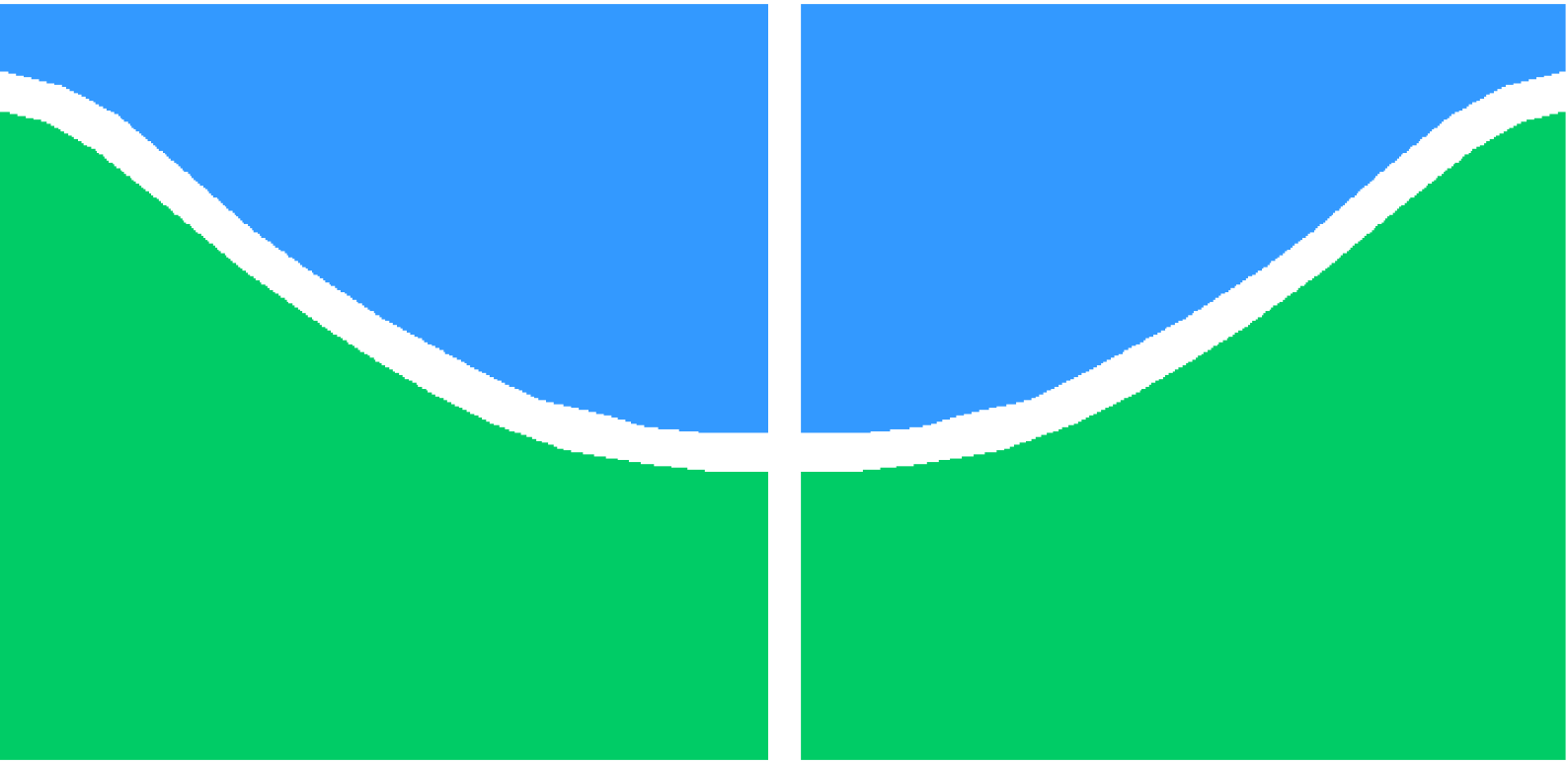


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Estudo de Caso da Melhoria do Processo de Verificação e Validação de Software em um Órgão do Setor Público

Autor: Renan Cristyan Araujo Pinheiro

Orientador: Dra. Fabiana Freitas Mendes

Brasília, DF

2022

Renan Cristyan Araujo Pinheiro

Estudo de Caso da Melhoria do Processo de Verificação e Validação de Software em um Órgão do Setor Público

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: Dra. Fabiana Freitas Mendes

Brasília, DF

2022

Renan Cristyan Araujo Pinheiro

Estudo de Caso da Melhoria do Processo de Verificação e Validação de Software em um Órgão do Setor Público/ Renan Cristyan Araujo Pinheiro. – Brasília, DF, 2022-

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

Orientador: Dra. Fabiana Freitas Mendes

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. Melhoria de Processo de Software. 2. Verificação e Validação. I. Dra. Fabiana Freitas Mendes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo de Caso da Melhoria do Processo de Verificação e Validação de Software em um Órgão do Setor Público

CDU 02:141:005.6

Renan Cristyan Araujo Pinheiro

Estudo de Caso da Melhoria do Processo de Verificação e Validação de Software em um Órgão do Setor Público

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).

Dra. Fabiana Freitas Mendes
Orientador

Dra. Elaine Venson
Convidado 1

Dr. Sérgio Antônio Andrade de Freitas
Convidado 2

Brasília, DF
2022

"Talvez não tenha conseguido fazer o melhor,
mas lutei para que o melhor fosse feito.
Não sou o que deveria ser, mas Graças a Deus,
não sou o que era antes" (Martin Luther King Jr.)

Resumo

Entregar produtos de software de alta qualidade é um dos objetivos principais de muitas fábricas de software, pois a qualidade de seus produtos influencia fortemente em suas reputações e, consequentemente, em seus lucros. Uma das maneiras de assegurar a qualidade do produto de software é através da identificação e tratamento de defeitos. Assim sendo, o processo de verificação e validação do produto é de importância fundamental. O objetivo deste trabalho é realizar uma análise do processo de verificação e validação de um produto de software em um órgão do setor público e propor algumas melhorias para os pontos fracos identificados. Para isso, foi realizado um estudo de caso em um dos projetos do órgão, com foco na interface de usuário. Primeiramente foi feita uma análise do processo em questão considerando os princípios do MPS.BR (Melhoria do Processo de Software Brasileiro) e do CMMI (Capability Maturity Model Integration). Após a identificação de pontos fracos, foram sugeridas formas de tratar os problemas, considerando a realidade do projeto e suas limitações. O projeto analisado executa tarefas de verificação e validação em alguns produtos de trabalho, porém não planeja e documenta o processo de maneira adequada. Com a implementação das sugestões propostas, a qualidade do processo de verificação e validação no projeto pode melhorar consideravelmente.

Palavras-chave: Qualidade de Software, Verificação e Validação, Melhoria de Processo de Software.

Abstract

Delivering high quality software products is one of the main goals of many software companies, because the quality of their products strongly influences their reputations and, consequently, on their profits. One of the ways to ensure the software product quality is through the identification and treatment of defects. Therefore, the product verification and validation process is fundamental. The main goal of this work is to carry out an analysis of the verification and validation process of a software product in a public sector agency and propose some improvements to the identified weaknesses. Because of this, a case study was carried out in one of the agency's projects, with focus on user interface. First, we carried out a process analysis considering the principles of MPS.BR (Melhoria do Processo de Software Brasileiro) and CMMI (Capability Maturity Model Integration). After identifying weaknesses, we suggested ways for dealing with them, considering the reality of the project and its limitations. The analyzed project performs verification and validation tasks on some work products, but does not properly plan and document the process. With the implementation of the proposed suggestions, the quality of the verification and validation process in the project can improve considerably.

Key-words: Software Quality, Verification and Validation, Software Process Improvement.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
ADR	Architectural Decisions Record
AFR	Afirmação
API	Application Programming Interface
CP	Capacidade do Processo
CMMI	Capability Maturity Model Integration
GQM	Goal, Question, Metric
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IND	Indicador Direto
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MPS	Melhoria de Processo de Software
MPS.BR	Melhoria do Processo de Software Brasileiro
MPT.BR	Melhoria do Processo de Teste Brasileiro
PF	Ponto Fraco
OE	Objetivo Específico
OG	Objetivo Geral
RAP	Resultado de Atributo de Processo
SEI	Software Engineering Institute
SQuaRE	Software product Quality Requirements and Evaluation
TCC	Trabalho de Conclusão de Curso

TS	Teste de Software
UX	User Experience
VV	Verificação e Validação

Sumário

1 Introdução

A engenharia de software é essencial no mundo moderno, pois diversas atividades dependem diretamente de software, como sistemas financeiros, sistemas governamentais, sistemas bancários, entretenimento, música, jogos, entre outras (SOMMERVILLE, 2011). Um dos principais fatores que influenciam o sucesso de projetos de software é a qualidade de software. As qualidades do produto, do processo e dos recursos podem influenciar diretamente no custo de um projeto de software, no tempo necessário para entregá-lo e na satisfação do usuário (SOMMERVILLE, 2011).

Além disso, para cada produto resultante de um processo de engenharia, o objetivo principal é gerar valor para o usuário, ao passo que se equilibre os custos e tempo de desenvolvimento. No caso de produtos de software, o usuário pode estar interessado em preço acessível, tempo de desenvolvimento curto e qualidade de software. A qualidade de software é alcançada ao se cumprir todos os requisitos de software, independente de como são agrupados ou nomeados (BOURQUE; FAIRLEY, 2014).

A qualidade de software é pesquisada amplamente devido a sua importância para o sucesso de projetos de software. Dessa forma, diferentes normas internacionais foram criadas visando classificar e organizar as subdivisões e características da qualidade de software, entre elas a série de normas ISO/IEC 25000, ou SQuaRE (Software product Quality Requirements and Evaluation) (ISO, 2005). O modelo de qualidade definido pela SQuaRE consiste em duas partes, o modelo para qualidade de software interna e externa e o modelo para qualidade em uso.

Os requisitos de qualidade interna especificam o nível de qualidade requirido do ponto de vista interno do produto de software e podem ser aplicados a elementos não executáveis do software, como documentos, manuais e artefatos (ISO, 2005). Por outro lado, os requisitos de qualidade externa tratam do nível de qualidade do ponto de vista externo. Esses requisitos derivam dos requisitos do usuário, utilizados como objetivos das verificações e validações técnicas de um produto de software (ISO, 2005). Em suma, os requisitos de qualidade em uso especificam os requisitos de qualidade do ponto de vista do usuário final. Esses requisitos derivam do contexto de uso, isto é, de que tipo de usuário se trata, do ambiente de uso, equipamentos utilizados e tarefas a serem executadas (ISO, 2005).

O processo de software é uma série de atividades executadas visando construir um produto de software (SOMMERVILLE, 2011). Técnicas, ferramentas, pessoal, documentação, cronograma e custos do projeto afetam diretamente a qualidade do produto final (SOMMERVILLE, 2011). Tendo isso em mente, existe um grande interesse por parte da

comunidade acadêmica em buscar maneiras de melhorar os processos existentes e criar técnicas inovadoras para reduzir custos de produção, encurtar o tempo de desenvolvimento e garantir um produto de software de alta qualidade que atenda às demandas de seus clientes.

1.1 Justificativa

A qualidade de um produto de software pode impactar diretamente em seu custo de produção, tempo de desenvolvimento e satisfação geral do usuário. Uma das formas de garantir que a qualidade está em níveis aceitáveis é através do processo de teste de software. Entretanto, mesmo sendo evidente a importância deste processo, muitas vezes ele é negligenciado, implementado de maneira incompleta ou inadequada, afetando diretamente a qualidade do produto.

Tendo em mente essa problemática, o trabalho atual visa realizar uma análise do processo de verificação e validação de um projeto real, visando identificar possíveis pontos fracos, e sugerir algumas melhorias que possam tratar os problemas identificados. Com a aplicação das melhorias propostas, o processo de verificação e validação do projeto poderá ser planejado, executado e analisado de maneira mais organizada e sistemática.

Inicialmente, o foco deste trabalho era analisar apenas o processo de teste de software. Porém, com o desenvolvimento do estudo de caso, o escopo do trabalho foi ampliado para englobar o processo de verificação e validação de uma maneira mais abrangente.

1.2 Objetivos

O objetivo geral (OG) deste trabalho é **avaliar e propor algumas melhorias ao processo de verificação e validação de software de um projeto real desenvolvido por um órgão do setor público**. Para que o objetivo geral seja considerado atingido, os seguintes objetivos específicos devem ser concluídos:

- OE1: Estudar os conteúdos de melhoria de processo de software e verificação e validação
- OE2: Avaliar um processo de verificação e validação
- OE3: Propor algumas melhorias no processo para lidar com os pontos fracos identificados

1.3 Cronograma

A Tabela 1 apresenta as etapas realizadas na primeira etapa do trabalho. Após o desenvolvimento da primeira etapa, foram realizadas adaptações e modificações para o TCC2.

Tabela 1 – Cronograma TCC1

Meses / Semanas																	
	Janeiro				Fevereiro				Março					Abril			
Etapas	1	2	3	4	1	2	3	4	1	2	3	4	5	1	2	3	4
Definição do tema	x																
Estudo teórico	x	x	x	x	x					x	x	x			x	x	
Escrita de fundamentação teórica		x	x	x	x												
Planejamento do estudo de caso					x	x	x	x									
Escrita da metodologia							x	x	x								
Estudo de métricas									x	x	x						
Seleção de métricas											x	x					
Revisão													x	x	x	x	
Entrega																	x

A Tabela 2 apresenta as datas aproximadas de desenvolvimento das principais etapas do trabalho, listando-as de acordo com as semanas dos meses de julho, agosto e setembro.

Tabela 2 – Cronograma TCC2

Etapas	Julho				Agosto					Setembro			
	1	2	3	4	1	2	3	4	5	1	2	3	4
Preparativos e adaptação do TCC1	X	X	X										
Estudo teórico		X	X	X									
Escrita de fundamentação teórica			X	X									
Planejamento da metodologia e do estudo de caso			X	X									
Estudo do contexto do projeto				X	X								
Execução da análise do processo de VV					X	X	X	X					
Identificação dos pontos fracos no processo de VV						X	X	X	X				
Proposição de algumas melhorias							X	X	X	X			
Revisão do texto										X	X	X	X
Entrega													X

1.4 Metodologia

Este trabalho pretende avaliar e propor algumas melhorias ao processo de verificação e validação de software de um projeto real. Para tanto, optou-se por executar um estudo de caso em um projeto de uma organização pública para aplicar os conceitos estudados. A metodologia, como pode ser observado na Figura 1, foi organizada em quatro fases principais: descoberta, investigação, proposição de melhorias e síntese.

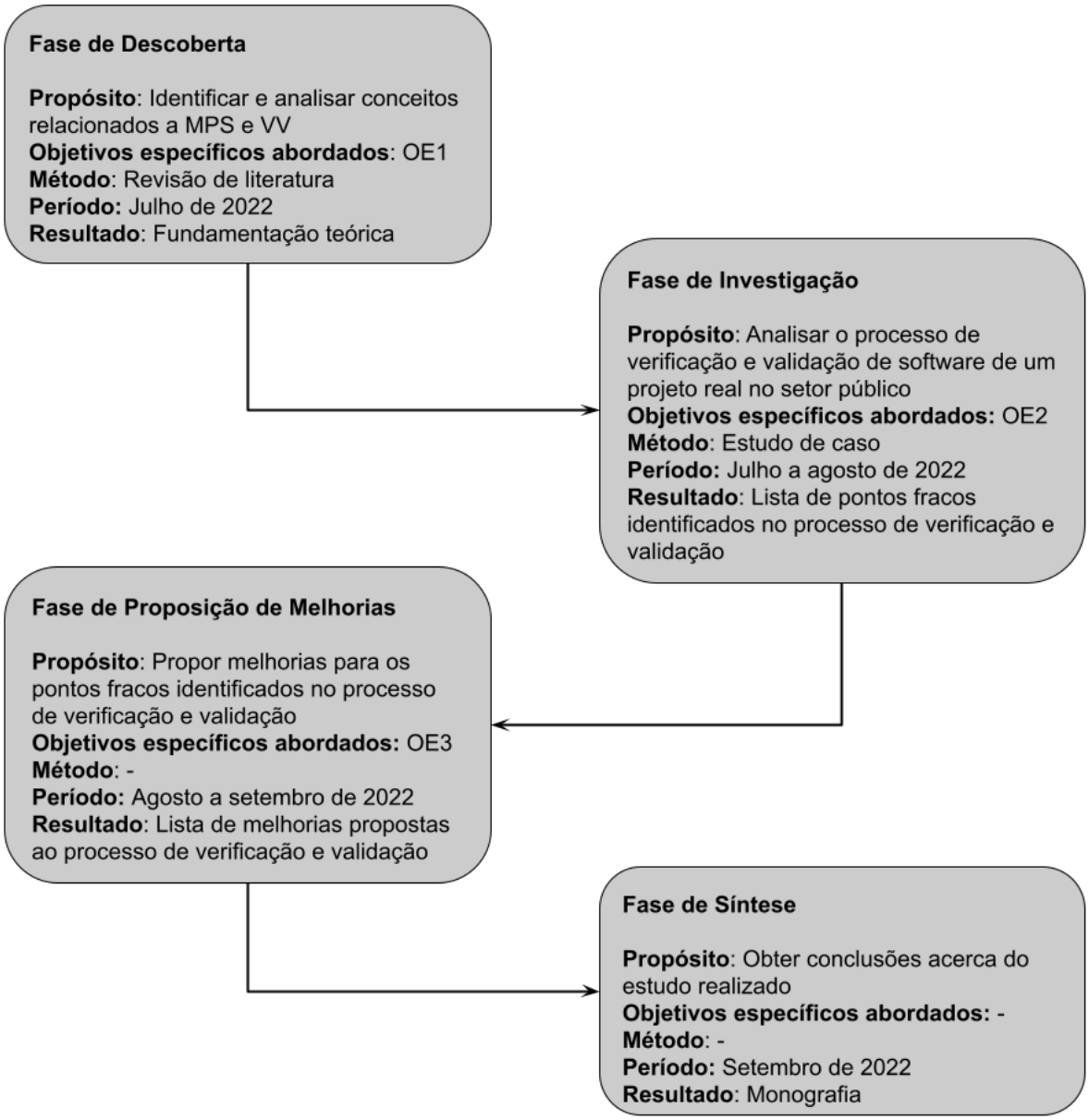


Figura 1 – Fases da metodologia

A fase de descoberta consistiu em realizar uma pesquisa com o intuito de entender os principais conceitos-chave, a fim de poder aplicá-los em uma situação real. Essa fase cumpriu com sucesso o OE1. Os resultados da pesquisa bibliográfica foram registrados no Capítulo 2.

A fase de investigação consistiu na aplicação de um estudo de caso descritivo em uma organização do setor público. Para tanto, foi inicialmente selecionado um projeto utilizando critérios de seleção relativamente simples. O projeto foi então estudado e compreendido por conversas e entrevistas com membros da equipe. Após a escolha do método de avaliação do processo de teste de software, foi realizada uma análise sobre o processo de verificação e validação do projeto, e os resultados da análise foram compilados em uma lista de pontos fracos identificados. Essa fase satisfez o OE2.

A fase de proposição de melhorias teve como objetivo utilizar os dados obtidos na fase anterior para propor algumas melhorias ao processo de verificação e validação de software do projeto. As melhorias propostas foram baseadas na literatura acadêmica a partir de modelos, artigos e boas práticas já utilizadas em outros contextos. Com isso, o OE3 foi atingido.

Finalmente, a fase de síntese tratou de concluir todo o trabalho realizado na forma de uma monografia a ser submetida à Universidade de Brasília como um dos requisitos para formação em engenharia de software.

1.4.1 Fase de Descoberta: Pesquisa bibliográfica

Na primeira etapa do trabalho foram realizadas as primeiras buscas por referencial teórico e formulação do Capítulo 2, além do estudo sobre como planejar e executar um estudo de caso. Os principais assuntos abordados foram qualidade de software, processo de software, melhoria de processo de software e teste de software. Sobre melhoria de processo de software, foi realizado um estudo sobre três importantes modelos da área: CMMI-DEV, MPS.BR e MPT.BR.

1.4.2 Fase de Investigação: Estudo de Caso

O estudo de caso deste trabalho foi desenvolvido segundo o *template* de protocolo sugerido por Brereton et al. (2008), além de inspirações do modelo proposto por Runeson e Höst (2009).

Os passos propostos por Brereton et al. (2008) são: definir o design do estudo de caso, preparar e coletar evidências, analisar os dados coletados e reportar os resultados do estudo.

O primeiro passo visa definir os objetivos do estudo de caso e realizar o planejamento inicial de sua execução. Os objetivos do estudo de caso são debatidos na Seção 1.4.2.1. O segundo passo aborda os critérios de seleção do estudo de caso, debatido na Seção 1.4.2.2. O terceiro passo determina como os dados serão coletados, definindo procedimentos e métodos a serem aplicados. Os procedimentos para coleta de dados são abordados na Seção 1.4.2.3.

1.4.2.1 Objetivos do Estudo de Caso

O objetivo principal do estudo de caso é identificar pontos fracos no processo de verificação e validação de software de um projeto real. Para tanto, foi observado um projeto em um "Orgão X", onde seu processo de verificação e validação foi avaliado utilizando dois dos modelos de avaliação de processos existentes, e, em outra fase da metodologia deste trabalho, foi realizada uma discussão acerca de possíveis melhorias aplicáveis aos pontos fracos identificados.

Para obter informações sobre o contexto do projeto, foram realizadas principalmente entrevistas semiestruturadas com membros da equipe para entender os processos internos. Além disso, foi realizada uma análise da documentação relevante para a execução do trabalho.

