticas da verificação de *software* para obtenção de um produto com qualidade. A figura 21 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 2 - Portal da Transparência					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?	2				
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?	1	1			
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?		2			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?	2				
A manutenibilidade do código ficou melhor após as inspeções de código?	2				

Figura 21: Índice de Satisfação dos Desenvolvedores - *Sprint* 2 do Portal

6.3.2 SICOR - Sistema de Correição

6.3.2.1 Planejamento

Em conversa com a área de negócio, as seguintes funcionalidades foram acordadas:

- Inclusão de Documento Originário
- Consulta de Documentos
- Exclusão de Documentos
- Alteração de Documentos
- Cadastro de Unidades - Pendência da *Sprint* 1

As funcionalidades descritas acima caracterizam-se como o núcleo do Módulo Correcional, visto que todas as denúncias que chegam à SUCOR são por meio de documentos originários. Adicionalmente, a primeira *sprint* favoreceu a escolha destes itens, pois toda a estrutura já estava implementada.

6.3.2.2 Ação

Após priorização junto a área de negócio, a equipe de desenvolvimento registrou todas as tarefas relativas à implementação das funcionalidades no TFS. Adicionalmente, a equipe procurou estabelecer mais prioridades quanto à implementação de testes unitários. Pelo fato de terem sido alocados operações elementares, somente os testes para os módulos mais críticos foram desenvolvidos. Assim, a *sprint* foi executada com êxito, entregando tudo que foi acordado.

6.3.2.3 Avaliação

Como mencionado na seção anterior, a *sprint* 2 do projeto SICOR foi executada com êxito. As tabelas 14 e 15 exibem um resumo das métricas coletadas para as camadas de *frontend* e *backend*, respectivamente.

Tabela 14: Tabela Resumo - Métricas *Sprint* 2 - SICOR (*Frontend*)

Número de Defeitos	0
Taxa de Acertos por Linha de Código	2

Tabela 15: Tabela Resumo - Métricas *Sprint* 2 - SICOR (*Backend*)

Número de Defeitos	0
Taxa de Acertos por Linha de Código	2,76

A partir dos dados exibidos nas tabelas acima é possível perceber uma baixa quantidade de defeitos, que evidencia uma maior precisão por parte dos desenvolvedores nos momentos de implementação. Adicionalmente, a suíte de testes unitários implementada foi capaz de exercitar, em média, 2 vezes o trecho de código sob teste na camada de *frontend* e 2,76 vezes o trecho de código sob teste na camada de *backend*. Ao final da *sprint*, percebeu-se um percentual total de cobetura igual a 73,86% na camada de *frontend* e 60% na camada de *backend*. É válido ressaltar que não foi reportado nenhuma percepção de falha do sistema por parte da área de negócio durante a homologação.

Embora o percentual total de cobertura tenha diminuído, percebe-se que a suíte de testes ainda caracteriza-se como consistente, visto que o número de acertos por linha de código aumentou. Como mencionado anteriormente, a redução do percentual deve-se ao aumento de linhas de código. De forma geral, à medida em que o projeto cresce, mais difícil é manter o percentual de 100% de cobertura. Outro aspecto é que uma cobertura integral traz maior custo de desenvolvimento. Assim, é necessário ser mais seletivo quando se implementa testes unitários.

Com relação ao quesito manutenibilidade e complexidade do código, coletou-se a métrica Flog. A seguir, tem-se a tabela 16, que exibe a pontuação total da métrica para as principais classes implementadas.

Tabela 16: Flog *Sprint* 2 - SICOR (*Backend*)

Documento - *Controller*	57,3
Documento - Modelo	44,1
Criar Documento - *Service*	17,3
Atualizar Documento - *Service*	7,1
Pesquisar Documento - *Service*	29,6

Com base nos dados apresentados pela tabela acima, percebe-se baixas pontuações de complexidade. Mesmo nas classes que apresentaram pontuações totais acima de 40, a média para os métodos da classe foi inferior à 8.