Para identificar pontos fracos no processo de teste do projeto, foi utilizada uma adaptação entre os modelos CMMI-DEV e MPS.BR. Os conceitos de representação contínua do CMMI-DEV foram aplicados para se avaliar o nível de capacidade do processo Verificação e Validação, descrito no guia de referência do MPS.BR. O processo de Verificação e Validação foi analisado considerando o nível G de capacidade do processo do MPS.BR, apesar de ser originalmente um processo do nível D de capacidade. Essa adaptação foi realizada, pois o objetivo deste trabalho não é avaliar outros processos de software não relacionados à verificação e validação. Também foi utilizado o guia de avaliação do MPS.BR para servir como orientação de como avaliar o processo de verificação e validação do projeto, além de utilizar a planilha de indicadores fornecida publicamente pelo MPS.BR como base para a avaliação das evidências.

Finalmente, foi elaborada uma lista de pontos fracos encontrados no processo de teste do projeto, que serviu como base para a fase de proposição de algumas melhorias.

Para a execução deste trabalho, optou-se por realizar um único estudo de caso de maneira holística. O contexto do estudo de caso é abordado na Seção 3.1. Os critérios de seleção do estudo de caso são debatidos na Seção 1.4.2.2.

1.4.2.2 Critérios de seleção

Com o intuito de selecionar um estudo de caso específico, foram determinados critérios de seleção que deveriam ser atendidos. Os critérios determinados foram:

1. O projeto deve estar em desenvolvimento;
2. O projeto deve possuir interface gráfica;
3. O projeto deve possuir pelo menos testes unitários implementados, e outros tipos de teste, se possível.

Após a filtragem e exclusão de casos que não cumpriram os critérios, foi realizado um contato com os responsáveis pelo projeto, para conversar sobre a possibilidade de realizar um estudo de caso, para ser utilizado como principal fonte de dados para a monografia, como requisito do trabalho de conclusão de curso de engenharia de software. O estudo de caso foi então aprovado pelos responsáveis. Para oficializar a realização da pesquisa na organização, um termo de consentimento livre e esclarecido foi entregue aos responsáveis antes da coleta de dados, com esclarecimentos sobre o que seria feito, como seria feito e que tipo de dados seriam necessários. Os responsáveis então formalizaram a liberação dos dados através de suas assinaturas. O conteúdo do termo assinado encontra-se no Apêndice B.

1.4.2.3 Plano de Coleta de Dados

Para se obter os dados necessários para a pesquisa, foi elaborado um plano de coleta de dados, com os seguintes passos: identificação e seleção dos dados, coleta dos dados, análise dos dados, validade do plano de coleta de dados, limitações do plano de coleta de dados, e relatório. Cada passo é descrito a seguir.

Um projeto de um orgão público, referenciado apenas por "Orgão X", foi escolhido para a realização do estudo de caso e obtenção de dados. Uma entrevista semiestruturada foi realizada visando entender o funcionamento do projeto, seus objetivos, metodologias de desenvolvimento e, principalmente, o processo de verificação e validação.

Os dados sobre qualidade do processo coletados foram relacionados ao processo Verificação e Validação (VV) do modelo MPS.BR, aplicando a representação contínua proposta pelo modelo CMMI-DEV. Em outras palavras, a forma de aplicação da avaliação do processo é um misto entre dois modelos. Dados adicionais sobre o processo, ou seja, medições de qualidade do produto, podem ser coletadas caso sejam necessárias para um entendimento aprofundado do contexto do projeto.

Os dados utilizados foram coletados e analisados conforme as diretrizes dos guias de referência e de avaliação do MPS.BR, além dos conceitos de representação contínua do CMMI-DEV. Foi utilizada uma planilha de avaliação fornecida publicamente pelo modelo MPS.BR, por ser um modelo de planilha completo e satisfazer com sucesso os objetivos da avaliação do processo do projeto. Com essas informações obtidas, foi possível listar os problemas identificados relacionados ao processo de teste, e cada problema teve um identificador para que pudessem ser referenciados nas conclusões do trabalho. Os detalhes relacionados à validade e às limitações do plano são discutidos na Seção 5.1.

Ao final, após realizar a análise dos dados, foi possível identificar pontos fracos no processo de verificação e validação de software do projeto. Esses pontos fracos foram agrupados em uma lista, onde cada item possui um identificador único que foi ser referenciado nas etapas seguintes do trabalho. Com isso, foi possível propor algumas melhorias

ao processo, conforme a literatura e considerando o contexto do projeto.

1.4.3 Fase de Proposição de Melhorias: Soluções Propostas

Após a obtenção da lista de pontos fracos identificados, foi feita uma nova pesquisa sobre como lidar com os problemas identificados, e então seus resultados foram documentados. Inicialmente foram utilizados os modelos abordados no estudo de caso, ou seja, o CMMI-DEV em conjunto com o MPS.BR, e pesquisado sobre o que os modelos previram sobre os problemas identificados. Dessa forma, ao final dessa fase foi elaborada uma lista de algumas melhorias propostas ao processo de verificação e validação, apresentando soluções plausíveis para cada ponto fraco identificado.

1.4.4 Fase de Síntese: Resultados do Trabalho

O intuito da última fase da metodologia foi agrupar todo o conhecimento obtido após a execução das fases anteriores, ou seja, após a pesquisa bibliográfica e o estudo de caso, e tirar conclusões sobre a análise do processo de verificação e validação de software do projeto estudado. Nessa conclusão foram feitas recomendações de como aplicar as melhorias propostas, bem como maneiras de se evitar os problemas mais comuns observados. Um detalhe importante a se destacar é que não faz parte do escopo deste trabalho acompanhar a implementação das melhorias propostas no projeto, porém essa etapa pode ser um trabalho futuro.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: O Capítulo 2 faz uma breve revisão dos conceitos-chave de interesse, além de apresentar definições e termos que servem de base para os capítulos seguintes. O Capítulo 3 aborda a avaliação informal do processo de teste de software do projeto, conforme o modelo escolhido e adaptado, exibindo os resultados obtidos após a avaliação e os pontos fracos identificados. O Capítulo 4 utiliza os resultados do capítulo anterior e a lista de pontos fracos identificados para sugerir algumas melhorias plausíveis de serem aplicadas ao processo de teste do projeto, considerando a forma de trabalhar da equipe de desenvolvimento. Finalmente, o Capítulo 5 traz um resumo do que foi estudado e conclui de maneira geral o trabalho, fazendo uma discussão sobre se os objetivos principais foram cumpridos ou não.

2 Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos teóricos para a execução do trabalho. A qualidade de software é um conceito base para o entendimento dos temas principais do trabalho, abordada inicialmente na Seção 2.1.

Os processos de verificação e validação são apenas alguns dos diferentes processos de software existentes. Para fornecer uma base relacionada a esses conteúdos, a Seção 2.2 apresenta definições e exemplos de modelos utilizados para auxiliar na melhoria de processo de software, com destaque para o modelo MPS.BR.

Finalmente, a Seção 2.3 apresenta os principais conceitos do processo de teste de software, além de abordar e distinguir os conceitos de verificação e validação, utilizados durante a fase de análise deste trabalho. Além disso, também é feita uma discussão sobre ferramentas de análise estática de código, apresentando uma das ferramentas utilizadas no projeto, o SonarQube.

O Apêndice A possui reúne algumas das definições discutidas neste capítulo para acesso rápido.

2.1 Qualidade de Software

Qualidade de software é "o grau com o qual um conjunto de características intrínsecas cumpre requisitos" (ISO, 1994). A qualidade de um produto de software está intimamente relacionada com os requisitos elicitados e estes servem como base para determinar os critérios de aceitação da qualidade (SANTIAGO, 2011).

A qualidade de software é um ramo essencial da engenharia de software, pois para qualquer produto desenvolvido, o objetivo principal é entregar um produto que atinja as expectativas do cliente enquanto se equilibra os custos e o tempo do projeto (BOURQUE; FAIRLEY, 2014).

Obter um nível satisfatório de qualidade em um produto gera gastos pela natureza imprescindível do próprio software, pois o mesmo é um produto essencialmente intelectual, estando assim suscetível a erro durante seu ciclo de vida (KRASNER, 1998 apud OLIVEIRA; PETRINI; PEREIRA, 2015).

Os custos de qualidade são divididos em duas categorias, custos de conformidade e de não-conformidade, e quatro subcategorias, custos de prevenção, avaliação, falhas internas e falhas externas, relacionadas como exemplificado pela Figura 2.

Custos de prevenção são aqueles relacionados a atividades como investimento em

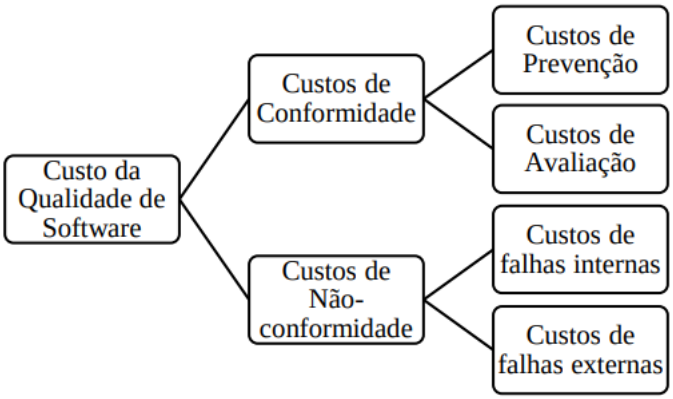


Figura 2 – Modelo de custo da Qualidade de Software. Fonte: (OLIVEIRA; PETRINI; PEREIRA, 2015)

ferramentas, infraestrutura de qualidade, treinamentos e melhoria de processo de software. Normalmente estes custos não estão atrelados a apenas um projeto específico, mas englobam toda a organização. Custos de avaliação são aqueles derivados de atividades que encontram defeitos, tais como testes e revisões. Custos de falhas internas são aqueles relacionados a corrigir os defeitos encontrados durante os testes e revisões antes da entrega do produto ao cliente. Custos de falhas externas são aqueles relacionados a tratar problemas no software encontrados após a entrega do software ao cliente. (BOURQUE; FAIRLEY, 2014)

Durante anos ocorreram debates entre os especialistas sobre como padronizar e determinar o que caracteriza a qualidade de software. Uma das principais normas que visaram atingir este objetivo foi a ISO/IEC 9126.

A ISO/IEC 9126 é uma norma cujo objetivo é descrever um modelo de qualidade do produto de software, composto de qualidade interna, qualidade externa e qualidade em uso (ABNT, 2003). Ela faz uma revisão da norma NBR 13596, que foi uma norma anterior que definiu seis características de qualidade de software além de outros conceitos. Novas inclusões foram feitas na norma ISO/IEC 9126 em comparação com sua antecessora, entre elas a adição do importante conceito de qualidade em uso e a especificação de um modelo de qualidade. Os conceitos de qualidade interna, qualidade externa e qualidade em uso são abordados e detalhados no documento, demonstrando as relações entre si e como uma impacta direta ou indiretamente na outra (ABNT, 2003). As relações entre os modelos de qualidade são ilustrados na Figura 3.

Apesar de ter servido como referência por muitos anos, a SQuaRE substituiu a norma ISO/IEC 9126 por ser mais completa e atualizada, adicionando novos conceitos e tópicos relevantes, além de exemplos e uma reorganização geral na forma de divisões, cada uma abordando tópicos diferentes sobre qualidade de software.

SQuaRE (*Software product Quality Requirements and Evaluation* — Requisitos e

Avaliação de Qualidade de produto de Software) é uma série de normas de qualidade de software cujo objetivo é estruturar a avaliação da qualidade dos produtos de software. Definida pela norma ISO/IEC 25000 (ISO, 2005), consiste em cinco divisões, sendo elas:

- Divisão de Gestão da Qualidade (ISO/IEC 2500n): Define os termos e definições comuns às outras divisões, apresentando os principais termos abordados e realizando uma visão geral sobre o modelo.
- Divisão de Modelo da Qualidade (ISO/IEC 2501n): Detalha modelos de qualidade para sistemas computacionais, produtos de software, qualidade em uso e dados. Descreve, entre outras coisas, o modelo de qualidade de produto de software, definindo suas características e subcaracterísticas, além de abordar a qualidade em uso de software.
- Divisão de Medição de Qualidade (ISO/IEC 2502n): Apresenta definições matemáticas de métricas de software, um modelo de referência de medição de qualidade de produto de software, além de um guia de suas aplicações.
- Divisão de Requisitos de Qualidade (ISO/IEC 2503n): Provê guias e recomendações para desenvolver requisitos de qualidade. Esses requisitos podem ser utilizados no processo de elicitação de requisitos de um produto de software a ser desenvolvido ou como entrada em um modelo de avaliação.
- Divisão de Avaliação da Qualidade (ISO/IEC 2504n): Provê requisitos, recomendações e guias para a avaliação do produto de software.

A norma ISO/IEC 25010 apresenta dois modelos de qualidade presentes nas normas SQuaRE (ISO, 2010): modelo de qualidade em uso e modelo de qualidade do produto. O modelo de qualidade em uso define cinco características relacionadas à interação com o sistema. Já o modelo de qualidade do produto apresenta características e propriedades de qualidade de software divididas em oito categorias, cada uma com suas respectivas subcaracterísticas. A SQuaRE também define o modelo de qualidade de dados, porém ele é detalhado na norma ISO/IEC 25012.

A qualidade do processo afeta diretamente a qualidade do produto, que afeta diretamente a qualidade em uso. Portanto, para melhorar a qualidade em uso de um produto de software, pode-se investir em melhoria da qualidade interna e externa (BALTHAZAR, 1981).

A qualidade do produto é dividida em duas partes: qualidade interna e qualidade externa. Qualidade interna é o conjunto das características de um produto de software do ponto de vista interno, ou seja, perceptível pelos desenvolvedores, mas não perceptível diretamente pelo usuário. Já a qualidade externa é o conjunto de características do ponto

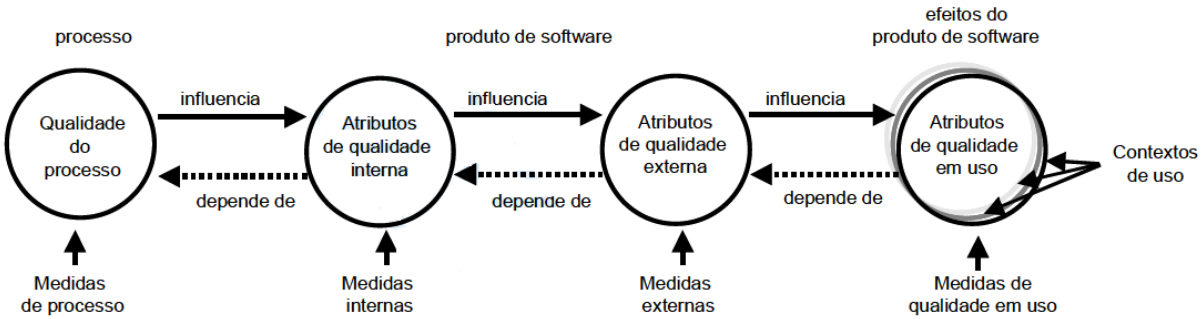


Figura 3 – Qualidade no ciclo de vida. Fonte: (ISO, 2010)

de vista externo. A qualidade externa é perceptível quando o software está sendo executado, o que pode ser analisado utilizando métricas apropriadas enquanto se testa em um ambiente controlado com dados simulados (ABNT, 2003).

2.2 Processo de Software

Processo de software é uma série de atividades, ações e tarefas necessárias para a produção de um produto de software de alta qualidade. (PRESSMAN, 2016). Vários processos de software foram criados e experimentados ao longo dos anos, porém existem quatro atividades fundamentais que devem estar presentes nos processos (SOMMERVILLE, 2011):

- Especificação de software: Identificar quais funcionalidades o produto deve apresentar, e quais restrições ele possuirá;
- Projeto e implementação de software: Planejar e construir o produto visando cumprir as especificações;
- Validação de software: Assegurar que o produto construído atenda às expectativas do cliente;
- Evolução de software: Garantir que as mudanças nas necessidades do cliente sejam atendidas.

Diferentes processos são apropriados para diferentes tipos de software. Por exemplo, um sistema de controle de um veículo autônomo possui um processo de software distinto de um jogo eletrônico.

A existência de um processo de software definido e otimizado em um projeto não garante que o produto será entregue no prazo esperado ou que irá satisfazer as expectativas do cliente. Os processos de software são atividades centradas em humanos, estando sujeitos

a erros ou resultados diferentes dos esperados (UNTERKALMSTEINER et al., 2011). Por isso, é importante utilizar modelos de melhoria de processos de software para manter o controle do projeto, prever resultados e garantir um nível de qualidade geral satisfatório.

A qualidade do processo de software influencia diretamente na qualidade do produto (PRESSMAN, 2016). Técnicas, ferramentas e recursos humanos tem papel crucial na qualidade do produto final. O investimento na melhoria do processo de software visa, além da melhoria na qualidade do produto, reduzir custos de implementação e o tempo para se entregar o produto (UNTERKALMSTEINER et al., 2011). A Figura 4 ilustra os principais fatores que influenciam a qualidade do produto, já mencionados anteriormente.

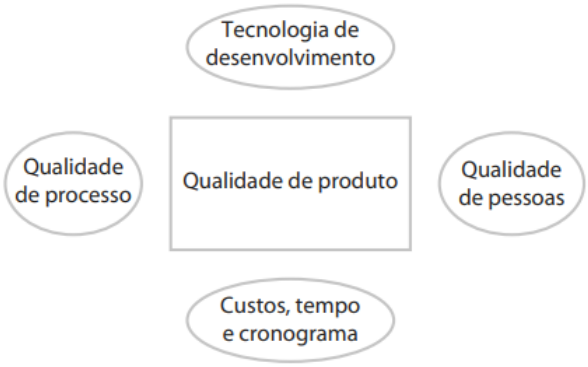


Figura 4 – Fatores que influenciam a qualidade do produto. Fonte: (SOMMERVILLE, 2011)

Mesmo com processos de software estabelecidos, sempre existem oportunidades de melhorias, dependendo dos objetivos da organização. Por exemplo, se o objetivo for a melhoria da qualidade de software, é possível implementar tarefas e atividades que modificam a forma que o produto é desenvolvido e testado (SOMMERVILLE, 2011).

Não existe modelo de processo de software ideal, por isso sempre existe a possibilidade de melhorias no processo (SOMMERVILLE, 2011). A melhoria de processo de software é uma maneira de se buscar atingir processos de alta qualidade que geram produtos de alta qualidade (SOMMERVILLE, 2011).

A medição de processo de software é um importante aspecto para a melhoria de processo de software, e fornece pelo menos dois benefícios (UNTERKALMSTEINER et al., 2011):

- A medição permite avaliar e comparar os modelos de melhoria de processo de software, suas estratégias adotadas e seus impactos.
- Ao tornar visível os resultados da aplicação da melhoria de processo de software, são gerados incentivos para se continuar e é justificável o esforço realizado.

A medição de processos de software, por si só, não é suficiente para determinar se a qualidade do produto melhorou. Também deve ser realizada a medição de atributos do produto e esses dados devem ser relacionados (SOMMERVILLE, 2011). Existem três tipos principais de métricas de processo que devem ser coletados (SOMMERVILLE, 2011):

- Tempo: tempo necessário para se planejar, executar, documentar um processo, etc.
- Recursos: dinheiro, pessoas necessárias para execução, memória e capacidade de processamento.
- Número de ocorrências de um evento: número de defeitos encontrados, quantidade de linhas de código alteradas, etc.

O processo de melhoria de processo de software é um ciclo de três subprocessos (conforme ilustrado na Figura 5), que consiste em medir o processo, analisar o processo e realizar mudanças.

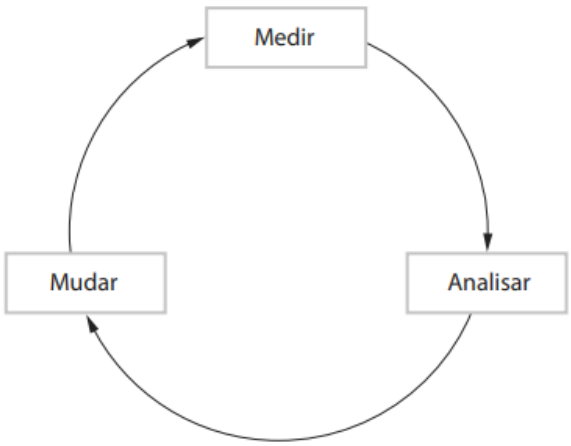


Figura 5 – Ciclo de melhoria de processos. Fonte: (SOMMERVILLE, 2011)

O primeiro subprocesso do ciclo de melhoria faz a medição dos atributos do processo. Isso será utilizado para realizar comparações entre os modelos e determinar se as mudanças surtiram o efeito esperado ou não. O subprocesso de analisar consiste em observar os gargalos e pontos fracos do processo. Em outras palavras, nesse subprocesso são identificados os pontos que necessitam de melhoria. Finalmente, no subprocesso de mudar são propostas melhorias para os pontos fracos observados anteriormente. Após as mudanças terem sido implementadas, um novo ciclo se inicia e então é avaliado se as mudanças foram satisfatórias (SOMMERVILLE, 2011).

Muitos modelos e normas surgiram ao longo dos anos visando auxiliar na melhoria de processos de software. Entre eles, as normas ISO/IEC 12207 e 15504 que servem como base para muitos modelos modernos, e os modelos CMMI (*Capability Maturity Model*

Integration), MPS.BR (Melhoria de Processo do Software Brasileiro) e MPT.BR (Melhoria do Processo de Teste Brasileiro).

A seguir serão feitas abordagens resumidas sobre os modelos CMMI, na Seção 2.2.1, MPS.BR, na Seção 2.2.2, e MPT.BR, na Seção 2.2.3. O entendimento desses modelos é uma etapa fundamental antes da realização da análise do processo de verificação e validação no projeto.

2.2.1 CMMI

Os modelos CMMI (*Capability Maturity Model Integration*) são uma coleção de melhores práticas para auxiliar as organizações a melhorar seus processos (CMMI, 2010). Foram desenvolvidos em conjunto por membros da indústria, do governo e do SEI (*Software Engineering Institute*).

O guia CMMI-DEV é especificamente voltado para o desenvolvimento de produtos e serviços. Aplica os conceitos e boas práticas do CMMI geral com o intuito de auxiliar no desenvolvimento de produtos de alta qualidade que atinjam as expectativas dos clientes e usuários finais (CMMI, 2010). O guia apresenta uma estrutura que abrange todo o ciclo de desenvolvimento de um produto de software, desde seu planejamento inicial até a manutenção e evolução.

Possui 22 áreas de processos representando 22 agrupamentos de atividades, procedimentos e tarefas. Cada uma relacionada com um processo diferente no ciclo de desenvolvimento de um produto de software. Cada área de processo possui objetivos específicos e objetivos gerais, sendo um conjunto de características que devem estar presentes no processo da organização para a área de processo ser considerada implementada satisfatoriamente (CMMI, 2010). Das 22 áreas de processos do CMMI-DEV, dezesseis são áreas de processos-base, uma é área de processo compartilhada e cinco são específicas de desenvolvimento de software, sendo elas: abordagem do desenvolvimento de requisitos, solução técnica, integração do produto, verificação e validação (CMMI, 2010).

O CMMI-DEV possui dois tipos de representação: contínua e por estágios (CMMI, 2010). A representação contínua permite a organização a alcançar níveis de capacidade, enquanto a representação por estágios permite alcançar níveis de maturidade. A principal diferença entre os dois tipos de representação é que, enquanto na contínua as áreas de processo podem ser avaliadas individualmente, na representação por estágios a avaliação deve ser realizada para todas as áreas de processos que compõem o nível de maturidade selecionado (MELLO, 2011). A representação por estágios não faz parte do escopo desse trabalho, então não será abordada em detalhes.

Cada tipo de representação possui seus próprios níveis, descrições e características. Os níveis de capacidade são apresentados na Tabela 3. A seguir tem-se uma descrição dos

Tabela 3 – Níveis de capacidade do CMMI-DEV. Fonte: (CMMI, 2010)

Nível	Representação contínua Nível de capacidade
0	Incompleto
1	Desempenhado
2	Gerenciado
3	Definido
4	<não existe>
5	<não existe>

níveis de **capacidade** fornecido pelo CMMI-DEV (CMMI, 2010):

- Nível 0 - incompleto: um processo é considerado incompleto quando não é desempenhado ou é parcialmente desempenhado. Pelo menos um objetivo específico de sua área de processo não é atingido. Não existem objetivos genéricos nesse nível, uma vez que um processo parcialmente desempenhado não tem necessidade de ser institucionalizado.

- Nível 1 - desempenhado: um processo é considerado desempenhado quando realiza o trabalho necessário para produzir os resultados esperados pelo processo. Os objetivos específicos da área de processo são atingidos, porém, o processo ainda não é institucionalizado por não atingir objetivos genéricos.

- Nível 2 - gerenciado: um processo é considerado gerenciado quando é um processo desempenhado, planejado e executado de acordo com políticas organizacionais. São aplicados os recursos disponíveis para se produzir resultados previsíveis. O processo é monitorado, controlado e revisado para se manter alinhado à sua própria descrição.

- Nível 3 - definido: um processo é considerado definido quando é gerenciado, porém, é firmemente alinhado com o conjunto padrão de processos da organização, e seus resultados contribuem para a padronização de processos da organização de uma maneira geral.

Uma das principais diferenças entre o nível 2 (gerenciado) e o nível 3 (definido) de capacidade é que, no nível 2 podem existir diferenças na padronização do processo em diferentes instâncias (por exemplo, o mesmo processo aplicado em diferentes projetos da organização podem ter padrões diferentes), enquanto no nível 3, o processo é documentado, mantido e gerenciado de uma maneira mais abrangente e consistente pela organização. Além disso, o rigor para documentar todo o processo, suas entradas, saídas, resultados esperados e outros fatores é muito maior no nível 3 do que no nível 2 (CMMI, 2010).

2.2.2 MPS.BR

O MPS.BR (Melhoria de Processo do Software Brasileiro) é uma iniciativa para melhorar a capacidade de desenvolvimento de software nas organizações brasileiras (KALINOWSKI et al., 2010), criado em 2003 pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), com apoio do Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTIC), Financiadora de Estudos e Projetos (FINEP), Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) e Banco Interamericano de Desenvolvimento (BID) (SOFTEX, 2021a).

Tem como um dos principais objetivos melhorar a competitividade das organizações brasileiras desenvolvedoras de software, através da disseminação de um modelo de processo de software que seja robusto e, simultaneamente, economicamente viável para pequenas e médias empresas (KALINOWSKI et al., 2010).

O MPS.BR possui três modelos de referência: MR-MPS-SV, como referência para serviços, MR-MPS-RH, como referência para gestão de pessoas, e MR-MPS-SW, como referência para software. O modelo possui também um guia denominado método de avaliação, MA-MPS, cujo objetivo é auxiliar na avaliação da aplicação dos conceitos relacionados ao MPS.BR. Apenas o MR-MPS-SW e o MA-MPS fazem parte do escopo deste trabalho, pois questões relacionadas a serviços e gestão de pessoas não são relacionadas diretamente ao tema principal.

2.2.2.1 Modelo de referência - MR-MPS-SW

O modelo de referência MPS para software (MR-MPS-SW) apresenta os termos, definições e conceitos para o entendimento e aplicação do MPS.BR por parte das organizações que implementam produtos de software. (SOFTEX, 2021a).

O MPS.BR baseia-se em normas internacionais renomadas e amplamente conhecidas, como a ISO/IEC 12207 e a ISO/IEC 15504, que abordam, entre outros assuntos, a avaliação da capacidade de processos baseada em modelos e processos específicos do ciclo de vida de produtos de software (KALINOWSKI et al., 2010). Além disso, o modelo é compatível com outro modelo internacional de grande importância, o CMMI (*Capability Maturity Model Integration*).

As normas internacionais existentes antes da criação do modelo MPS.BR apresentavam formas de implementar as melhorias de processos, porém, os custos envolvidos para realizá-las eram altos, tornando a tarefa difícil principalmente para pequenas e médias empresas. Essa é uma preocupação que o modelo MPS.BR visa abordar diretamente através de seus guias de implementação e disseminação de boas práticas de engenharia de software (SOFTEX, 2021a).

O MR-MPS-SW define níveis de maturidade, classificados de A a G, sendo uma

combinação de processos e capacidade (SOFTEX, 2021a). Os processos definidos pelo MR-MPS-SW são classificados em dois conjuntos: processos de projeto e processos organizacionais, conforme ilustrado na Figura 6.

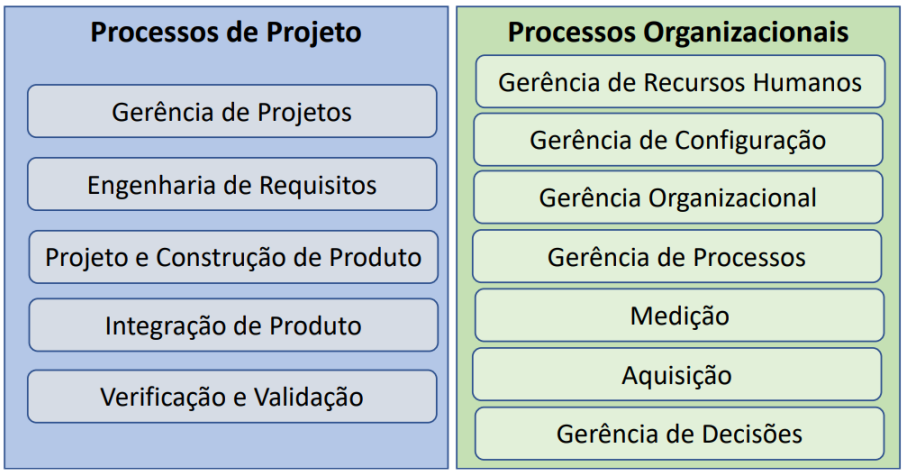


Figura 6 – Conjuntos de processos definidos pelo MR-MPS-SW. Fonte: (SOFTEX, 2021a)

Os processos de projeto são processos executados no desenvolvimento de software, ou seja, tem o intuito de produzir um novo produto, realizar sua manutenção ou evolução. Já os processos organizacionais são processos que dão suporte ao projeto de software, assegurando que os recursos disponíveis sejam bem utilizados e que as expectativas do projeto sejam atingidas (SOFTEX, 2021a).

Os processos descritos no MPS.BR tem um conjunto de resultados esperados que devem ser atingidos para poderem ser considerados implementados satisfatoriamente. Foram definidos sete níveis de maturidade, conforme ilustrado na Figura 7, sendo eles: A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado), sendo o nível G o primeiro e o nível A o mais maduro (SOFTEX, 2021a).

Um dos focos deste trabalho é a avaliação do processo de testes (verificação e validação) de um projeto real, portanto o processo de Verificação e Validação (VV), pertencente ao nível D de maturidade, foi escolhido como base. Uma observação relevante é que, nas versões anteriores do MPS.BR, o processo de VV era considerado dois processos separados.

Cada nível possui uma capacidade de processo, indo de CP-G até CP-A. A capacidade de processo "caracteriza o quanto o processo é capaz de alcançar os objetivos de negócio atuais e futuros" (SOFTEX, 2021a). Assim como os processos, os níveis de capacidade de processo possem resultados esperados que devem ser implementados para serem considerados satisfeitos. As versões mais antigas do modelo, anteriores a 2021, definiam o termo RAP (Resultado de Atributo de Processo) para referenciar os itens que devem ser

Figura 7 – Níveis de maturidade do MPS.BR. Fonte: (SOFTEX, 2021a)

implementados para que o nível de capacidade de processo fosse considerado satisfeito. Porém, a nova versão do MPS.BR utiliza o termo "resultados esperados" para processos e capacidade de processos.

Os níveis de maturidade do MPS.BR são compatíveis com os níveis de maturidade do modelo CMMI, e a relação entre os níveis dos diferentes modelos é ilustrada pela Figura 8. Como o MPS.BR possui dois níveis a mais, a mudança de nível é mais gradual por parte da organização, se tornando mais economicamente viável para pequenas e médias empresas.

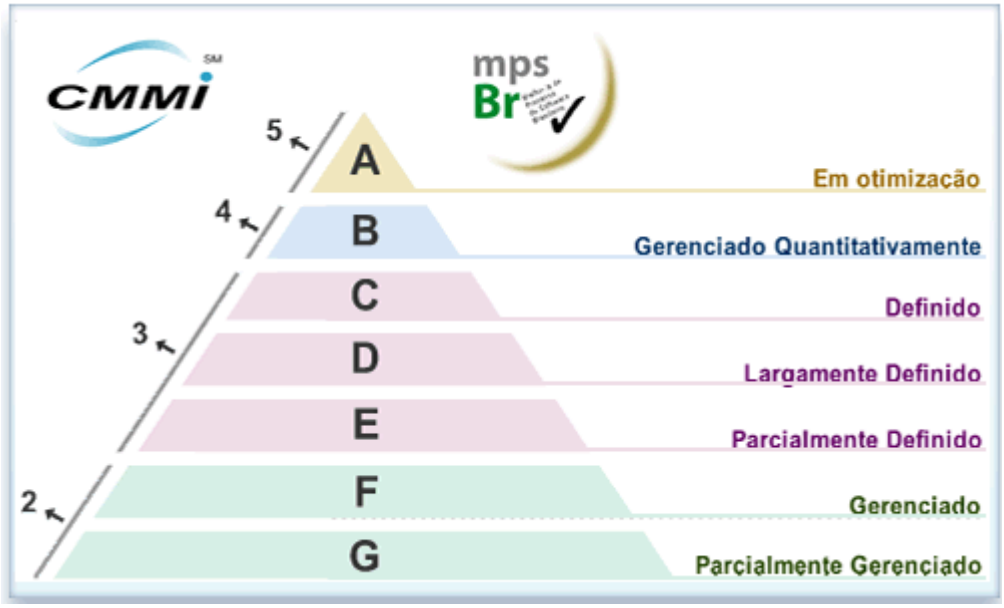


Figura 8 – Relação entre níveis de maturidade do MPS.BR e do CMMI. Fonte: (QUALIDADEBR, 2022)

A seguir, na Seção 2.2.2.2, são apresentados os principais conceitos relacionados ao Método de Avaliação do MPS.BR (MA-MPS), do qual foram retirados princípios para

a avaliação do processo de verificação e validação realizada neste trabalho.

2.2.2.2 Método de avaliação — MA-MPS

O MA-MPS é um guia que auxilia na avaliação de conformidade dos modelos MPS.BR (SOFTEX, 2021a). O nível de implementação de um processo é avaliado a partir de indicadores (ou evidências). Os indicadores podem ser de dois tipos, diretos ou afirmações. Indicadores diretos são artefatos que comprovem que o resultado esperado foi cumprido, ou o produto principal da realização de uma determinada tarefa. Afirmações são obtidas por entrevistas e apresentações e confirmam a implementação de processos, resultados esperados e nível de capacidade de processo (SOFTEX, 2021b).

Durante o processo de avaliação, resultados esperados dos processos e dos níveis de capacidade são classificados conforme o grau de implementação, como visto nas Tabelas 4 e 5. O grau de implementação de um resultado esperado é classificado entre totalmente, largamente, parcialmente e não implementado. A partir da análise de indicadores diretos, afirmações e pontos fracos relacionados a cada resultado esperado, é possível atribuir um grau de implementação e, ao final, é possível confirmar se o processo em questão satisfaz o modelo MPS.BR ou não.

Tabela 4 – Graus de implementação de um resultado esperado do processo. Fonte: (SOFTEX, 2021b)

Grau de implementação	Caracterização
Totalmente implementado (T)	- O indicador direto está presente e é julgado adequado - Existe pelo menos uma afirmação confirmando a implementação - Não foi notado nenhum ponto fraco substancial
Largamente implementado (L)	- O indicador direto está presente e é julgado adequado - Existe pelo menos uma afirmação confirmando a implementação - Foi notado um ou mais pontos fracos substanciais
Parcialmente implementado (P)	- O indicador direto não está presente ou é julgado inadequado - Artefatos/afirmações sugerem que alguns aspectos do resultado esperado estão implementados - Foi notado um ou mais pontos fracos substanciais
Não implementado (N)	- Qualquer situação diferente das acima
Não avaliado (NA)	- O projeto/serviço/área não está na fase de desenvolvimento que permite atender ao resultado ou não faz parte do escopo do projeto atender ao resultado
Fora do escopo (F)	- O resultado esperado está fora do escopo da avaliação

Tabela 5 – Graus de implementação do nível de capacidade de processo. Fonte: (SOFTEX, 2021b)

Grau de implementação	Caracterização	Porcentagem de implementação dos resultados relacionados
Totalmente implementado (T)	Existe evidência da implementação completa e sistemática da capacidade de processo no processo avaliado. Não existem pontos fracos relevantes para esta capacidade de processo no processo avaliado.	>85% a 100%
Largamente implementado (L)	Existe evidência de um grau significativo de implementação da capacidade de processo no processo avaliado. Existem pontos fracos para esta capacidade de processo no processo avaliado.	>50% a 85%
Parcialmente implementado (P)	Existe evidência para alguma implementação da capacidade de processo no processo avaliado. Alguns aspectos de implementação não são possíveis de predizer. Existem pontos fracos para esta capacidade de processo no processo avaliado.	>15% a 50%
Não implementado (N)	Existe pouca ou nenhuma evidência da implementação da capacidade do processo no processo avaliado.	0 a 15%

Um processo é classificado como satisfeito quando todos os resultados esperados foram classificados como T (totalmente implementado) ou L (largamente implementado) e os resultados esperados da capacidade de processo do nível em que se encontra também for classificado como T ou L. Em quaisquer outras situações, o processo é considerado não satisfeito (SOFTEX, 2021b).

Para uma organização ser classificada em um dos níveis de maturidade do MPS.BR, todos os processos e a capacidade do processo do nível específico devem ser classificados como satisfeitos (SOFTEX, 2021b).

2.2.3 MPT.BR

O MPT.BR (Melhoria do Processo de Teste Brasileiro) é um modelo de melhoria do processo de testes desenvolvido pela Softex Recife que visa melhorar a capacidade de teste de software principalmente de micro e pequenas empresas, utilizando as melhores

práticas desenvolvidas pela academia e pela indústria (SOFTEXRECIFE, 2011). Seus principais objetivos são: tornar-se um modelo de referência para questões relacionadas a melhorias de processo de teste de software, auxiliar na melhoria contínua de processos de teste de software conforme o nível de maturidade da organização, fornecer base para identificação e avaliação do grau de maturidade do processo de teste das organizações, e reunir as melhores práticas e conciliar com a complexidade conforme o grau de maturidade das organizações (SOFTEXRECIFE, 2011).

O modelo tem como base diversos outros modelos de teste de software e melhoria de processo de software, como TSM (*Testability Support Model*), TMM (*Testing Maturity Model*), TMMI (*Testing Maturity Model Integration*), CMMI (*Capability Maturity Model Integration*), MPS.BR (Melhoria do Processo de Software Brasileiro) e outros.

Dentre os principais diferenciais do MPT.BR para os outros modelos, pode-se citar o foco nas características e necessidades de micro e pequenas empresas desenvolvedoras de software, processos de implantação curtos, baixo custo de implementação se comparado com modelos de maturidade internacionais e compatibilidade com os principais modelos de maturidade internacionais e práticas ágeis (SOFTEXRECIFE, 2011).

O MPT.BR possui dois guias principais, o modelo de referência, que apresenta a estrutura básica do modelo com seus conceitos, áreas de processo e práticas; e o guia de avaliação, que apresenta instruções para avaliar uma organização segundo o MPT.BR (SOFTEXRECIFE, 2011). A partir da avaliação de uma organização, ela pode ser classificada em cinco níveis: Nível 1 - Parcialmente Gerenciado, Nível 2 - Gerenciado, Nível 3 - Definido, Nível 4 - Prevenção de Defeitos e Nível 5 - Automação e Otimização. A relação entre os níveis de maturidade e suas respectivas áreas de processo é ilustrada na Figura 9. As áreas de processo definidas pelo modelo MPT.BR estão listadas na Tabela 6.

Apesar de ser um modelo especificamente voltado para a melhoria do processo de teste, um dos principais focos deste trabalho, ele não foi escolhido para realizar a avaliação do processo de verificação e validação no projeto. Os detalhes da justificativa são abordados na Seção 3.2.

Finalizada a discussão sobre processos de software, é necessário abordar especificamente o processo principal do escopo deste trabalho, o processo de verificação e validação, aprofundado na Seção 2.3.

2.3 Verificação e Validação

Verificação e validação são dois processos de software que costumam ser executados em conjunto, e pode ser difícil determinar onde um começa e outro termina. São utilizados para testar o produto e seus artefatos associados. O processo de teste de software tem

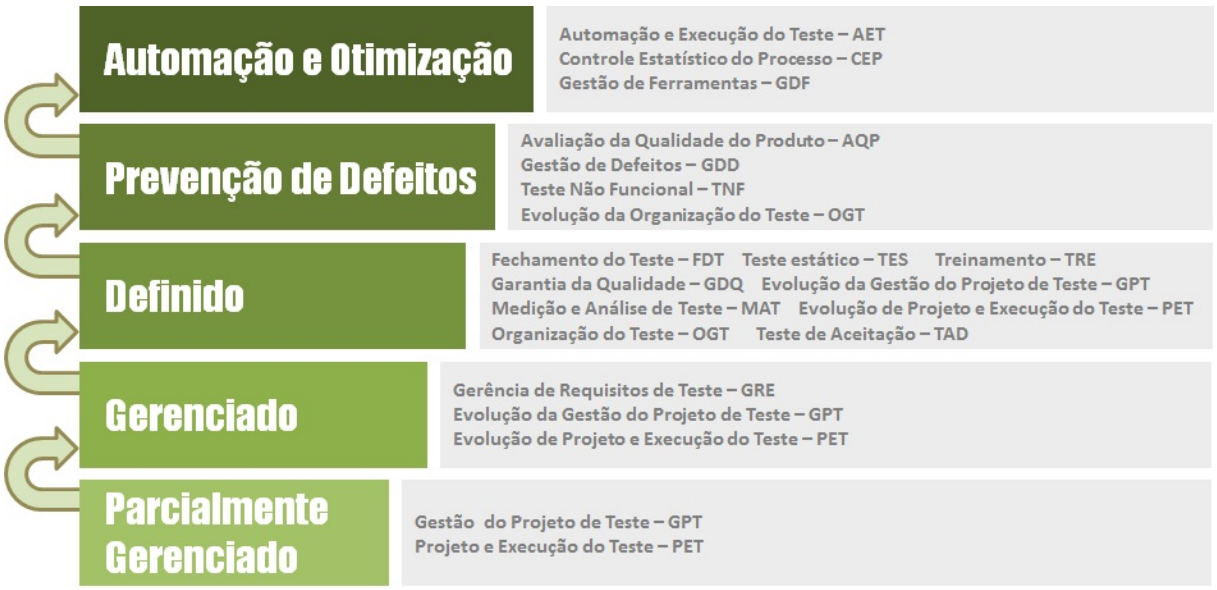


Figura 9 – Níveis de maturidade do modelo MPT.BR. Fonte: (SOFTEXRECIFE, 2011).