Por fim, tem-se as respostas relacionadas ao questionário de verificação da satisfação dos desenvolvedores ao utilizarem o *framework*. Assim como no projeto do Portal, percebeu-se maior aceitação e entendimento da importância das práticas da verificação de *software* para obtenção de um produto com qualidade. A figura 22 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 2 - Sistema de Correição					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?	2				
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?	2				
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?	1	1			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?			2		
A manutenibilidade do código ficou melhor após as inspeções de código?	2				

Figura 22: Índice de Satisfação dos Desenvolvedores - *Sprint* 2 do SICOR

6.3.3 Sistema de Perícia Médica

6.3.3.1 Planejamento

A segunda iteração de desenvolvimento do projeto de construção do Sistema de Perícia Médica teve enfoque na sincronização de dados entre os *tablets* portadores da aplicação. Esse mecanismo é fundamental para os testes de aptidão promovidos pela Instituição X, parceira e patrocinadora do Laboratório Fábrica de *Software*.

As funcionalidades discutidas e acordadas entre a área de negócio e a área de desenvolvimento foram:

- Criptografia do banco de dados da aplicação *mobile*
- Registro de log de operações efetuadas na aplicação *mobile*
- Transmissão de informações entre *tablets*
- Mesclagem de dados a partir da compração do horário da alteração

6.3.3.2 Ação

Assim como em todas as iterações relatadas até a presente seção, a equipe de desenvolvimento registrou as atividades na ferramenta TFS para gerenciamento e controle do andamento da *sprint*. Ao final, somente a funcionalidade de mesclagem de dados obteve testes unitários implementados. Nas demais, os módulos foram implementados, mas não tiveram testes unitários. As inspeções foram efetuadas em todos os módulos.

6.3.3.3 Avaliação

A partir das inspeções efetuadas pela equipe, foi identificado um total de 4 defeitos no código. Os defeitos foram corrigidos somente após o prazo de encerramenta da *sprint*. Da mesma forma como na *sprint* 1, a suíte de testes não se caracterizou como consistente a partir de uma perspectiva geral. Muitos testes não foram implementados e os módulos que obtiveram implementação de testes unitários careciam de mais cenários.

De forma semelhante à *sprint* 1, a área de negócio não contemplou falhas durante a homologação dos itens desenvolvidos.

Por fim, coletou-se o índice de manutenibilidade das classes alteradas durante a execução da *sprint*. A tabela 17 exibe os resultados da coleta.

Tabela 17: Índice de Manutenibilidade *Sprint* 2 - Sistema de Perícia Médica (*Backend*)

ColaboradorDAO - *Mobile*	66
CandidatoDAO - *Mobile*	70
EquipeContratanteDAO - *Mobile*	66
MembroEquipeContratanteDAO - *Mobile*	67
InscritoDAO - *Mobile*	67
ResultadoDAO - *Mobile*	67
MiniBanco - *Mobile*	56

A partir dos dados apresentados pela tabela acima, é possível concluir que as inspeções auxiliaram na estabilidade do índice de manutenibilidade e favoreceu o aumento deste para algumas classes tendo em vista o índice de manutenibilidade médio relatado na seção de diagnóstico do projeto.

Com relação ao uso do *framework*, a equipe do Laboratório Fábrica de *Software* apresentou maior receptividade em relação à *sprint* anterior. A figura 23 exibe a quantidade de respostas para cada questão do questionário de verificação da satisfação.

Sprint 2 - Sistema de Perícia Médica					
Questão	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
As inspeções de código se mostraram eficazes na identificação de defeitos no código?		2			
Os testes unitários implementados de acordo com o guia, de fato, exercitam o código de maneira eficiente?		2			
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?		2			
O número de falhas percebidos pelo usuário passou a ser menor a cada release entregue?		2			
A manutenibilidade do código ficou melhor após as inspeções de código?	2				

Figura 23: Índice de Satisfação dos Desenvolvedores - *Sprint* 2 do Sistema de Perícia Médica

6.4 Comparativo dos Ciclos

A partir da execução de dois ciclos de pesquisa-ação foi possível perceber a mudança de opinião que as equipes de desenvolvimento apresentaram quanto à utilização do *framework* de avaliação da qualidade de código.

No caso do projeto do Portal, após execução do segundo ciclo de pesquisa, foi possível perceber que 100% dos desenvolvedores passaram a concordar totalmente com o aspecto de que a manutenibilidade do código passou a melhorar após realização de inspeções (questão 5). Adicionalmente, 100% dos desenvolvedores passaram a concordar totalmente quanto à diminuição do número de falhas percebidas pelos usuários (questão 4).