Tabela 6 – Áreas de processo do modelo MPT.BR. Fonte: (SOFTEXRECIFE, 2011)

ID	Área de Processo
AET	Automação e Execução do Teste
AQP	Avaliação da Qualidade do Produto
CEP	Controle Estatístico do Processo
FDT	Fechamento do Teste
GDD	Gestão de Defeitos
GDF	Gestão de Ferramentas
GDQ	Garantia da Qualidade
GPT	Gerência de Projetos de Teste
GRT	Gerência de Requisitos de Teste
MAT	Medição e Análise de Teste
OGT	Organização do Teste
PET	Projeto e Execução doe Teste
TDA	Teste de Aceitação
TES	Teste Estático
TNF	Teste Não-Funcional
TRE	Treinamento

dois objetivos principais (SOMMERVILLE, 2011):

- Assegurar que o produto cumpre os requisitos elicitados
- Mostrar situações onde o produto se comporta de maneira inesperada, incorreta ou em discordância com os requisitos elicitados

Os conceitos de verificação e validação estão diretamente relacionados com esses objetivos. Através da validação, a equipe se pergunta se está construindo o produto certo, enquanto com a verificação, a pergunta é se o produto está sendo construído corretamente (BOEHM, 1979 apud SOMMERVILLE, 2011).

A Seção 2.3.1 faz uma discussão mais aprofundada sobre o processo de verificação, enquanto a Seção 2.3.2 aborda especificamente sobre o processo de validação. Já a Seção 2.3.3 faz uma discussão sobre as ferramentas de análise estática de código, que costumam ser utilizadas no contexto de verificação.

2.3.1 Verificação

A verificação de software pretende assegurar que os requisitos especificados foram implementados corretamente (SOFTEX, 2016a). O processo ocorre em diferentes fases do ciclo de vida do projeto, ao se verificar artefatos como requisitos, documentos de arquitetura, código-fonte, arquivos de configuração, entre outros. Por fim, o produto final é também verificado para garantir que seus componentes em conjunto cumprem os requisitos funcionais e não funcionais.

Ao se realizar o processo de verificação, alguns benefícios para o projeto podem ser observados, como a melhora em se gerenciar o projeto, a identificação de erros antecipadamente e, por consequência, o menor custo de reparação de erros, a identificação de requisitos que não poderão ser implementados no projeto, entre outros (SOFTEX, 2016a).

Para se verificar artefatos de software, diversas técnicas foram desenvolvidas e aprimoradas ao longo das últimas décadas. Dentre elas, destacam-se os testes e a revisão por pares (SOFTEX, 2016a).

Teste de software, segundo Bourque e Fairley (2014), é o processo de verificar dinamicamente se um programa realiza os comportamentos esperados em um conjunto finito de testes. Deve-se ter em mente que, mesmo em pequenos projetos, testar todas as possibilidades e combinações de comportamentos do produto é impraticável, portanto é preciso priorizar e selecionar o que deve ser testado e de que maneira, além de determinar os resultados considerados satisfatórios em um contexto específico.

Quanto mais complexo um produto de software se torna, maior o seu determinado esforço de teste, ou seja, a quantidade de recursos necessários para se executar os testes.

Nenhum software é igual, consequentemente alguns produtos são mais fáceis de se testar do que outros. Em outras palavras, alguns produtos de software tem maior nível de testabilidade do que outros.

Os testes visam examinar o comportamento do software através de sua execução (JURISTO; MORENO; VEGAS, 2003). Por ser utilizado na execução do sistema, outras etapas do projeto já foram executadas, como elicitação de requisitos e design da arquitetura, por isso, em projetos que se baseiam apenas na execução de testes, defeitos podem ser detectados tardiamente, aumentando consideravelmente o custo de reparação (SOFTEX, 2016a).

Normalmente os testes são implementados pelos próprios desenvolvedores do código-fonte, porém em alguns casos, pode existir uma equipe especializada no planejamento, execução e documentação de testes e resultados (SOMMERVILLE, 2011). Durante o desenvolvimento de testes, três tipos principais podem ser implementados (SOMMERVILLE, 2011): testes unitários, testes de componentes e testes de sistema.

Testes unitários são voltados para unidades individuais de código, como classes ou funções, e visam testar a funcionalidade dessas unidades e se sua execução ocorre conforme esperado (SOMMERVILLE, 2011).

Testes de componentes (também conhecidos como testes de integração) são relacionados à integração entre duas ou mais unidades individuais que interagem entre si para formar um componente. O principal foco é testar a interface do componente e a interação entre as unidades (SOMMERVILLE, 2011).

Testes de sistema são relacionados às integrações entre alguns ou todos os componentes do sistema. O sistema é testado na totalidade e o principal foco é na interação entre os componentes (SOMMERVILLE, 2011).

A revisão por pares é um método estático de verificação, que consiste no exame de um artefato de um projeto por qualquer membro da equipe, exceto o autor do artefato (SOFTEX, 2016a). A avaliação é guiada pela definição de critérios objetivos. Como é um método estático, pode ser executado antes da implementação de código-fonte, ajudando a detectar defeitos com antecedência (SOFTEX, 2016a). Existem diferentes maneiras de se aplicar a revisão por pares. Algumas das técnicas mais conhecidas são os *walkthrough* e as inspeções.

O *walkthrough* consiste em uma reunião de um grupo de no máximo sete pessoas, com responsabilidades bem definidas, para examinar artefatos, identificar defeitos e propor soluções (SOFTEX, 2016a). Para tanto, é necessário realizar o planejamento da atividade, definindo cronograma, local, agenda, responsabilidades dos membros e *checklists* de avaliação (SOFTEX, 2016a).

A inspeção ocorre de maneira semelhante, porém com inspetores individuais avali-

ando artefatos que não desenvolveram, com base em critérios determinados previamente. Dependendo da severidade dos problemas encontrados, os inspetores decidem pela aceitação ou não do artefato (SOFTEX, 2016a).

2.3.2 Validação

A validação de software é um processo crucial para a garantia da qualidade de software, pois visa assegurar que o produto esteja conforme as necessidades do usuário (SOFTEX, 2016a). O processo pode ocorrer no início do ciclo de vida do produto, quando o usuário pode validar os requisitos para confirmar que atendem suas necessidades, ou ao final do ciclo de vida, quando o produto já está prestes a se tornar operacional. Dentre as diversas técnicas utilizadas para validar um produto de software, destacam-se prototipação, testes alfa, testes beta e testes de aceitação (SOFTEX, 2016a).

O protótipo é uma versão inicial do sistema, utilizado para testar conceitos, apresentar opções do projeto e validar requisitos (SOMMERVILLE, 2011). Através do protótipo, os usuários podem detectar erros de interpretação nos requisitos ou na implementação (SOMMERVILLE, 2011).

Os testes alfa são estratégias de teste de usuário, onde os usuários auxiliam ativamente na identificação de defeitos e os reportam aos desenvolvedores (SOMMERVILLE, 2011). Normalmente, os desenvolvedores trabalham com base nos requisitos, mas não possuem detalhes que importam no uso prático do sistema, então as informações fornecidas pelos usuários são de grande valor (SOMMERVILLE, 2011).

Testes beta são tipos de teste de usuário que ocorrem quando uma nova versão de um produto de software, por vezes inacabado, é liberado para um grupo específico de usuários, ou para qualquer pessoa que tenha interesse em utilizar o sistema (SOMMERVILLE, 2011). Costuma ser aplicado em sistemas utilizados em diferentes ambientes, ao contrário de sistemas encomendados, já que os desenvolvedores não conseguem replicar todos os ambientes possíveis que o produto será executado (SOMMERVILLE, 2011). As informações reportadas pelos usuários que utilizam a versão beta do produto são de grande importância para identificar defeitos que seriam difíceis de detectar com outras abordagens de teste (SOMMERVILLE, 2011).

O teste de aceitação permite ao usuário final executar as funções do sistema para determinar se a implementação deve ser aceita em operação ou não (SOFTEX, 2016a). Sommerville (2011) apresenta seis estágios do teste de aceitação (Figura 10).

Inicialmente deve-se estabelecer os critérios de aceitação, através do consenso entre desenvolvedores e usuários. Em seguida ocorre a etapa de planejamento de testes, onde são estabelecidos os recursos, orçamento e cronograma necessários. Logo após o planejamento, os testes de aceitação são projetados e implementados considerando as características

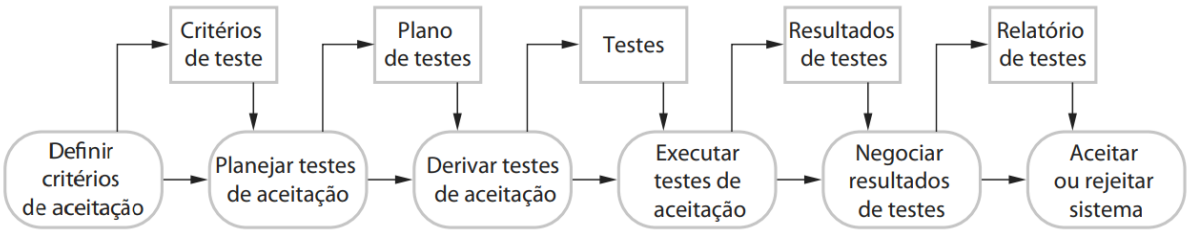


Figura 10 – Estágios do processo de teste de aceitação. Fonte: (SOMMERVILLE, 2011)

funcionais e não-funcionais do sistema, cobrindo os critérios mais importantes. Os testes de aceitação são executados em um ambiente parecido com o ambiente real que o sistema é utilizado. Com os resultados dos testes, são realizadas negociações para discutir os problemas encontrados. Por fim, os usuários devem determinar se o sistema tem condições de ser liberado para uso ou se os problemas identificados são severos demais. Em caso de rejeição, uma nova sessão de testes de aceitação deve ser realizada, reiniciando o ciclo de estágios do processo.

2.3.3 Ferramentas de Análise Estática de Código

A análise estática de código visa aprimorar os atributos de qualidade interna do produto de software e encontrar problemas antes de executá-lo (MARCILIO et al., 2019). No desenvolvimento de software é comum a utilização de ferramentas que facilitam esse processo, realizando uma porção de atividades auxiliares, tais como detecção de vulnerabilidades de segurança, gargalos de desempenho e trechos com más práticas de programação, conhecidos como *code smells* (MARCILIO et al., 2019).

O SonarQube (SonarSource S.A., 2022) é uma das muitas ferramentas disponíveis atualmente, e é utilizada na organização que foi realizado o estudo de caso deste trabalho. A ferramenta é utilizada no contexto de verificação, pois é possível encontrar defeitos na implementação do código-fonte.

O Sonar pode ser configurado através da *pipeline* para executar a análise a cada *commit*, *push* ou *pull request*, ou ainda em situações específicas a serem determinadas pelo desenvolvedor.

O projeto alvo utiliza o SonarQube em um dos passos do pipeline de integração contínua, onde está configurado para executar a cada novo *commit*. Caso o resultado da análise do Sonar seja negativo, ou seja, após a análise, é possível obter os resultados através da interface gráfica do Sonar ou a partir da API, onde uma requisição HTTP é realizada a um *endpoint* e a resposta é realizada no formato JSON. Neste trabalho as métricas do sonar serão obtidas através da requisição HTTP, utilizando como base diferentes *commits* na *branch* principal.

A NDepend (NDepend, 2022) é uma ferramenta de análise estática de código focada no *framework* .NET. Possui funcionalidades como destaque de *code smells* à medida que são introduzidos em tempo real durante o desenvolvimento, integração com serviços de integração contínua, personalização de *quality gates* para padronização de código, entre outras.

Outra ferramenta de análise estática de código complexa é a Axivion Suite (Axivion, 2022), com ênfase em sistemas embarcados. Possui verificação de conformidade com regulamentos atuais da MISRA (*Motor Industry Software Reliability Association*), CERT/CC (*Computer Emergency Response Team Coordination Center*), ISO/IEC, entre outras. Além disso, a ferramenta realiza análise do fluxo de controle do software, possíveis condições de corrida, *code smells*, código duplicado, código inalcançável, integração com serviços de integração contínua, entre outras funcionalidades.

3 Avaliação do Processo de Verificação e Validação

Neste capítulo é realizada a análise informal do processo de Verificação e Validação (VV) do projeto selecionado do Orgão X. A Seção 3.1 contextualiza o projeto onde a avaliação ocorreu, abordando o projeto a ser analisado, a forma que a equipe trabalha e uma visão geral sobre o projeto. A Seção 3.2 apresenta o modelo utilizado para realizar a avaliação, suas adaptações ao trabalho atual e as justificativas de sua escolha. A Seção 3.3 realiza a análise dos resultados esperados do processo de VV, além de pontuar pontos fracos identificados. A Seção 3.4 aborda a análise do nível G de capacidade do processo, definido pelo MPS.BR (SOFTEX, 2021a). Finalmente, a Seção 3.5 faz um resumo do que foi analisado e tira conclusões a respeito do processo de VV do projeto, em relação ao modelo utilizado.

3.1 Visão geral do Órgão X

A seguir são apresentados os principais resultados obtidos após a execução do planejamento apresentado na Seção 1.4. Os dados a seguir foram obtidos através de uma **entrevista**, realizada no dia 11 de julho de 2022, com o gerente do projeto escolhido. A entrevista foi gravada com consentimento explícito do indivíduo, e em seguida seu conteúdo foi transcrito.

O Órgão X é uma organização pública com sede em Brasília e lida com questões financeiras e orçamentárias, possuindo interações com o Governo Federal. Sua estrutura organizacional consiste em diversas secretarias direcionadas a assuntos específicos como, por exemplo, a secretaria de gestão de pessoas, de gestão de processos ou de soluções de tecnologia da informação. Essa última, cuja sigla é STI, também subdivide-se em equipes diferentes, cada uma lidando com diferentes projetos simultaneamente. Uma dessas equipes está atualmente trabalhando no projeto alvo desse trabalho, o qual é melhor descrito na Seção 3.1.1 a seguir.

3.1.1 Contexto do projeto

O projeto escolhido para estudo de caso se trata do *frontend* (interface de usuário) de um sistema de gerenciamento de documentos e processos internos do órgão, utilizado principalmente por uma das secretarias voltadas para esses procedimentos. O sistema possui uma arquitetura de microsserviço, ou seja, pequenos serviços independentes e autô-

nomos que trocam dados entre si e que podem ser testados individualmente (THÖNES, 2015). Para o escopo deste trabalho, apenas o *frontend* do sistema será analisado por estar mais próximo do usuário final.

O *frontend* é a principal forma de interação do usuário com o sistema, porém não é a única. Existem outras ferramentas, como APIs, que permitem o acesso direto aos dados de interesse, dependendo apenas da autorização associada ao usuário. Nem todos os dados do sistema estão disponíveis para todos os funcionários do órgão.

O *frontend* é desenvolvido principalmente utilizando a linguagem de programação TypeScript com a biblioteca React, porém possui trechos de código em menores proporções escritos em Kotlin, Javascript, HTML e CSS. Interage principalmente com dois microsserviços: um que fornece interfaces REST com dados utilizados eventualmente, como serviços de autenticação, e outro que expõe uma interface de comunicação para dados em tempo real, como informações dos processos editados constantemente. A maioria dos dados exibidos pelo *frontend* são requisitados de outros sistemas, o *frontend* apenas faz a integração dos dados e sua apresentação de maneira agradável ao usuário final.

O conjunto do *frontend* e os serviços que interage é a terceira versão de um grande projeto que continua sendo mantido depois de quase vinte anos pela Órgão X. Uma das principais preocupações atuais é garantir uma boa manutenibilidade para o sistema, uma vez que a expectativa é que ele deve ser mantido por, no mínimo, dez anos.

Na Seção 3.1.2 é feita uma discussão acerca do processo de desenvolvimento do projeto, pontuando questões sobre atividades comuns da equipe e algumas das interações entre os membros.

3.1.2 Metodologia de desenvolvimento

O processo de desenvolvimento do *frontend* começa com a elicitação de requisitos dos usuários finais, funcionários das secretarias que trabalham com processos, documentos e sessões. São organizadas reuniões virtuais nas quais os usuários fornecem sugestões, reclamações e oportunidades de melhorias a representantes das equipes de UX (*User Experience*) e de desenvolvimento. Com os requisitos em mãos, a equipe de UX prepara protótipos de alta fidelidade para serem validados pelos usuários. Após os devidos ajustes e a aprovação dos protótipos, eles são encaminhados à equipe de desenvolvimento, que realiza a implementação e teste dos protótipos, além de validá-los e homologá-los nos ambientes específicos. Finalmente, com as novas funcionalidades aprovadas, uma nova versão do produto de software é liberada para acesso de todos os usuários. Ressalta-se que as equipes de UX e de desenvolvimento são equipes diferentes, porém trabalham em conjunto no projeto do *frontend* do sistema.

As funcionalidades a serem implementadas após aprovação no protótipo de alta

fidelidade costumam ser divididas em mais de uma *merge request* para facilitar o processo de revisão de código. Cada tarefa pode ser desempenhada por um único desenvolvedor ou pode acontecer programação por pares, a depender da complexidade da tarefa.

A interação dos usuários do sistema com a equipe de desenvolvimento é realizada por e-mail e ferramentas de comunicação, como *Microsoft Teams*, e discutidos assuntos como novas demandas, problemas encontrados e sugestões de funcionalidades. Dependendo da necessidade, são organizados testes de usabilidade com os usuários (considerados como testes de aceitação pela equipe) visando validar novas funcionalidades utilizando protótipos de alta fidelidade. Os resultados obtidos nesses testes são então armazenados e analisados posteriormente e, caso tenham resultados satisfatórios, o protótipo é então aprovado para seguir para sua implementação de fato no sistema principal.

A equipe de desenvolvimento aplica conceitos de *Scrum* e *Extreme Programming*, como ciclos de trabalho baseados em *sprints*, reuniões diárias rápidas onde todos os membros são incentivados a participar, padronização de código, programação em pares e integração contínua.

Sobre o processo de testes, a equipe tenta entregar cada novo trecho de código em conjunto com seu devido teste. Não há uma equipe especializada em testes ou qualidade de software, então os testes são de total responsabilidade do desenvolvedor. Existem em maior parte testes unitários e testes de componentes React, além de arquivos menores de configuração. Também são utilizados testes denominados testes de contrato, onde são simuladas requisições HTTP REST utilizando arquivos JSON com entradas e saídas de dados bem definidas. Uma pequena quantidade de testes end-to-end no sistema também foi relatada.

Como o projeto implementa a arquitetura de microsserviços, é possível que um ou outro microsserviço que esteja instável possa afetar diretamente a qualidade em uso do usuário final. Mesmo que o *frontend* tenha boa testabilidade, a qualidade em uso seria afetada negativamente nesse cenário.

Por se tratar de um sistema ainda em estágios iniciais de desenvolvimento, algumas funcionalidades passam por constantes mudanças e refatorações visando satisfazer as necessidades dos usuários. Essas mudanças podem tornar os resultados de alguns testes obsoletos, em especial os testes que não são automatizados.

O processo de desenvolvimento da equipe não segue um padrão organizacional, já que não existe um padrão entre as equipes diferentes da STI. Diferentes equipes podem ter diferentes processos, documentos e artefatos produzidos. Boa parte da documentação produzida se dá pelo uso da plataforma GitLab, onde são geradas *issues*, *branches* e *merge requests*.

Apesar de o processo de desenvolvimento não ter uma documentação formalizada,

no projeto existe um documento chamado ADR - *Architectural Decisions Record* (Registro de Decisões Arquiteturais). O intuito do ADR é documentar quaisquer mudanças na arquitetura do projeto que possam ter impactos significativos. Por exemplo, ao substituir o banco de dados por outro serviço com mais desempenho, ou ao se utilizar uma nova biblioteca integrada ao React para padronizar componentes, essas decisões devem ser registradas e justificadas, além de serem notificadas à equipe de desenvolvimento.

Na raiz do projeto também existe um repositório com guias, tanto para o *frontend* quanto para os sistemas que ele interage. Nesses guias são descritos procedimentos básicos como testes de *containers* e testes de estados. Entretanto, alguns trechos da documentação estão em branco e não há previsão de quando serão implementados.

Na Seção 3.1.3 é analisado o uso de métricas por parte da equipe, e de que forma essa atividade pode influenciar no processo de verificação e validação.

3.1.3 Uso de métricas no projeto

Após uma entrevista com um dos responsáveis pelo projeto, foram obtidas informações importantes sobre o sistema. Dentre elas, notou-se que as métricas de software que estão disponíveis são coletadas através do SonarQube e são pouco utilizadas no dia a dia do desenvolvimento. Para a equipe, o *feedback* rápido de ferramentas como *linters* é mais interessante. Percebeu-se que existem reclamações frequentes sobre o tempo necessário para o SonarQube realizar a análise estática do código e apresentar o relatório das métricas, sendo esse um dos principais motivos para não se utilizar as métricas no projeto.

Observou-se que não existem métricas formalizadas de qualidade em uso no sistema, porém os testes realizados com os usuários são documentados. Apesar da falta de métricas, percebe-se que a testabilidade do sistema pode estar diretamente relacionada com a percepção de qualidade do usuário. Essa afirmação foi feita pelo profissional responsável pelo design de UX do projeto, baseado em sua experiência. O membro da equipe de UX relata que usuários diferentes podem ter interesses e necessidades diferentes, o que pode ser difícil de se testar e validar. Da mesma forma, uma funcionalidade que é fácil de se testar com usuários diferentes pode ser bem recebida pelos mesmos, sugerindo que sua percepção de qualidade é boa.

3.2 Modelo de avaliação utilizado

Como visto no Capítulo 2, existem diferentes modelos que podem ser utilizados para a avaliação dos processos de software de uma organização. Considerando as opções disponíveis, contexto do projeto e as próprias características do TCC, optou-se por realizar

uma adaptação entre o modelo CMMI-DEV e o MPS.BR, semelhantemente à estratégia abordada por Xoteslem (2021).

Os conceitos de representação contínua do CMMI-DEV, discutidos na Seção 2.2.1, serão utilizados para se avaliar um nível de capacidade específico do MPS.BR. O MPS.BR foi escolhido ao invés do MPT.BR como uma das bases para a avaliação do processo de teste de software por uma série de motivos. Primeiramente, o modelo MPS.BR é o mais recente, tendo sua última versão lançada em janeiro de 2021 (SOFTEX, 2021a), enquanto a última versão do MPT.BR é de 2011 (SOFTEXRECIFE, 2011), dez anos mais desatualizado.

Uma planilha de indicadores elaborada em 2021 pelo MPS.BR para auxiliar na avaliação de processos foi disponibilizada na internet de maneira gratuita. Já o MPT.BR não disponibiliza nenhum tipo de suporte público para a avaliação de processos.

O foco inicial do trabalho era avaliar apenas o processo de teste de software do projeto. Porém, decidiu-se expandir o escopo da avaliação para englobar o processo de verificação e validação. O MPS.BR possui um processo chamado Verificação e Validação (VV) que possui apenas cinco resultados esperados, tornando sua avaliação bem mais simplificada se comparado às diversas áreas de processo do modelo MPT.BR. Além disso, não é necessário realizar nenhum tipo de priorização ou questionário sobre qual processo do MPT.BR utilizar, já que o processo VV atende todas as necessidades do trabalho. Por outro lado, tais estratégias seriam necessárias ao se utilizar o modelo MPT.BR, ato que consumiria uma parte considerável do tempo disponível para o desenvolvimento do trabalho.

Finalmente, considerando o curto período para finalizar o TCC, seria impraticável utilizar o MPT.BR nessa situação. Feitas essas observações, a estratégia de utilizar a representação contínua do CMMI-DEV com o processo de VV do modelo MPS.BR foi a escolha mais plausível para esse trabalho. O processo de VV será analisado considerando o nível G de capacidade do processo, apesar do modelo MPS.BR classificá-lo como um processo de nível D de capacidade.

Deve-se observar que a versão mais recente do guia geral do MPS.BR (janeiro de 2021) define o processo de Verificação e Validação como um único processo. Entretanto, as versões anteriores do modelo definiam os processos separadamente.

O MA-MPS indica que devem ser selecionados, no mínimo, dois projetos para avaliações nível G. Entretanto, para este trabalho será realizada uma adaptação, pois apenas um único projeto será analisado. Essa decisão foi tomada com base no prazo para realizar a avaliação, já que seria difícil analisar dois projetos diferentes com o tempo disponível.

Apesar de existirem duas áreas de processo no CMMI-DEV que são de interesse do

trabalho (Verificação e Validação), elas não foram escolhidas como método principal de análise do processo de teste. A abordagem de representação por estágios não é aplicável ao escopo do trabalho, pois o interesse é apenas em processos relacionados a teste de software, e não em processos gerenciais, organizacionais ou relacionados a outras fases do ciclo de vida do software.

Após a coleta de dados na planilha de indicadores, foram obtidos indicadores diretos (IND), através da análise do repositório e da documentação existente, e afirmações (AFR), utilizando como base as entrevistas realizadas. O termo "indicador indireto" não é mais utilizado na versão mais recente do MPS.BR, por isso não será utilizado nesta avaliação.

Cada indicador possui uma ou mais imagens que comprovem sua existência, retirados do projeto do Órgão X. Entretanto, termos sensíveis que possam identificar o órgão, projeto e membros foram obstruídos propositalmente com tarjas pretas, visando manter o sigilo.

3.3 Resultados esperados do processo Verificação e Validação - VV

O processo de Verificação e Validação visa "confirmar que os produtos de trabalho selecionados atendem aos requisitos especificados, pela execução de testes e revisão por pares, e que um produto ou componente do produto atenderá a seu uso pretendido quando colocado no ambiente operacional" (SOFTEX, 2021a).

Em versões anteriores a 2021, o MPS.BR definia o processo de Verificação e o processo de Validação de formas separadas. Sabendo disso, neste trabalho serão explicitadas os indicadores e afirmações diretamente relacionados com verificação ou validação, considerando que os dois processos foram avaliados separadamente. O processo de VV possui cinco resultados esperados, sendo descritos a seguir.

3.3.1 Resultado esperado VV1

> Produtos de trabalho a serem verificados e validados são selecionados (SOFTEX, 2021a).

Para atender esse resultado esperado, são necessárias evidências que comprovem que existe uma forma de selecionar produtos de trabalho a serem verificados e validados no projeto. Foram obtidos os seguintes indicadores:

- IND - Verificação: O documento "Frontend - Testes.md" especifica, além de outros detalhes, quais elementos devem ser testados: *UI components, container components,*

hooks, reducers, selectors e *utils* (Figura 11). O documento é uma evidência direta de que alguns elementos do código-fonte devem ser verificados.

- IND - Verificação: O documento "BFF - Testes.md" especifica, além de outros detalhes, quais elementos devem ser testados: *Controllers*, APIs e funções de suporte (Figura 12).
- AFR - Verificação: Através da entrevista realizada no dia 11 de julho de 2022, foi confirmado que os principais elementos da biblioteca React, base do *frontend*, devem ser testados.
- AFR - Validação: Os usuários finais, membros das equipes de desenvolvimento e de UX periodicamente realizam reuniões virtuais onde elicitam requisitos e priorizam funcionalidades (Figura 13). Apesar de, frequentemente, as reuniões serem realizadas semanalmente, podem ocorrer reuniões extraordinárias para tratar de assuntos urgentes.
- IND - Validação: As funcionalidades priorizadas pelos usuários finais são registradas na plataforma Miro na forma de pequenos blocos de anotações (Figura 14).

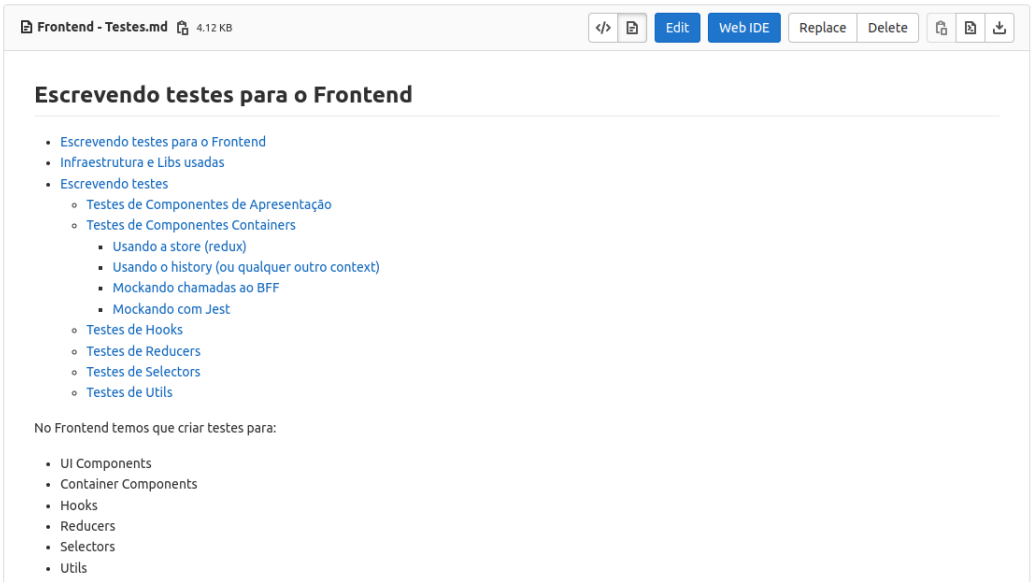


Figura 11 – Documento "Frontend — Testes.md"

Segundo o MPS.BR, os principais produtos de trabalho são normalmente selecionados para atividades de validação, como o plano do projeto, o documento de requisitos, o documento de análise, o documento de projeto e o código-fonte (SOFTEX, 2016a).

Devido ao *frontend* do projeto possuir pouca documentação, o principal produto de trabalho a ser verificado é o próprio código-fonte do projeto. Não foram encontradas evidências em relação ao processo de verificação de requisitos, artefatos de arquitetura

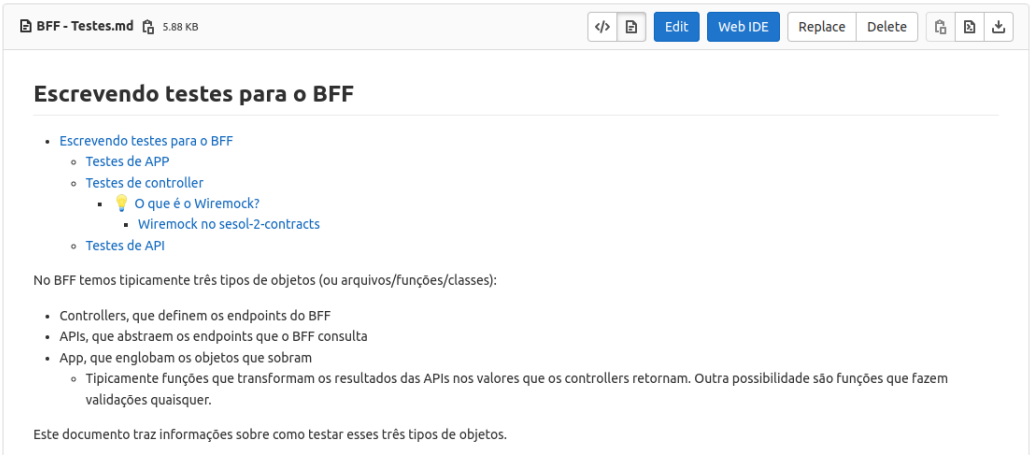


Figura 12 – Documento "BFF — Testes.md"

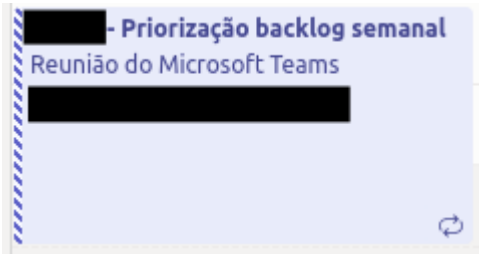


Figura 13 – Registro de reunião semanal na plataforma Microsoft Teams do projeto

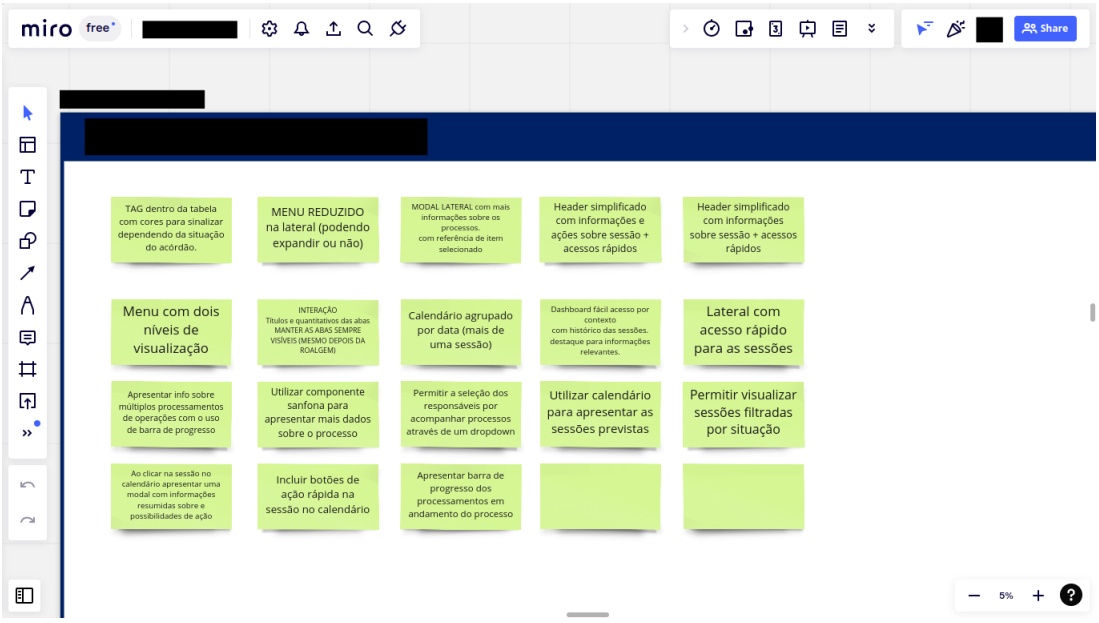


Figura 14 – Exemplos de funcionalidades priorizadas pelos usuários

e de projeto. Consequentemente, não foram encontradas evidências de um procedimento formal para selecionar produtos a serem verificados.

Os principais produtos de trabalho a serem validados são os requisitos e a interface de usuário, através dos testes de usabilidade. Não foram encontrados critérios de seleção de produtos de trabalho a serem validados. Também não foram encontrados guias e documentos ao nível organizacional orientando sobre produtos de trabalho que devem ser validados. Os pontos fracos identificados no resultado esperado VV1 estão resumidos na Tabela 7.

Tabela 7 – Pontos fracos identificados no resultado esperado VV1

PF	**Descrição do ponto fraco**
PF1	Falta de critérios de seleção de produtos de trabalho a serem verificados e validados

Considerando os graus de implementação possíveis de se atribuir a um resultado esperado de um processo, conforme a Tabela 4, os indicadores presentes são considerados inadequados e pontos fracos significativos foram observados. Portanto, o resultado esperado VV1 é classificado como **parcialmente implementado**.

3.3.2 Resultado esperado VV2

> Procedimentos e material de apoio são definidos, mantidos atualizados e usados para preparação e realização de revisões por pares (SOFTEX, 2021a).

Para esse resultado, são necessárias evidências de que a revisão por pares é planejada e realizada, além de evidências de materiais de apoio para realização da tarefa. Foram encontradas os seguintes indicadores:

- IND — Verificação: Toda modificação feita na base de código do projeto é realizada em *branches* separadas da *branch* principal, e então as *branches* são mescladas no processo de *merge request*. A revisão de código ocorre nesse processo (Figuras 15, 16, 17, 18 e 19).
- AFR — Verificação: Através da entrevista realizada no dia 11 de julho de 2022, ficou claro que a principal forma de assegurar a qualidade do produto é através da revisão de código em *merge requests*. O código que cada membro implementa deve ser revisado por outro membro diferente do autor. As métricas de qualidade do produto não são muito utilizadas pela equipe.
- IND — Verificação: Na atividade de revisão de código, os papeis definidos são desenvolvedor e revisor (Figuras 16 e 18). Um membro da equipe pode ser desenvolvedor

de um trecho de código e revisor de outro, mas nunca os dois papéis simultaneamente, ou seja, um desenvolvedor não pode revisar o próprio código.

- AFR — Verificação: O processo de revisão de código possui uma documentação básica sobre como preparar e realizar a atividade e é um processo altamente difundido e realizado pela equipe. Novos membros são encorajados a estudar a documentação do processo, porém também dependem de experiências passadas com desenvolvimento de software e da interação com membros antigos para entender o processo (Figuras 20 e 21).

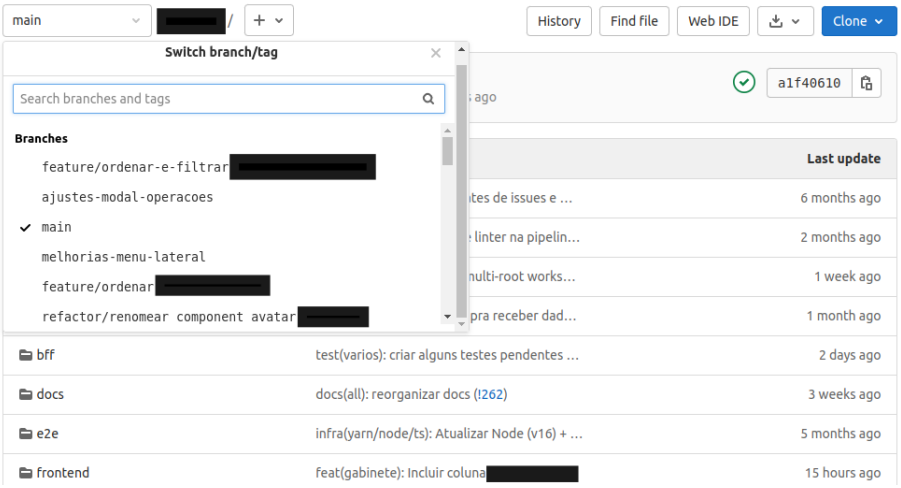


Figura 15 – Exemplos de branches do projeto

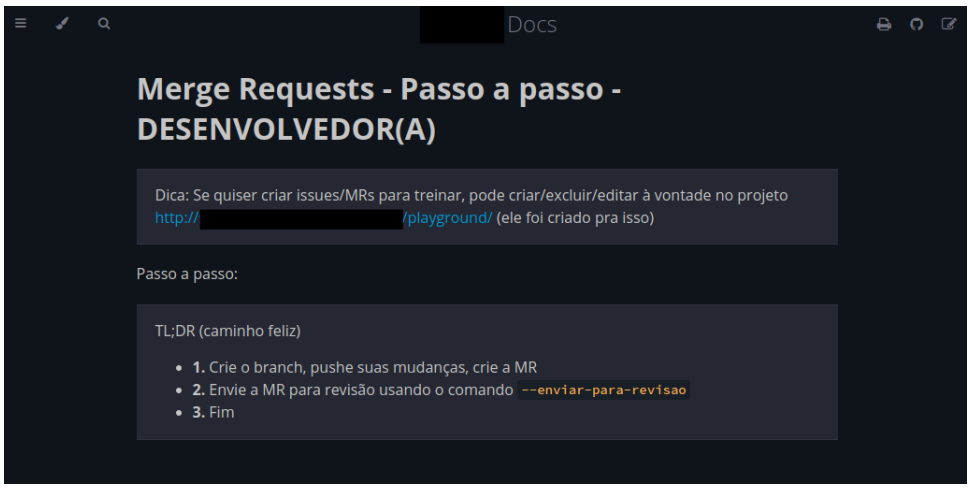


Figura 16 – Versão resumida do processo de criação de merge request

Não foi encontrado nenhum tipo de cronograma ou documento planejando os recursos necessários para executar o processo de verificação. Da mesma forma, não foram encontradas evidências do uso de materiais ou ferramentas com o objetivo de planejar atividades de revisão por pares.

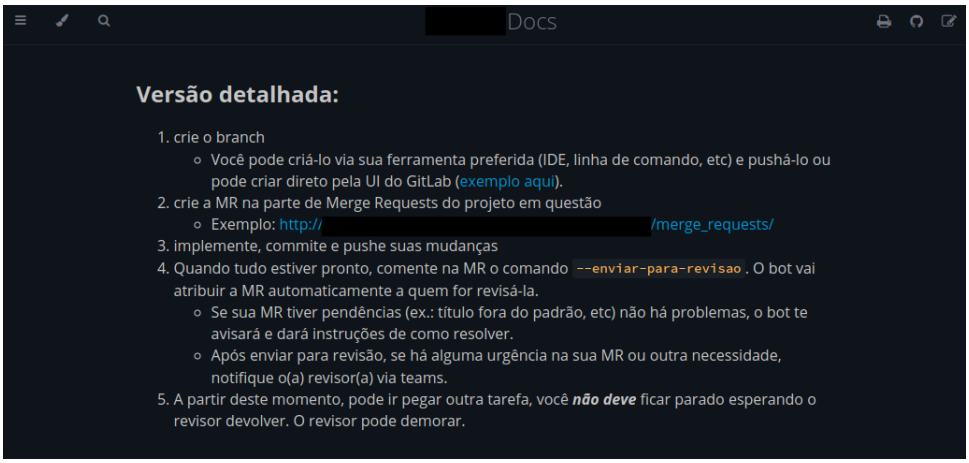


Figura 17 – Versão detalhada do processo de criação de merge request

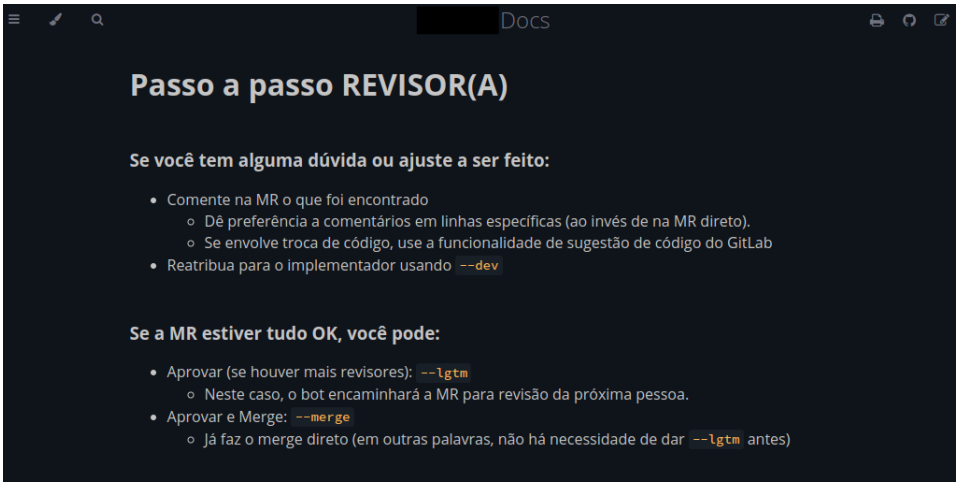


Figura 18 – Guia para revisão de merge request

Figura 19 – Exemplo de merge request no projeto

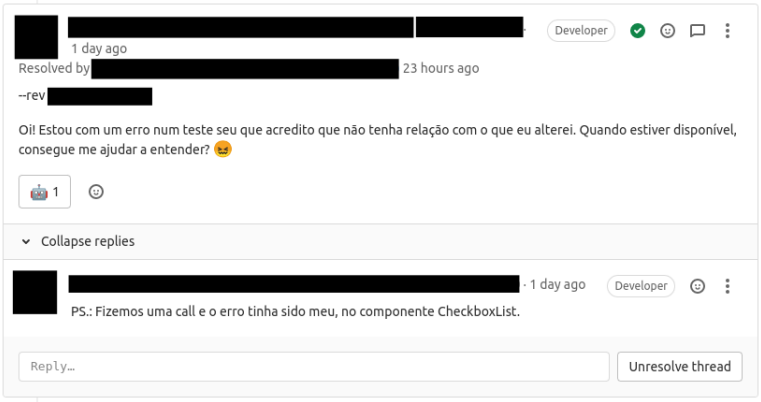


Figura 20 – Exemplo 1 de interação de membros da equipe durante merge requests

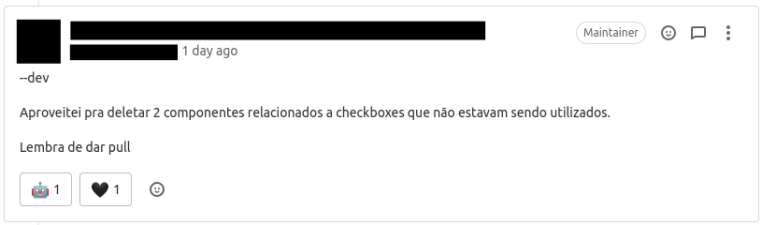


Figura 21 – Exemplo 2 de interação de membros da equipe durante merge requests

O processo de validação se baseia fortemente no uso de testes de usabilidade, planejado no documento "Teste de usabilidade — Tarefas.xlsx".