A figura 24 exibe dois gráficos comparativos, *sprint* 1 e *sprint* 2, que evidenciam as percepções dos desenvolvedores do Portal quanto ao uso do *framework*. Apenas para fins de esclarecimento, as figuras exibem gráficos das duas *sprints*, sendo os dados da *sprint* 1 à esquerda e os dados da *sprint* 2 à direita. As barras exibem a quantidade de respostas percebidas para cada questão (Q1, Q2, Q3, Q4 e Q5), bem como a classificação da resposta quanto à escala *Likert*, sendo as abreviações:

- CT - Concordo Totalmente
- CP - Concordo Parcialmente
- N - Neutro
- DP - Discordo Parcialmente
- DT - Discordo Totalmente

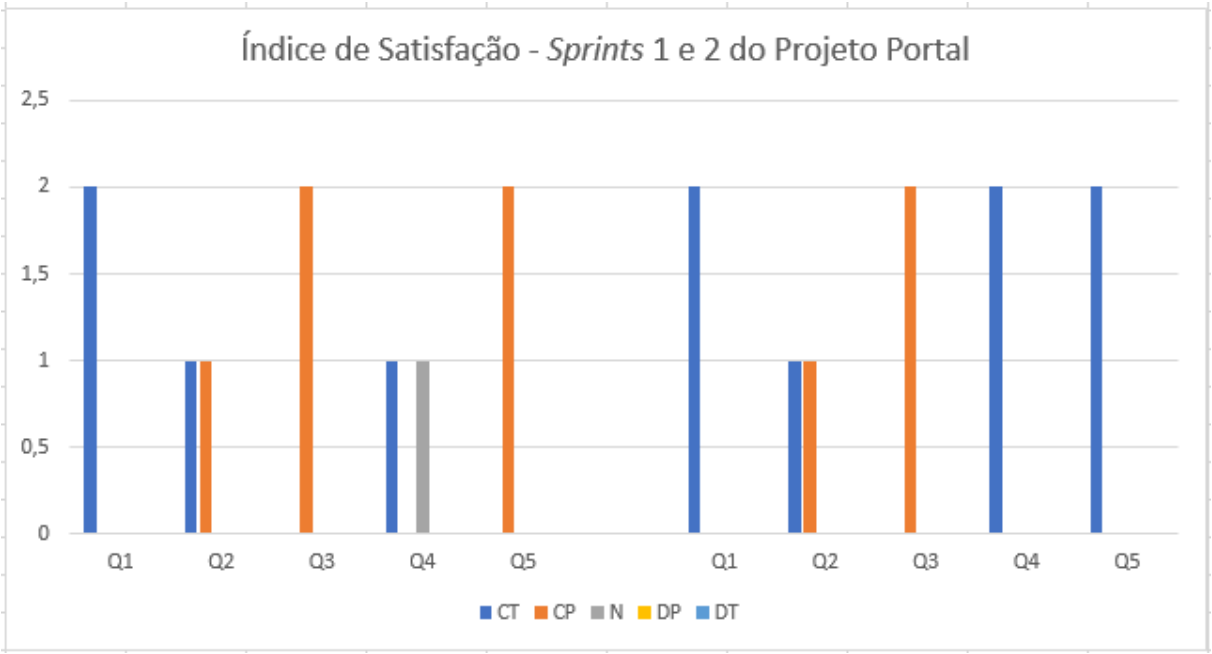

Figura 24: Índice de Satisfação dos Desenvolvedores - Portal da Transparência - Evolução dos Ciclos

Com relação ao projeto SICOR, a equipe de desenvolvimento apresentou percepção similar à equipe do portal quanto ao aspecto de que as inspeções se mostraram eficientes na identificação de defeitos no código (questão 1), ou seja, 100% dos desenvolvedores das duas equipes concordaram totalmente quanto a este aspecto em ambos os ciclos de pesquisa. Adicionalmente, ao final do ciclo 2, 100% da equipe de desenvolvimento do SICOR passou a concordar totalmente quanto ao aspecto de que os testes unitários implementados de acordo com o guia proposto pelo *framework* exercitam o código de maneira eficiente (questão 2).

A figura 25 exibe dois gráficos comparativos, *sprint* 1 e *sprint* 2, que evidenciam as percepções dos desenvolvedores do SICOR quanto ao uso do *framework*.