Foram encontradas evidências de que a revisão por pares ocorre com o intuito de revisar o código-fonte. Porém, não foram encontradas evidências da revisão por pares em outros artefatos, como no plano do projeto, documento de requisitos, documento de arquitetura e arquivos de configuração de software. Os pontos fracos identificados no resultado esperado VV2 estão resumidos na Tabela 8.

Tabela 8 – Pontos fracos identificados no resultado esperado VV2

PF	Descrição do ponto fraco
PF2	Falta de planejamento de cronograma e recursos necessários para execução da revisão por pares
PF3	Falta da realização de revisão por pares em outros produtos de trabalho além do código-fonte

Utilizando os critérios apresentados na Tabela 4, as evidências e indicadores são considerados adequados e pontos fracos significativos foram observados. Portanto, o resultado esperado VV2 é classificado como **parcialmente implementado**.

3.3.3 Resultado esperado VV3

Métodos, procedimentos, critérios e ambientes são definidos, mantidos atualizados e usados durante as atividades de teste com fins de verifica-

ção e validação (SOFTEX, 2021a).

Para esse resultado, é necessário obter indicadores de que existem ferramentas, procedimentos, documentos e ambientes para auxiliar as atividades de teste, verificação e validação. Foram obtidos os seguintes indicadores:

- IND — Verificação: A existência de uma etapa no pipeline específica para executar os testes do sistema. A Figura 22 ilustra a pipeline do projeto que é executada após cada *commit* realizado. Uma observação importante é que, apesar de existir o ambiente de ACEITE (ou homologação), ele não chega a ser utilizado pela equipe. O principal foco é apenas nos ambientes de DESENVOL e PRODUÇÃO.
- IND — Verificação: O documento "Testes de contrato" apresenta uma introdução sobre o conceito, ferramentas utilizadas no projeto e uma série de boas práticas para realização da tarefa (Figura 23).
- IND - Validação: O documento "Teste de usabilidade - Tarefas.xlsx" possui critérios de sucesso ou falha para cada uma das tarefas, ou seja, dependendo do desempenho do usuário ao executar uma tarefa específica, é possível classificar a tarefa como realizada com sucesso ou não (Figura 24).
- IND — Validação: A utilização de protótipos de alta fidelidade visando validar novas funcionalidades ou alterações significativas em funcionalidades existentes antes que as mesmas sejam implementadas (Figura 27).
- AFR - Validação: São realizadas reuniões periódicas visando entender as demandas do usuário. As reuniões entre usuários, membros da equipe de UX e da equipe de desenvolvimento são agendadas de maneira improvisada, conforme a necessidade. Não existe um cronograma predeterminado.
- AFR - Validação: Técnicas como heurísticas de Nielsen e ferramentas como HotJar e Google Analytics são utilizadas para análise de usabilidade e compreensão da interação do usuário com o sistema.
- IND - Validação: O documento "Teste de usabilidade - Tarefas.xlsx" apresenta um roteiro das principais tarefas a serem executadas durante o teste de usabilidade (Figura 24). O teste de usabilidade pode ser considerado como teste de aceitação no contexto do projeto.

O ambiente de homologação (referenciado no projeto como ACEITE) não é utilizado para verificar o código-fonte. Entretanto, considerando que o processo de verificação ocorre na maioria no ambiente de desenvolvimento (referenciado no projeto como DESENVOL), isso não é considerado um ponto fraco significativo.

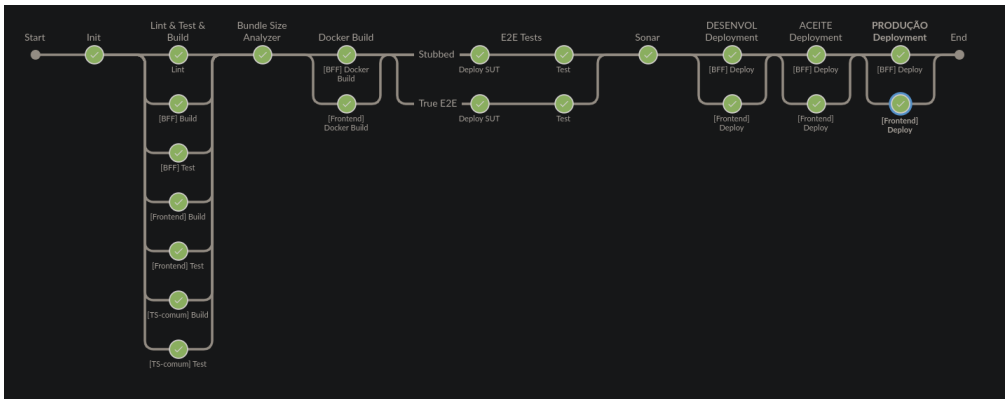


Figura 22 – Pipeline do projeto no Órgão X.

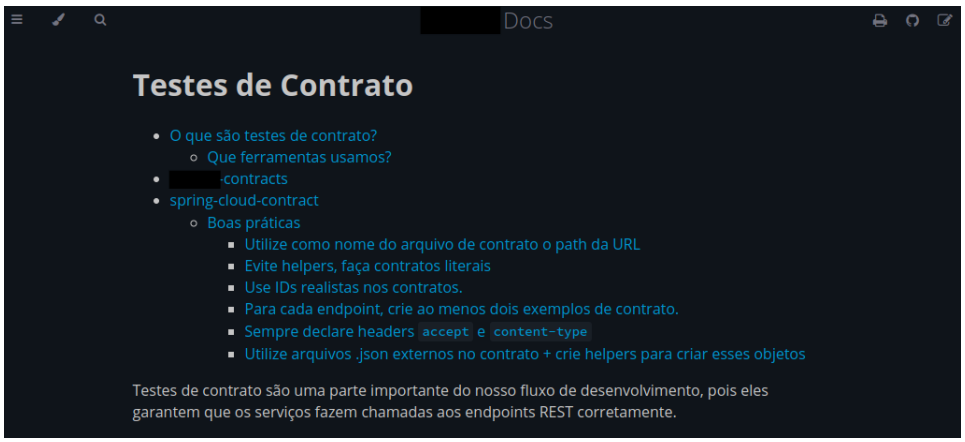


Figura 23 – Sumário do documento “Testes de contrato”

#	Tarefas	Cenários	Como saberemos que deu certo?	Como saberemos que deu errado?
1	Selecionar sessões de outras datas e colegiados. Localizar as informações que identificam a sessão e sua situação atual. Verificar a disponibilidade de documentos dos processos pautados e seus autores. Iniciar a sessão.	Como integrante da equipe da Secretaria das Sessões, você quer garantir que a Sessão [redacted] de hoje (15 Junho) seja iniciada. De acordo com o [redacted], a referida sessão está em qual status? Antes de iniciar a sessão, você poderia identificar quais processos possuem documentos ainda não enviados pelos [redacted]? Inicie a sessão.	O usuário deve ser capaz de navegar pelo calendário e selecionar a sessão precisamente. O usuário deve ser capaz de visualizar as informações e tag da situação sobre a sessão no cabeçalho ou informações contextuais no botão de ação da sessão. O usuário identificará a disponibilidade de documentos no processo. O usuário iniciará a sessão clicando no botão Iniciar Sessão.	O usuário terá dificuldade em navegar pelo calendário e identificar a sessão correta. O usuário apresentará dificuldades para encontrar as informações em tela ou não encontrará. O usuário terá dificuldade em visualizar a diferença entre os documentos disponíveis e os não disponíveis, e identificar os nomes dos autores.
2	Visualizar quem da [redacted] está acompanhando o processo. Registrar que está acompanhando um processo.	Você iniciou a sessão e agora você e seus colegas conseguem sinalizar que estão acompanhando determinados processos da sessão. Você consegue identificar quem está acompanhando o processo [redacted] Registre no [redacted] que você vai acompanhar o processo [redacted]	O usuário entenderá o conceito "Acompanhar processo" e selecionará o ícone de acompanhar processo	O usuário não conseguirá identificar o nome dos colegas que estão acompanhando os processos. O usuário terá dificuldade em entender o significado do ícone e não conseguirá executar a ação.

Figura 24 – Exemplo de planejamento de tarefas a serem executadas no teste de usabilidade

O projeto não possui casos de testes, e os testes são implementados baseando-se na experiência do desenvolvedor. A falta da padronização de como testar as funcionalidades pode gerar inconsistências no sistema, com componentes muito bem testados e outros nem tanto. Isso é considerado um ponto fraco significativo.

Foi identificado que no documento "Frontend — Testes.md", apesar da especificação de quais elementos do frontend devem ser testados, não existe uma descrição apropriada de como realizar os testes. O próprio documento tem um trecho que diz: "Este documento traz informações sobre como testar essas coisas", porém tais informações não foram descritas, existindo apenas um indicador de TODO (to do — a ser realizado) (Figura 25). Apenas os testes de container components estão descritos de maneira detalhada (Figura 26).

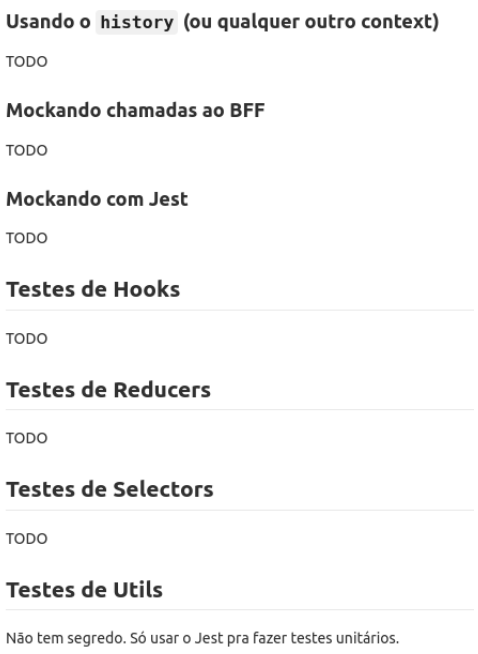


Figura 25 – Trechos do documento "Frontend - Tests.md" sem documentação

Existe um relativo suporte ao processo de validação do frontend do sistema, porém parte da documentação está consideravelmente desatualizada. A última alteração no documento "Teste de usabilidade - Tarefas.xlsx", por exemplo, foi feita em junho de 2021 (mais de 1 ano no momento da análise). Nesse período, as tarefas documentadas podem ter sofrido alterações, e novas funcionalidades podem ter sido adicionadas ao sistema, mas não foram consideradas no documento. Considerando que métodos e procedimentos usados durante as atividades de validação devem ser mantidos atualizados, como descrito pelo resultado esperado VV3, isso é considerado um ponto fraco significativo.

Não foram encontradas evidências da existência de planejamento, métodos, procedimentos e critérios para realização das atividades de verificação e validação em outros

Testes de Componentes Containers

Para renderizar um container para teste, usamos o `renderWithProviders`, que usa o `render` do testing-library/react. Além de fazer tudo o que o `render` faz, o `renderWithProviders` adiciona Providers como wrapper pro container que será renderizado e disponibiliza alguns contextos usados nesses Providers no objeto retornado pra serem usados nos testes.

```
const result = renderWithProviders(<FiltroInstrucoesSessaoPorRelatorContainer />)
```

Se nunca usou o `testing-library/react`, é recomendado que leia a documentação sobre API e Queries.

Como se trata de container, é esperado que tenhamos que lidar com uso da store do Redux, com chamadas ao BFF, com outros contextos, e talvez até mockar hooks ou componentes. Os tópicos abaixo mostram criar testes pra essas situações.

Usando a `store` (redux)

Um container pode acessar a `store` do Redux para ler ou modificar um estado.

Para testar se um container modifica um estado corretamente, é necessário acessar a `store` para checar se tal estado é o esperado. Isso pode ser feito pegando a `store` dentro do objeto que é retornado pelo `renderWithProviders` e usando o selector apropriado pro estado que deseja verificar:

```
describe(Counter, () => {
  it('incrementa contador ao clicar no botão "Add"', () => {
    // setup
    const component = <Counter />
    const result = renderWithProviders(component)
    // execute
    act(() => userEvent.click(result.getByText('Add')))
    // verify
    const counter = selectCounter(result.store.getState())
    expect(counter).toBe(1)
  })
})
```

Exemplo: FiltroColuna[redacted]Container.test.tsx

Figura 26 – Exemplo de descrição de como realizar testes em container components

artefatos além do código-fonte, dos requisitos e da interface de usuário. Os pontos fracos identificados no resultado esperado VV3 estão agrupados na Tabela 9.

Tabela 9 – Pontos fracos identificados no resultado esperado VV3

PF	Descrição do ponto fraco
PF4	Falta de planejamento de testes (casos de teste)
PF5	Falta de descrição de como testar determinados elementos da interface de usuário
PF6	Documentação desatualizada
PF7	Falta do planejamento de VV em outros produtos de trabalho além da interface de usuário e do código-fonte

Utilizando os critérios apresentados na Tabela 4, as evidências e indicadores não são considerados adequados e pontos fracos significativos foram observados. Portanto, o resultado esperado VV3 é classificado como **parcialmente implementado**.

3.3.4 Resultado esperado VV4

> Atividades de verificação e validação são realizadas e problemas identificados são tratados (SOFTEX, 2021a).

Para esse resultado, é necessário obter indicadores de que as atividades de verificação são realizadas e que os defeitos encontrados são registrados e tratados. Além disso, são necessários indicadores de que tarefas de validação são realizadas e os problemas identificados são resolvidos. Foram obtidos os seguintes indicadores:

- IND — Verificação: A existência de uma etapa de testes no *pipeline*, onde os testes automatizados são executados a cada novo *commit*, em qualquer *branch* (Figura 22). Quando pelo menos um teste falha, a *pipeline* notifica o desenvolvedor que deve tentar resolver o problema o mais rápido possível, para evitar introduzir *bugs* no sistema.

- IND — Verificação: Os problemas encontrados são registrados de maneira automatizada através da ferramenta SonarQube (Figura 28). Informações como quantidade e severidade dos *bugs* encontrados ficam disponíveis para a equipe de desenvolvimento.

- IND — Verificação: A utilização da palavra-chave "*fix*" em *commits* e *merge requests* para indicar que se está tratando algum tipo de problema (Figura 19).

- IND — Verificação: A revisão de código (revisão por pares) é executada durante cada *merge request* (Figuras 19, 20 e 21).

- AFR — Verificação: Através da entrevista, foi relatada a grande ênfase em tentar testar toda modificação feita no código do sistema, apesar de nem sempre ser possível testar tudo.

- AFR - Verificação: O projeto possui três tipos de testes para verificar o código-fonte: testes unitários, testes de contrato (termo utilizado pelos membros da equipe para testes de integração) e testes *end-to-end*. A maioria dos testes implementados são unitários, tendo poucos testes de contrato e pouquíssimos testes *end-to-end*. A diferença da quantidade de testes por tipo é esperada devido à complexidade de suas implementações.

- IND - Validação: Os testes de usabilidade são planejados (em parte) com base no documento "Teste de usabilidade - Tarefas.xlsx", onde os critérios de sucesso ou falha das atividades a serem desempenhadas são definidas (Figura 24).

- IND — Validação: Os testes de usabilidade são realizados e suas execuções são gravadas, com o consentimento dos participantes, para fins de análises posteriores (Figura 27).

- AFR — Validação: O protótipo de alta fidelidade é utilizado para validar funcionalidades (Figura 27).

Por não existirem casos de teste, as funcionalidades implementadas são testadas conforme a experiência do desenvolvedor. Por não ter procedimentos formais sobre o que deve ser testado, alguns trechos de código podem ser testados de maneira superficial, o que pode causar defeitos no software a medida que a manutenção do sistema é realizada.

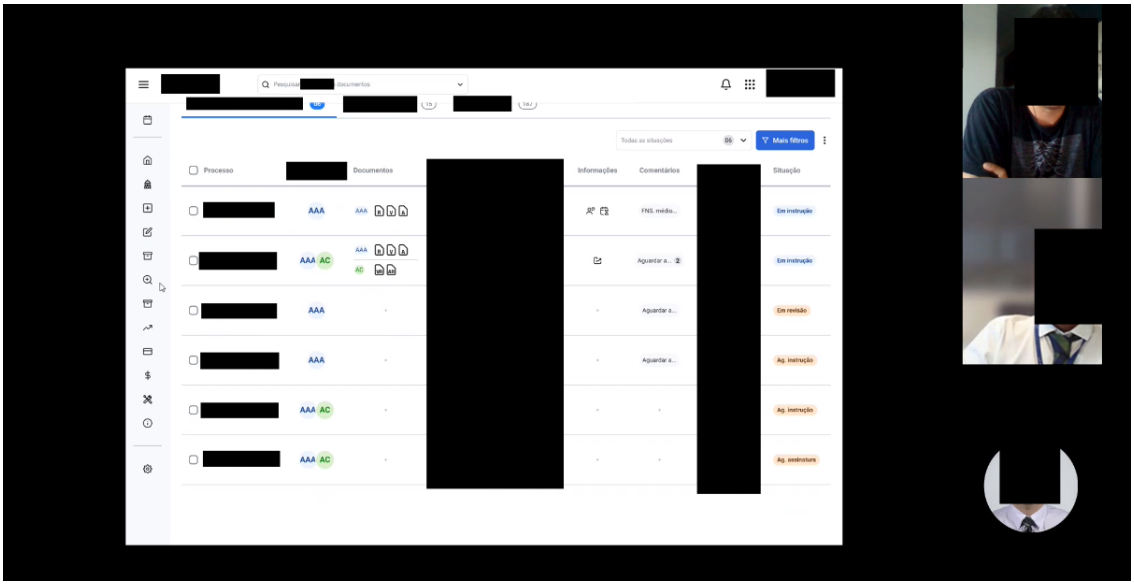


Figura 27 – Gravação da execução de um teste de usabilidade

Os testes automatizados dos componentes do *frontend* são executados na pipeline de integração contínua após cada *commit*. Os indicadores fornecidos pela ferramenta SonarQube, que executa após cada *commit* em uma das etapas da *pipeline*, não são utilizados na prática pela equipe. O gerente do projeto afirmou em entrevista que a equipe se baseia mais na revisão por pares e em ferramentas *linter* do que nos indicadores de análise estática de código para avaliar a qualidade do produto. Isso é considerado um ponto fraco.

Os testes de usabilidade são executados com os usuários quando são necessários. Trata-se de uma chamada de vídeo pela plataforma *Microsoft Teams*, onde os usuários são instruídos a realizar tarefas específicas, conforme as orientações dos membros da equipe de UX.

Não foram encontradas evidências da realização de verificação e validação de outros artefatos além dos requisitos e do código-fonte. Não é possível afirmar se artefatos como plano do projeto e documentos de arquitetura foram ou não verificados e validados. Isso é considerado um ponto fraco significativo. Os pontos fracos identificados no resultado esperado VV4 estão agrupados na Tabela 10.

Tabela 10 – Pontos fracos identificados no resultado esperado VV4

PF	Descrição do ponto fraco
PF8	Falta da execução de VV em outros produtos de trabalho além da interface de usuário e do código-fonte

Utilizando os critérios apresentados na Tabela 4, foram encontradas evidências de que o código-fonte é verificado e as funcionalidades são validadas, porém, não foram

encontradas evidências em relação a outros produtos de trabalho. Portanto, o resultado esperado VV4 é classificado como **parcialmente implementado**.

3.3.5 Resultado esperado VV5

> Os resultados das atividades de verificação e validação são analisados, registrados e comunicados (SOFTEX, 2021a).

Para esse resultado, são necessárias evidências de que os resultados das atividades de verificação e validação são registrados. Por exemplo, os testes que obtiveram sucesso e os que falharam são registrados e analisados pela equipe. Além disso, são necessárias evidências de que os resultados da validação são analisados e utilizados. Foram obtidos os seguintes indicadores:

- IND - Verificação: A existência de uma etapa de testes no *pipeline*. Quando um teste falha após realizar um *commit*, o desenvolvedor é notificado e ele deve resolver o problema o mais rápido possível.
- IND - Verificação: A existência de indicadores obtidos através da análise de código estático, como número de testes unitários, número de testes que falharam e porcentagem de código coberto por testes (Figura 28).
- AFR - Validação: Através da entrevista, foi identificado que os resultados dos testes de usabilidade são gravados para análise posterior (Figura 27).
- IND - Validação: Após análise das gravações de testes de usabilidades, pontos importantes são registrados em um mural da plataforma Miro (Figura 29).

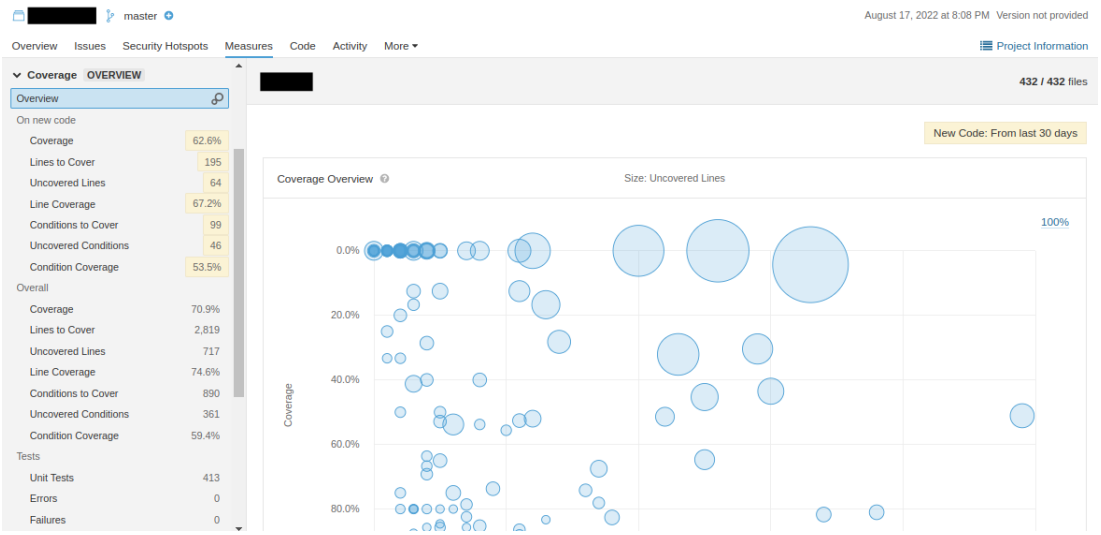


Figura 28 – Exemplos de indicadores obtidos utilizando SonarQube

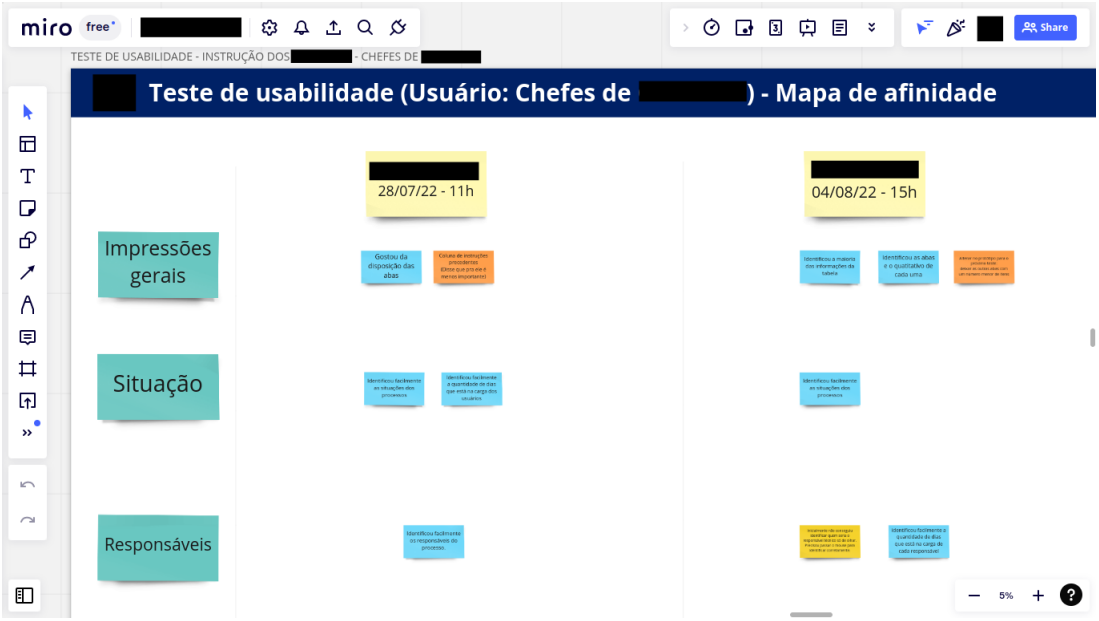


Figura 29 – Exemplos de resultados de testes de usabilidade

Como a verificação e validação de outros artefatos do projeto além de requisitos e código-fonte não é realizada, não existem resultados dessas atividades para serem analisados.

Os resultados dos testes automatizados do *frontend* ficam disponíveis na plataforma SonarQube. Entretanto, os indicadores são pouco utilizados pela equipe. Informações como *code smells*, trechos de código com chances de se tornarem *bugs*. Esse tipo de informação pode facilitar na prevenção de defeitos, porém são ignorados pela equipe. Isso é considerado um ponto fraco.

Os resultados dos testes de usabilidade são gravados e ficam disponíveis para análise posterior da equipe. Apesar da existência da gravação de alguns testes de usabilidade, a maioria das informações é registrada na forma de anotações na plataforma Miro. Os pontos fracos identificados no resultado esperado VV5 estão agrupados na Tabela 11.

Tabela 11 – Pontos fracos identificados no resultado esperado VV5

PF	Descrição do ponto fraco
PF9	Falta da análise de resultados de VV em outros produtos de trabalho além da interface de usuário e do código-fonte
PF10	Desuso de indicadores de testes pela equipe

Utilizando os critérios apresentados na Tabela 4, existem evidências e afirmações de que os resultados dos testes de usabilidade e de testes automatizados são analisados. Entretanto, não existem evidências da análise de resultados de VV de outros artefatos. Portanto, o resultado esperado VV5 é classificado como **parcialmente implementado**.

3.4 Resultados esperados do nível G de capacidade do processo - CP-G

A capacidade do processo é uma forma de avaliar o grau de institucionalização com que o processo é implementado (SOFTEX, 2016b). "No nível de capacidade G, a execução do processo é gerenciada" (SOFTEX, 2021a).

Conforme visto no Capítulo 2, a versão de 2021 do guia geral do MPS.BR utiliza o termo "resultados esperados" para capacidade de processos. Em versões anteriores a 2021, o modelo utilizava o termo RAP (Resultado de Atributo de Processo). Os resultados esperados do nível CP-G são:

3.4.1 Resultado esperado CP-G1

O processo produz os resultados definidos (SOFTEX, 2021a).

Para satisfazer esse resultado, é necessário garantir que o processo transforme produtos de trabalho identificáveis em produtos de saída também identificáveis, atingindo o objetivo do processo.

Conforme a análise realizada na Seção 3.3, o projeto executa determinadas tarefas, como revisão por pares, testes automatizados, utilização de protótipos, e testes de usabilidade, que garantem que o processo de verificação e validação é realizado para determinados produtos de trabalho. Entretanto, não foram encontradas evidências da execução do processo de verificação e validação para outros produtos de trabalho, como documentos de arquitetura. Além disso, apesar de produzir os resultados definidos, certas partes do processo não possuem documentação adequada e atualizada. Com base na Tabela 5, o resultado esperado do nível CP-G1 é classificado como **parcialmente implementado**.

3.4.2 Resultado esperado CP-G2

A execução do processo é planejada e monitorada (SOFTEX, 2021a).

Para satisfazer esse resultado, são necessárias evidências de que existe planejamento e monitoramento do processo de verificação e validação de software do projeto, além da tomada de ações corretivas no processo, caso necessário.

Sobre o planejamento do processo de verificação, foi identificado no resultado esperado VV1 que existem pelo menos 2 documentos ("Frontend - Testes.md" e "BFF - Testes.md") que preveem que tipo de elementos do sistema devem ser testados. Alguns desses elementos possuem guias de como realizar os testes, porém outros não.

Não foram encontradas evidências do planejamento e monitoramento do processo de verificação de artefatos além do código-fonte do projeto.

O processo de validação sempre deve ser executado no ciclo de vida do projeto para garantir que os requisitos dos usuários finais sejam atingidos. Uma evidência do planejamento do processo de validação é o documento "Teste de usabilidade - Tarefas.xlsx" (Figura 24). Entretanto, o documento em questão encontra-se bastante desatualizado, sugerindo que possivelmente não é utilizado por vários meses. Sobre as reuniões de planejamento de testes de usabilidade com os usuários, elas ocorrem conforme a necessidade. Não existe um cronograma definido para as atividades.

Em relação ao monitoramento do processo verificação, foram identificadas evidências comprovando que existem etapas de elaboração, execução, análise e interpretação de teste de software no projeto, por testes unitários, revisão de código e pipelines de integração contínua. Essas atividades são obrigatórias no processo do Órgão X, portanto devem ser executadas independente de quais membros estejam envolvidos. Os resultados das atividades se tornam disponíveis para os outros no repositório do GitLab.

Em relação ao monitoramento do processo de validação, foram encontradas evidências de que o processo possui fases de planejamento, execução e análise documentadas de maneira simplificada, porém, partes da documentação estão desatualizadas.

Tendo em vista as observações realizadas na Seção 3.3 e as orientações da Tabela 5, o resultado esperado do nível CP-G2 é classificado como **parcialmente implementado**.

3.4.3 Resultado esperado CP-G3

> As pessoas estão preparadas para executar suas responsabilidades no processo (SOFTEX, 2021a).

Para satisfazer esse resultado, são necessárias evidências de que a equipe de desenvolvimento do projeto tem capacidade para executar os processos de verificação e validação de software.

O resultado esperado VV1 confirma que existe documentação (com partes incompletas e desatualizadas no momento da análise) para instruir os membros da equipe a testar determinados elementos do sistema. No resultado VV2 foi identificado que a equipe utiliza a estratégia de revisão de código através das merge requests de branches no projeto. Isso indica que, apesar do desenvolvedor implementar e testar novos trechos de código, outro desenvolvedor deve fazer uma revisão para garantir que o novo código esteja condizente com os requisitos estabelecidos. Além disso, os resultados esperados VV3, VV4 e VV5 confirmam que, através do uso de pipelines de integração contínua, trechos de código que não satisfaçam os testes implementados devem ser corrigidos o mais rápido possível.

Não foram encontradas evidências que assegurem que os membros do projeto possuam capacidade para executar os processos de verificação e validação. Assume-se que tal capacidade pode ter sido avaliada no processo de contratação dos membros da equipe, porém não há evidências concretas que sustentem essa afirmação.

Também não foram encontradas evidências de que existam treinamentos para os membros aprenderem mais sobre as tarefas relacionadas aos processos de verificação e validação.

Realizadas as observações e tomando como base a análise da Seção 3.3, e as orientações da Tabela 5, o resultado esperado do nível CP-G3 é classificado como **parcialmente implementado**.

Os pontos fracos identificados no resultado esperado CP-G3 foram resumidos na Tabela 12.

Tabela 12 – Pontos fracos identificados no resultado esperado CP-G3

PF	**Descrição do ponto fraco**
PF11	Falta de evidências da capacidade da equipe para executar as atividades do processo de VV

3.5 Análise dos resultados

Resumindo as seções anteriores, todos os resultados esperados, tanto do processo de Verificação e Validação, quanto do nível G de capacidade de processo, foram classificados como parcialmente implementados (Tabela 13).

Tabela 13 – Resumo da avaliação dos resultados esperados

Resultado esperado	Grau de implementação
VV1	Parcialmente implementado
VV2	Parcialmente implementado
VV3	Parcialmente implementado
VV4	Parcialmente implementado
VV5	Parcialmente implementado
CP-G1	Parcialmente implementado
CP-G2	Parcialmente implementado
CP-G3	Parcialmente implementado

Como visto na Seção 2.2.2.2, um processo é considerado satisfeito quando todos os resultados esperados do processo e da capacidade de processo são total ou largamente implementados. Todos os resultados esperados, de processo e de capacidade do processo, foram avaliados como parcialmente implementados. Sendo assim, considerando também o modelo de avaliação escolhido para este trabalho (descrito na Seção 3.2), é possível

afirmar que **o projeto não implementa de maneira satisfatória o processo de Verificação e Validação** do MPS.BR em relação ao nível G de capacidade de processo.

Foram identificados pontos fracos no processo que podem ser tratados para melhorar o processo de verificação e validação. O resumo dos pontos fracos e as relações com os resultados esperados podem ser observados na Tabela 14.

Tabela 14 – Lista de pontos fracos identificados

PF	**Descrição do ponto fraco**	**Origens do ponto fraco**
PF1	Falta de critérios de seleção de produtos de trabalho a serem verificados e validados	VV1, CP-G2
PF2	Falta de planejamento de cronograma e recursos necessários para execução da revisão por pares	VV2, CP-G2
PF3	Falta da realização de revisão por pares em outros produtos de trabalho além do código-fonte	VV2, CP-G2
PF4	Falta de planejamento de testes (casos de teste)	VV3, VV4, CP-G2
PF5	Falta de descrição de como testar determinados elementos da interface de usuário	VV3, CP-G1, CP-G2
PF6	Documentação desatualizada	VV3, CP-G1, CP-G2
PF7	Falta do planejamento de VV em outros produtos de trabalho além da interface de usuário e do código-fonte	VV3, CP-G2
PF8	Falta da execução de VV em outros produtos de trabalho além da interface de usuário e do código-fonte	VV4, CP-G1, CP-G2
PF9	Falta da análise de resultados de VV em outros produtos de trabalho além da interface de usuário e do código-fonte	VV5, CP-G2
PF10	Desuso de indicadores de testes pela equipe	VV5, VV4, CP-G2
PF11	Falta de evidências da capacidade da equipe para executar as atividades do processo de VV	CP-G3, CP-G2

4 Propostas de Melhorias

Neste capítulo são propostas algumas melhorias para os pontos fracos identificados no Capítulo 3. Cada ponto fraco listado na Tabela 14 é discutido a seguir, bem como alternativas de como mitigá-los.

4.1 PF1 - Falta de critérios de seleção de produtos de trabalho a serem verificados e validados

Foi observado que não existem evidências da seleção de quais produtos de trabalho devem ser validados, já que a verificação ocorre basicamente no código-fonte. A verificação dos produtos de software é essencial para a identificação e correção de defeitos, impactando diretamente na qualidade do produto de software.

Deve-se criar um documento que registre quais produtos de trabalho devem ser verificados. É importante considerar os riscos de não se verificar tais produtos e priorizá-los no documento. Os produtos de trabalho a serem selecionados devem seguir critérios estabelecidos pela equipe, como impacto no projeto, contribuições para a satisfação de requisitos e riscos envolvidos.

Semelhantemente ao processo de verificação, não foram encontradas evidências da seleção de produtos de trabalho a serem validados. É necessário criar critérios para definir quais produtos de trabalho devem ser validados. Deve-se considerar o impacto no projeto a longo prazo, contribuições para a satisfação de requisitos e os riscos envolvidos.

Também pode ser interessante utilizar uma lista em nível organizacional de produtos comumente validados. Diretrizes para guiar a seleção dos produtos de trabalho a serem validados podem ser criadas ou incrementadas.

Caso não exista uma listagem dos produtos de trabalho desenvolvidos no projeto, uma alternativa é elaborar o documento PBS (*Product Breakdown Structure*). Esse documento define todos os produtos de trabalho gerados no projeto e fornece suporte à gerência (GPS, 2010). O exemplo de PBS, mostrado na Figura 30, pode ser adaptado para as necessidades do projeto específico do Órgão X, servindo assim como base para a seleção de produtos de trabalho a serem verificados e validados.

O PBS, quando implementado, poderá ser utilizado como entrada para a tarefa de seleção de produtos de trabalho a serem validados. Os critérios de seleção podem ser definidos pela equipe e, posteriormente, ajustados conforme a necessidade.

O registro dos produtos de trabalho selecionados para verificação e validação pode

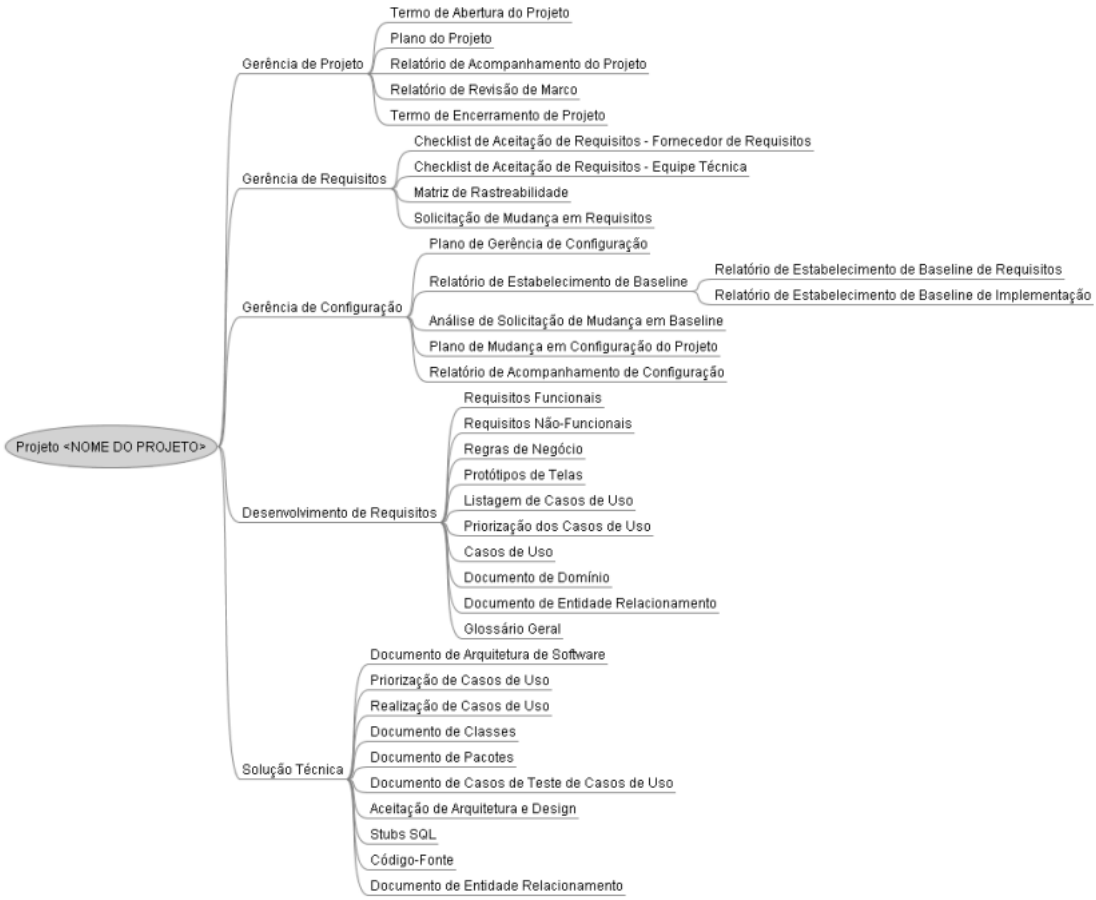


Figura 30 – Exemplo de *Product Breakdown Structure*. Fonte: (GPS, 2010)

ser feito no plano de teste. Um modelo é apresentado no Apêndice C, e a Seção C.5 lida especificamente com a listagem de produtos de trabalho a serem verificados e validados.

4.2 PF2 - Falta de planejamento de cronograma e recursos necessários para execução da revisão por pares

A equipe utiliza conceitos de metodologias ágeis, como Scrum, Extreme Programming e Kanban, em seu ciclo de desenvolvimento. Por isso, o foco é maior em desenvolver o produto do que em planejar e documentar as tarefas com grande precisão. Cronogramas e planejamento de recursos quase não são utilizados nas tarefas do projeto.

Apesar da equipe não utilizar cronogramas e planejamento de recursos com frequência, esses itens seriam de grande ajuda, pois poderiam auxiliar as tarefas a serem executadas no prazo, e por consequência, com menor desperdício de recursos.

Estimar no cronograma as principais datas das realizações das tarefas, como preparativos, execução e pós-tarefa. Realizar reuniões de planejamento, analisar a situação e registrar os recursos necessários para realização da revisão por pares: participantes,

equipamentos, ferramentas e quaisquer outros recursos necessários. A medida que datas, recursos e informações forem modificados, manter a documentação relacionada atualizada.

Checklists podem ser utilizados para identificar os itens a serem verificados, e podem avaliar aspectos como, por exemplo, regras de negócio, corretude, manutenibilidade, padrões de design e outras questões julgadas importantes para o projeto. Para cada questão da lista, fornecer as opções sim/não ou satisfatório/insatisfatório e, para cada questão não satisfeita, fornecer um comentário do porquê da avaliação. Um exemplo de checklist pode ser observado na Figura 31, e as questões podem ser adaptadas para atender ao contexto do projeto.