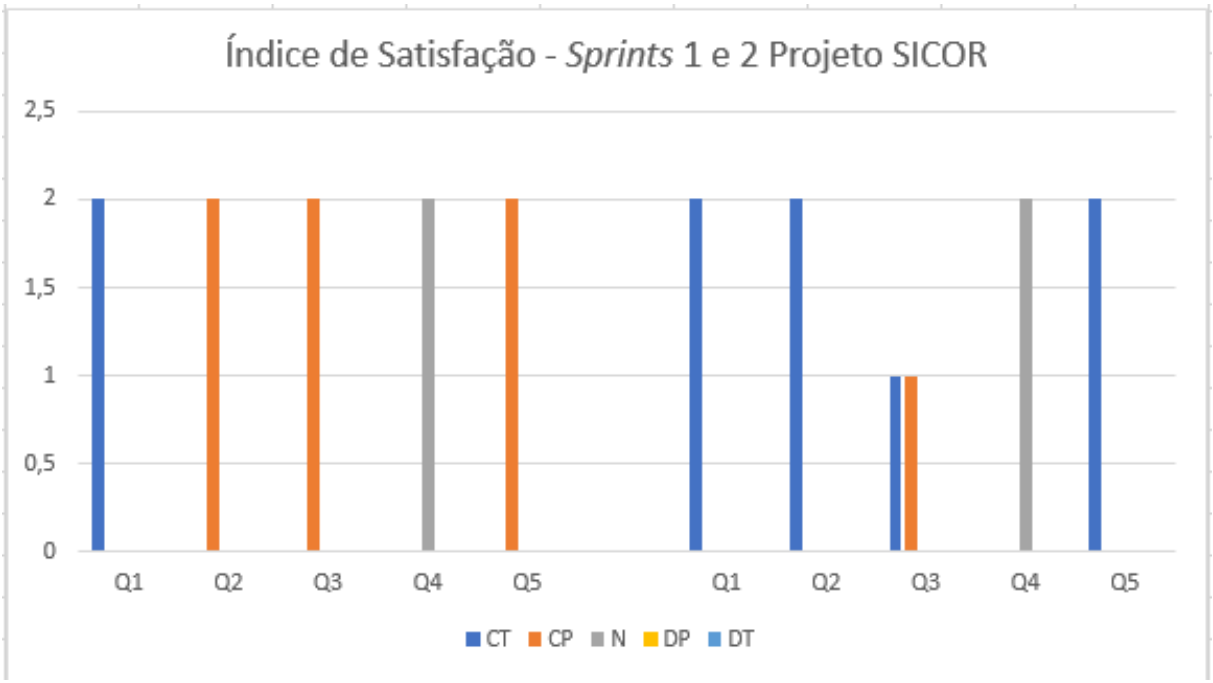

Figura 25: Índice de Satisfação dos Desenvolvedores - SICOR - Evolução dos Ciclos

No caso do projeto Sistema de Perícia Médica, foi notória a evolução da percepção da equipe de desenvolvimento quanto à importância das práticas de verificação de *software*. Para exemplificar essa afirmação, 100% dos desenvolvedores eram neutros, no ciclo 1, quanto à eficácia das inspeções na identificação de defeitos e da utilidade dos testes unitários (questões 1 e 2). No ciclo 2, 100% dos desenvolvedores passaram a concordar parcialmente quanto a estes aspecto. Contudo, desde o início, 100% da equipe concordava totalmente quanto a melhoria da manutenibilidade após realização de inspeções (questão 5).

A figura 26 exibe dois gráficos comparativos, *sprint* 1 e *sprint* 2, que evidenciam as percepções dos desenvolvedores do Sistema de Perícia Médica quanto ao uso do *framework*.

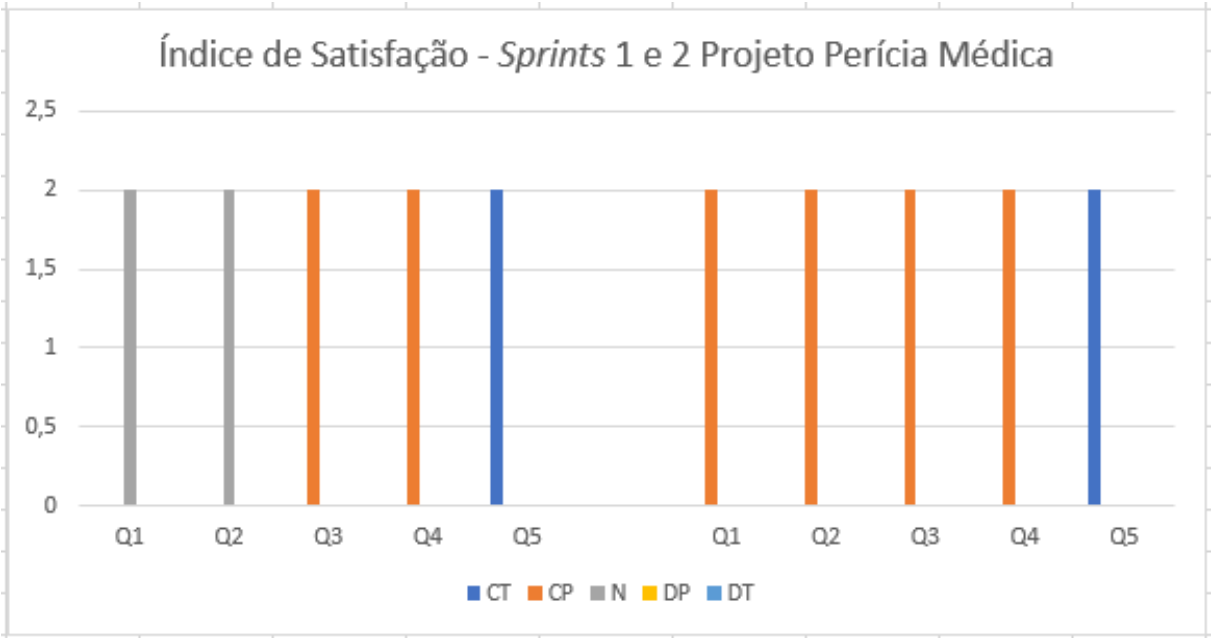

Figura 26: Índice de Satisfação dos Desenvolvedores - Sistema de Perícia Médica - Evolução dos Ciclos

Por fim, tem-se o gráfico geral para o projeto do Sistema de Ouvidoria. Neste caso, infelizmente não ocorreu a execução do segundo ciclo, mas é possível contemplar a receptividade dos desenvolvedores do mesmo quanto ao uso do *framework* em uma iteração de desenvolvimento. A figura 27 exibe o cenário geral.

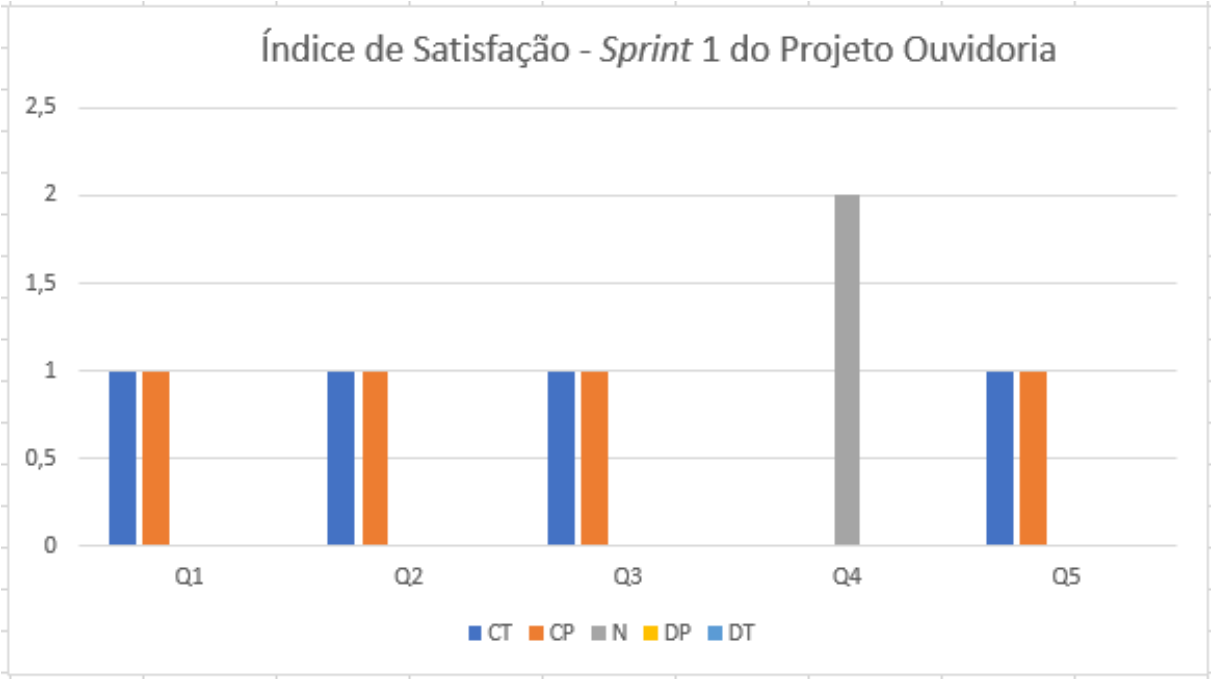

Figura 27: Índice de Satisfação dos Desenvolvedores - Sistema de Ouvidoria - Ciclo 1

Com base em todas as respostas coletadas, é possível perceber que não houve

nenhuma resposta do tipo discordo totalmente ou discordo parcialmente.

Adicionalmente, estabelecendo uma breve análise com relação às demais métricas coletadas, percebeu-se que o número de falhas contemplado em ambiente de produção reduziu de forma expressiva. Durante o período de uso do *framework*, em todos os projetos, constatou-se que não houve relato de falhas por parte das áreas de negócio e dos usuários.

Com relação ao índice de manutenibilidade nos projetos Portal, Sistema de Ouvidoria e Sistema de Perícia Médica, os dados coletados evidenciaram:

- aumento do índice para a maior parte das classes alteradas em relação ao índice de manutenibilidade médio constatado na fase de diagnóstico;
- obtenção de um excelente valor para as classes incluídas de acordo com os parâmetros da *Microsoft*.

Quanto ao projeto SICOR, percebeu-se, desde o início, pontuações totais baixas para a métrica Flog, evidenciando um código menos complexo e mais modularizado. Esse aspecto é fundamental para construção de sistemas, visto que possui relação direta com o quesito escalabilidade.

Outro aspecto que deve ser evidenciado é que durante a realização das inspeções foi possível constatar defeitos no código inerentes ao gerenciamento de recursos computacionais, tais como memória e processamento. Verificou-se defeitos dessa magnitude até mesmo no código das suítes de testes unitários. Contudo, em todos os projetos, os defeitos foram corrigidos antes da implantação em ambiente de produção. Adicionalmente, percebeu-se que as suítes de testes unitários elaboradas foram capazes de exercitar mais de uma vez o trecho de código sob teste, tornando a suíte mais eficaz.

O *framework*, de maneira geral, teve uma excelente aceitação e mostrou-se eficiente na verificação da qualidade do código.

7 Conclusão

Durante a realização deste trabalho, foi possível concluir a aplicação do plano metológico de pesquisa concebido para este estudo. O *framework* concebido durante a primeira parte do plano foi avaliado de acordo com os ciclos de pesquisa-ação executados nas instituições que se dispuseram a participar do estudo.

Os dados obtidos por meio da execução dos ciclos ressaltaram a importância de se estabelecer uma cultura de verificação de *software* em um processo de desenvolvimento. Em todos os projetos adotados para aplicação dos conceitos propostos pelo *framework*, percebeu-se a redução expressiva do número de falhas contemplado pelo usuário após entregas de *releases*. Além disso, percebeu-se que as inspeções, quando bem feitas, são precisas na identificação de defeitos no código.

Outro aspecto que foi evidenciado pela execução dos ciclos é que a realização de inspeções e implementação de testes unitários realmente se caracterizam como atividades complementares. No caso do Laboratório Fábrica de *Software*, embora a suíte de testes unitários não tenha sido concluída ao final dos ciclos, percebeu-se que a inspeção foi capaz de identificar defeitos que poderiam, futuramente, se apresentar como uma falha a partir da utilização do sistema.

É válido notar a partir dos dados apresentados que as heurísticas do *framework* inerentes à elaboração de testes unitários se mostraram eficazes na implementação da suíte de testes. Percebeu-se que as suítes foram capazes de exercitar mais de uma vez o trecho de código sob teste. Este aspecto demonstra que os testes consideraram mais de um cenário de uso dos módulos.

Adicionalmente, percebeu-se que os índices de manutenibilidade aumentaram em relação ao diagnóstico inicial dos projetos. Quando há uma maior preocupação com a qualidade do código implementado, os desenvolvedores projetam alternativas mais modulares de organização do código. Além disso, as inspeções foram capazes de evidenciar trechos de código que precisavam ser refatorados.

A coleta de dados efetuada durante a realização deste trabalho foi desafiadora para a equipe de pesquisa. Primeiramente, é importante notar que uma das instituições, no caso a CGDF, não possuía o mínimo da base de uma cultura de verificação de *software*. Felizmente, ao final dos dois ciclos, contemplou-se a excelente aceitação por parte dos integrantes da instituição. Em conversas com o diretor de tecnologia da instituição, foi dito que o uso do *framework* será permanente em todos os projetos de desenvolvimento. Por outro lado, no Laboratório Fábrica de *Software*, mesmo com a resistência inicial no uso do *framework*, os desenvolvedores mencionaram que o *framework* é promissor e pretendem

se organizar melhor para tornar seu uso permanente.

O procedimento técnico de pesquisa-ação também se mostrou como o mais apropriado para o contexto deste estudo. A partir da execução repetida de atividades, foi possível trabalhar uma mudança de percepção nos participantes do estudo. O único aspecto que realmente se manteve durante os ciclos foi a opinião dos desenvolvedores quanto à carga de trabalho trazida pelo uso do *framework*. Contudo, conforme afirmação da equipe do projeto SICOR, havendo disciplina, até mesmo uma equipe pequena é plenamente capaz de aplicar os procedimentos elencados pelo *framework*.

Como futuros trabalhos, poderia ser feito um acompanhamento completo de um projeto de desenvolvimento a fim de se coletar mais dados e estabelecer análises mais aprofundadas. A partir dessas análises seria possível pesquisar e acoplar mais práticas relacionadas à verificação de *software* e afirmar com mais propriedade como a qualidade de código pode ser plenamente obtida.

Referências

ANDALOUSSI, K. E. *Pesquisas-ações: ciências, desenvolvimento, democracia*. [S.l.]: Edufscar, 2004. Citado na página 44.

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 35 e 36.

BASILI, V. R.; ROMBACH, H. D. Goal question metric paradigm. 1994. Citado na página 46.

BERGEL, A.; PEñA, V. Increasing test coverage with hapao. 2012. Citado na página 38.

BIASI, L. B. Geração automatizada de drivers e stubs de teste para junit a partir de especificações u2tp. 2006. Citado na página 34.

BIFFL, S. et al. *Value-Based Software Engineering*. [S.l.]: Springer, 2014. Citado 4 vezes nas páginas 25, 27, 38 e 39.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.l.]: Domain Language, 2003. Citado na página 59.

FILHO, W. de P. P. *Engenharia de Software - Fundamentos, Métodos e Padrões*. [S.l.]: LTC, 2009. Citado 2 vezes nas páginas 31 e 32.

GANESAN, D. et al. An analysis of unit tests of a flight software product line. 2013. Citado 2 vezes nas páginas 35 e 36.

JúNIOR, J. R. de A. et al. Best practices in code inspection for safety-critical software. 2003. Citado na página 32.

KANER, C. Software negligence and testing coverage. Dept of Computer Sciences, Florida Tech, 1995. Citado 2 vezes nas páginas 26 e 27.

KITCHENHAM. Guidelines for performing systematic literature reviews in software engineering. 2007. Citado na página 41.

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 25 e 26.

MORESI, E. *Metodologia de pesquisa*. [S.l.]: Universidade Católica de Brasília, 2003. Citado na página 41.

PERSCHEID, M.; CASSOU, D.; HIRSCHFELD, R. Test quality feedback improving effectivity and efficiency of unit testing. 2012. Citado na página 37.

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 26 e 27.

SINGH, H.; BAWA, H. S. Management of effective verification and validation. 1995. Citado na página 31.

ZHU, H.; HALL, P. A. V.; MAY, J. H. R. Software unit test coverage and adequacy. 1997. Citado na página 37.

Anexos

ANEXO A – Detalhamento da Revisão Sistemática

Este anexo 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.