Preenchido por: <Colocar aqui o nome do responsável pela elaboração do relatório>

Em: <Colocar aqui a data da elaboração>

2.Relatório

Sim	Não	Questões
		Todos os artefatos de projeto foram avaliados?
		O conjunto de artefatos de design contempla de forma sistêmica todos os requisitos aprovados para o software?
		Há algum item no projeto que não se relaciona com nenhum requisito aprovado e, portanto, é desnecessário?
Explicação dos itens acima indicados como "Não"		
<Colocar aqui a justificativa para tal ocorrência de todos os itens acima identificados como "não", e apontar problemas, identificando-os de forma única. As ações corretivas para os problemas identificados durante este período deverão ser identificadas e acompanhadas em planilha adequada na Tabela de ações corretivas do projeto (ou seja, tem que ser repassadas ao Gerente do projeto).>		
Itens avaliados		
<Colocar aqui os itens que foram avaliados na verificação.>		

Figura 31 – Exemplo de checklist para auxiliar a verificação. Fonte: (GPS, 2010)

4.3 PF3 - Falta da realização de revisão por pares em outros produtos de trabalho além do código-fonte

A revisão por pares é um dos principais métodos de verificação disponíveis. Um artefato, ao ser revisado por outra pessoa que não o desenvolvedor, possui mais chances de ter seus defeitos detectados.

Outros artefatos que não sejam o código-fonte também devem ser verificados, e a revisão por pares é um método simples, porém eficaz para realizar essa tarefa. A documentação de planejamento, execução e análise da revisão por pares nos produtos de trabalho selecionados é essencial para assegurar a qualidade do produto.

Após a implementação de um produto de trabalho, outro membro da equipe deve realizar a revisão do artefato de acordo com critérios e diretrizes estabelecidos pela equipe. Essa revisão pode até mesmo ser realizada na forma de *merge requests*, como já ocorre atualmente no projeto na revisão de código-fonte.

Os defeitos identificados e dados coletados devem ser registrados em um documento e posteriormente analisados. Novas revisões por pares devem ser reagendadas caso sejam necessárias. Os detalhes da revisão por pares podem ser inclusos no documento de plano de teste, e um modelo é fornecido no Apêndice C.

A partir da análise dos dados da revisão por pares em um determinado produto de trabalho, será possível ter conhecimento sobre diversos fatores, como a quantidade de defeitos, severidade de defeitos, causas de defeitos, fases do projeto que determinados defeitos surgiram, requisitos de usuários afetados pelos defeitos, entre outras informações.

4.4 PF4 - Falta de planejamento de testes (casos de teste)

O projeto utiliza o documento "Teste de usabilidade - Tarefas.xlsx" para registrar os critérios de sucesso ou falha dos testes de usabilidade. Entretanto, não foram encontradas evidências de um documento semelhante para registrar critérios de testes unitários, de integração ou de sistema. Atualmente, a tarefa de testar o código-fonte depende fortemente da experiência do desenvolvedor, o que não é um critério consistente considerando os diferentes membros da equipe, e considerando a rotatividade da equipe. Uma maneira de facilitar a padronização dos testes implementados é através da utilização de casos de teste.

Os casos de teste podem ser utilizados no projeto para auxiliar no planejamento dos testes. Deve ser criado um modelo no diretório de documentação do projeto, no qual o caso de teste deve possuir um resumo, componente testado, pré-condições, passos para a execução, resultados esperados, executor, status, comentários. Um exemplo de modelo que pode ser aplicado é apresentado na Tabela 15.

A organização do planejamento de testes pode ser feita seguindo o modelo de plano de teste, na Seção C.6.

Dessa forma, os testes realizados no sistema e suas respectivas documentações irão manter um padrão entre os membros, facilitando o entendimento e a comunicação por parte da equipe.

Tabela 15 – Exemplo de modelo de caso de teste. Fonte: Adaptado de: (GPS, 2010)

Descrição	<Finalidade do teste realizado>
Componente	<Componente do sistema a ser testado>
Pré-condições	<Condições que devem ser satisfeitas antes de iniciar o teste>
Passos a serem executados	<Sequência de tarefas a serem realizadas durante o teste>
Resultados Esperados	<Resultados esperados após a execução do teste>
Executor	<Autor da execução do teste>
Status	<Classificação do teste: sucesso ou falha>
Comentários	<Observações realizadas durante a realização do teste, defeitos identificados e afins>

4.5 PF5 - Falta de descrição de como testar determinados elementos da interface de usuário

O documento "Frontend - Testes.md" é um dos documentos presentes no repositório GitLab do projeto, e visa auxiliar no processo de teste de elementos do frontend (interface de usuário). Ele fornece uma base para implementação dos diferentes tipos de testes no repositório, além de um guia geral de ferramentas utilizadas.

Durante a análise do documento, notou-se que, dos seis itens a serem testados (*UI components, container components, hooks, reducers, selectors* e *utils*), apenas o item *container component* possui descrição adequada. Todos os outros itens, ou possuem uma descrição rasa (como o item *utils*), ou possuem apenas o termo TODO (*to do* - a fazer) como preenchimento do texto, não servindo como guia para membros da equipe que utilizem o documento (Figura 25).

Uma melhoria sugerida é implementar a documentação inicial de cada item restante. Pode ser utilizada como referência a descrição do item *container component*, que possui pequenos textos explicativos, *links* para referências externas e trechos de código-fonte como exemplo. Outra alternativa é, no próprio documento, referenciar testes de cada elemento já existentes no repositório, para servirem de guia.

Em relação à complexidade da tarefa, poucos dias devem ser suficientes para estudar o conteúdo associado aos diferentes tipos de teste, escrever o texto e fornecer exemplos. Portanto, esse ponto fraco é considerado simples de ser resolvido.

A descrição detalhada dos procedimentos para testar os elementos do frontend no projeto será importante para a documentação geral do projeto, além de facilitar a padronização de código (ao seguir os guias da equipe), a inclusão e autonomia de novos membros que farão parte da equipe no futuro.

4.6 PF6 - Documentação desatualizada

Este ponto fraco se relaciona com outros pontos fracos identificados. De maneira geral, foi observado que o projeto possui pouca documentação formal. Considerando que é um projeto recente e que tem planos de ser mantido por muitos anos, é de extrema importância que a documentação seja tratada com seriedade quanto antes. Quanto mais o projeto avança, mais difícil e custoso se torna a construção de uma documentação precisa e atualizada.

Apesar de não existirem membros alocados exclusivamente para documentação de tarefas, os membros da equipe de desenvolvimento podem revesar a responsabilidade de documentar diferentes etapas do processo de desenvolvimento, visando a manutenibilidade futura do projeto.

Alguns documentos consultados para a realização da análise, discutida na Seção 3.3, estavam consideravelmente desatualizados. Foram encontrados documentos com mais de um ano sem modificações, indicando que não eram utilizados. A questão dos documentos desatualizados é preocupante porque, por mais que os membros atuais da equipe possam estar cientes do contexto e dos detalhes do projeto, o mesmo não pode ser dito para novos membros. Outro detalhe é que, à medida que os membros atuais deixam o projeto, eles podem acabar levando boa parte do conhecimento não formalizado, dificultando a integração e adaptação de novos membros.

Também foi observado que os documentos não foram encontrados de maneira centralizada: existem documentos no repositório GitLab do projeto, enquanto outros estavam localizados em pastas no Microsoft Teams. Essa descentralização não parece ser necessária, uma vez que o intuito do diretório “docs” no repositório GitLab é justamente agrupar os documentos do projeto.

Seria interessante se toda a documentação existente do projeto ficasse centralizada no diretório “docs” do GitLab do projeto, pois as chances de algum detalhe passar despercebido pela equipe seriam menores. Além disso, uma ótima vantagem de manter a documentação no GitLab é que poderia ser utilizado o versionamento de código para acompanhar a evolução da documentação, além de facilitar o processo de revisão por pares.

Como sugestão de melhoria, é recomendado que os documentos sejam revisados e incrementados regularmente, de preferência uma vez por sprint, ou caso não seja possível, uma vez por mês.

Outra sugestão é que sejam criados documentos com resumos das reuniões entre usuários e membros das equipes de UX e de desenvolvimento, onde possam ser listados os principais assuntos abordados, opiniões dos usuários (sugestões, elogios, criticas, reclamações) e considerações das equipes. Finalizada a tarefa, pode ser feita uma etapa de revisão

por pares para garantir que os conteúdos alterados ou adicionados estejam coerentes com a realidade do projeto.

As tarefas de manter os documentos não precisam ser atribuídas a exclusivamente um membro específico. Seria interessante que todos os membros participassem de maneira alternada.

4.7 PF7 - Falta do planejamento de VV em outros produtos de trabalho além da interface de usuário e do código-fonte

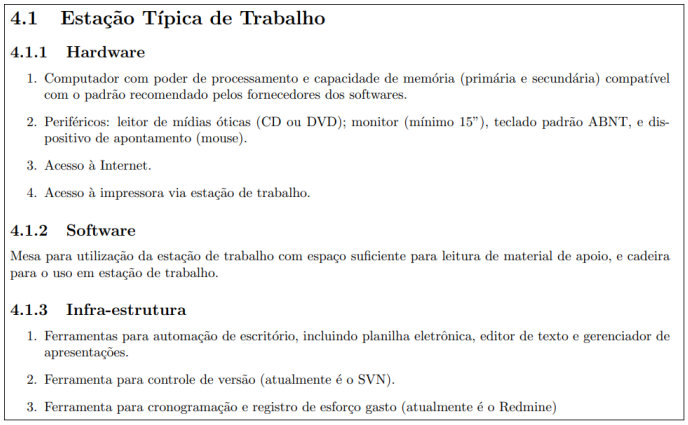
Atualmente, não existem critérios para selecionar produtos de trabalho a serem validados, apenas a interface de usuário e o código-fonte passam pelo processo. Por isso, é importante estabelecer critérios para determinar quais produtos de trabalho devem ser verificados e validados.

A verificação e validação, então, deve ser realizada nos produtos de trabalho selecionados através dos critérios estabelecidos e documentados pela equipe. É necessário planejar as etapas de verificação e validação, definindo cronogramas, recursos, materiais necessários, critérios de avaliação, infraestrutura necessária, membros envolvidos e suas responsabilidades. Além disso, as ferramentas de suporte às atividades devem ser determinadas.

Para auxiliar no planejamento das atividades de verificação e validação, é recomendável elaborar um plano de teste. O RUP (*Rational Unified Process*) apresenta um template de plano de teste que pode ser utilizado para se adaptar às necessidades do projeto (RSC, 2002). Um modelo baseado no template do RUP foi elaborado de maneira a facilitar a integração no projeto do Órgão X, e pode ser acessado no Apêndice C.

A infraestrutura necessária para execução das atividades pode ser documentada de maneira simples, listando as ferramentas utilizadas frequentemente no processo, como visto no exemplo da Figura 32.

A definição de papéis e responsabilidades dos membros do projeto pode ser interessante para deixar claro as tarefas esperadas para integrante. As Figuras 33 e 34 são exemplos de responsabilidades que podem ser documentadas no projeto, e outras adições podem ser feitas caso seja de interesse da equipe.



4.1 Estação Típica de Trabalho

4.1.1 Hardware

1. Computador com poder de processamento e capacidade de memória (primária e secundária) compatível com o padrão recomendado pelos fornecedores dos softwares.
2. Periféricos: leitor de mídias óticas (CD ou DVD); monitor (mínimo 15"), teclado padrão ABNT, e dispositivo de apontamento (mouse).
3. Acesso à Internet.
4. Acesso à impressora via estação de trabalho.

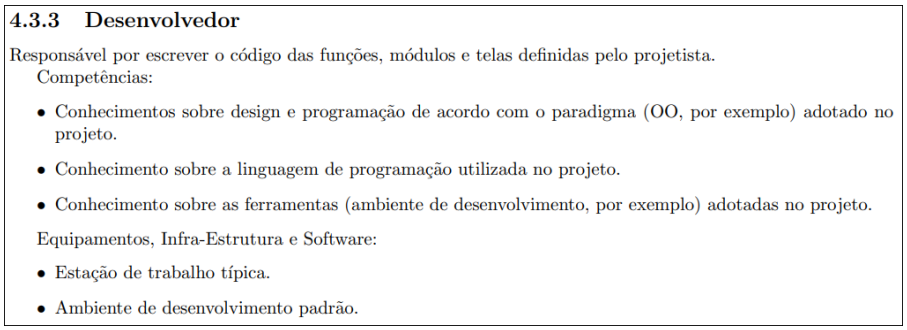
4.1.2 Software

Mesa para utilização da estação de trabalho com espaço suficiente para leitura de material de apoio, e cadeira para o uso em estação de trabalho.

4.1.3 Infra-estrutura

1. Ferramentas para automação de escritório, incluindo planilha eletrônica, editor de texto e gerenciador de apresentações.
2. Ferramenta para controle de versão (atualmente é o SVN).
3. Ferramenta para cronogramação e registro de esforço gasto (atualmente é o Redmine)

Figura 32 – Listagem de infraestrutura necessária para executar os processos. Fonte: (GPS, 2010)



4.3.3 Desenvolvedor

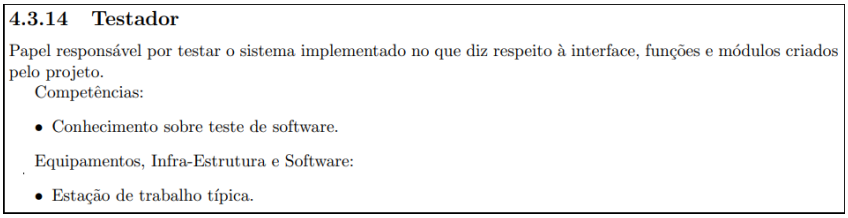
Responsável por escrever o código das funções, módulos e telas definidas pelo projetista.
Competências:

- Conhecimentos sobre design e programação de acordo com o paradigma (OO, por exemplo) adotado no projeto.
- Conhecimento sobre a linguagem de programação utilizada no projeto.
- Conhecimento sobre as ferramentas (ambiente de desenvolvimento, por exemplo) adotadas no projeto.

Equipamentos, Infra-Estrutura e Software:

- Estação de trabalho típica.
- Ambiente de desenvolvimento padrão.

Figura 33 – Definição dos papéis de desenvolvedor. Fonte: (GPS, 2010)



4.3.14 Testador

Papel responsável por testar o sistema implementado no que diz respeito à interface, funções e módulos criados pelo projeto.
Competências:

- Conhecimento sobre teste de software.

Equipamentos, Infra-Estrutura e Software:

- Estação de trabalho típica.

Figura 34 – Definição dos papéis de testador. Fonte: (GPS, 2010)

4.8 PF8 - Falta da execução de VV em outros produtos de trabalho além da interface de usuário e do código-fonte

Esse é considerado um dos principais pontos fracos do processo de VV do projeto. O código-fonte possui um bom suporte à verificação. A interface de usuário (frontend), o principal produto de software do projeto, possui um suporte considerável à validação. Entretanto, outros produtos de trabalho importantes do projeto não possuem suporte a VV, ou se possuem, não são documentados.

É de extrema importância verificar e validar os principais produtos de trabalho, começando desde os requisitos e indo até o produto pronto para ser entregue ao usuário.

Cada verificação e validação de produtos de trabalho específico devem ser documentados em detalhes, para auxiliar não só os membros atuais, mas também novos membros que possam integrar a equipe no futuro.

Critérios para validação e verificação de cada artefato podem ser definidos pela equipe ou baseados em diretrizes organizacionais (caso existam), e podem ser utilizados como *checklists* ou tabelas para auxiliar na avaliação dos produtos de trabalho. O exemplo apresentado na Seção C.6.2 pode ser utilizada como inspiração.

4.9 PF9 - Falta da análise de resultados de VV em outros produtos de trabalho além da interface de usuário e do código-fonte

Após realizar a execução da verificação e validação nos produtos de trabalho selecionados, é necessário analisar os dados gerados. Após a análise dos resultados dos testes, deve-se registrar informações, como, por exemplo, se os critérios definidos foram satisfeitos, se ações corretivas planejadas foram concluídas e se a verificação e validação foram executadas conforme planejado. Outras informações relevantes podem ser registradas conforme a necessidade. Com essas informações, deve ser possível avaliar se o processo satisfaz as expectativas da equipe e do cliente, ou identificar problemas que surgiram durante sua execução.

Com os dados obtidos é possível também organizar métricas e indicadores, para comparar o processo em diferentes etapas do ciclo de vida do projeto. Os comparativos dos indicadores com o passar do tempo podem ser apresentados na forma de gráficos e tabelas, facilitando seu entendimento.

4.10 PF10 - Desuso de indicadores de testes por parte da equipe

Através de entrevistas com o gerente do projeto, foi identificado que a equipe não utiliza com frequência os indicadores fornecidos pela ferramenta SonarQube. A ferramenta apresenta, entre outros indicadores, a quantidade de bugs encontrados, severidade dos bugs, cobertura de testes, condições não cobertas por testes, dívida técnica e code smells.

Seria interessante para o projeto que, periodicamente, fossem atribuídas tarefas relacionadas aos indicadores do SonarQube. Uma tarefa, por exemplo, poderia ser atingir um determinado valor de cobertura de testes, sugerido pela ferramenta. Essas ações são importantes para a manutenção do sistema e prevenir o surgimento de defeitos.

4.11 PF11 - Falta de evidências da capacidade da equipe para executar as atividades do processo de VV

Conforme discutido na Seção 3.4, não foram encontradas evidências que assegurem a capacidade dos membros para executar o processo. Em uma das entrevistas com o gerente do projeto, foi confirmado que a capacidade dos membros depende apenas de suas próprias habilidades pessoais.

Apesar de ser considerado um ponto fraco não existirem evidências da capacidade dos membros para executar os processos de verificação e validação, essa questão não chega a ser um problema, pois um dos requisitos para se tornar um membro da equipe de desenvolvimento é ser aprovado no processo seletivo do Órgão X. Esse processo, por si só, pode servir como garantia de que os membros aceitos no projeto possuem capacidade de projetar, desenvolver, testar e manter um projeto de software.

Entretanto, uma boa prática a ser realizada no projeto é a utilização de treinamentos, *walkthroughs*, guias e mentorias para auxiliar os membros a melhorarem suas capacidades de planejar, executar e analisar os processos de verificação e validação. Esses materiais de apoio podem ser implementados por membros mais experientes, ou mesmo retirados de fontes confiáveis, como livros, artigos e palestras de especialistas no assunto.

As Figuras 33 e 34 apresentam maneiras de se documentar as responsabilidades dos membros integrantes da equipe, como uma das atividades que podem contribuir para o processo de VV.

5 Conclusões

Este trabalho teve como objetivo avaliar e propor algumas melhorias ao processo de teste de software de um projeto real de um órgão do setor público. Inicialmente foi realizada uma pesquisa bibliográfica para preparar a fundamentação teórica a respeito dos principais assuntos a serem explorados. Em seguida, foi realizado um estudo de caso em um órgão referenciado como "Órgão X", onde o processo de verificação e validação foi avaliado utilizando como referência partes do modelo MPS.BR e do modelo CMMI-DEV. Com os resultados obtidos da análise, foram apresentadas algumas melhorias para os problemas identificados.

O processo de teste de software foi analisado de acordo com dois conceitos fundamentais: verificação e validação. Foi observado durante a análise que o processo de verificação do projeto é implementado em maior parte considerando o código-fonte, mas não possui formalidade em relação a planejamento de recursos, tempo ou pessoal capacitado. Além disso, são implementados diferentes tipos de testes, em especial testes unitários, e eles são automatizados em uma etapa do pipeline de integração contínua, configurado no repositório do projeto no GitLab. Não foram encontradas evidências da realização da verificação de outros produtos de trabalho, como plano de projeto e diagramas arquiteturais.

O processo de validação se baseia principalmente no uso de testes de usabilidade, no qual, conforme a necessidade, são organizadas reuniões entre membros das equipes de desenvolvimento e usuários finais para validar novas funcionalidades implementadas ou alterações em funcionalidades já existentes. Os resultados dos testes de usabilidade são analisados e documentados na forma de gravações, então mudanças são realizadas e novos testes são agendados. Assim como no processo de verificação, não foram encontradas evidências de que o projeto realiza a validação de outros produtos de trabalho além de requisitos e da própria interface de usuário.

De forma geral, foi possível observar que o projeto avaliado nesse trabalho implementa etapas essenciais da execução dos processos de verificação e validação. Os requisitos são revisados, funcionalidades e componentes são testados e usuários confirmam se tudo corresponde as suas expectativas. O principal problema está relacionado relacionado a documentação. Muitas questões não estão formalmente documentadas e só podem ser propriamente entendidas após conversas e discussões com outros membros. Essa prática pode dificultar a independência de novos membros ao lidar com tarefas realizadas a esses dois processos. Isso também pode prejudicar consideravelmente a manutenção do projeto a longo prazo.

Caso o projeto tenha seus pontos fracos relacionados a documentação resolvidos, e

o planejamento dos processos seja realizada de maneira apropriada, é possível melhorar de maneira significativa a qualidade dos processos de verificação e validação, e, consequentemente, melhorar a qualidade do produto em questão desenvolvido pela equipe do Órgão X. Além disso, os pontos fracos apresentados podem servir como indicadores de outras questões importantes no projeto, como a documentação relacionada a outros processos do ciclo de desenvolvimento. Com isso, é possível melhorar o desenvolvimento do projeto de uma maneira geral.

As melhorias propostas nesse trabalho tiveram suas discussões simplificadas, portanto, como trabalho futuro poderia ser investigado de forma mais aprofundada como implementar as melhorias de maneira detalhada no contexto do projeto. Também não foi possível apresentar a equipe nem os pontos fracos identificados nem as melhorias sugeridas. Dessa forma, também como trabalho futuro, seria importante apresentar os resultados desse trabalho e validar com a equipe tanto os pontos fracos como as sugestões de melhoria sugeridas.

Uma outra maneira de dar continuidade a este trabalho é acompanhar a implementação das melhorias propostas e analisar os impactos no desenvolvimento do projeto, comparando métricas de antes e depois da implementação. Caso os resultados da experiência sejam satisfatórios, novas melhorias ao processo de verificação e validação podem ser identificadas, utilizando procedimentos similares aos utilizados neste trabalho. Além disso, o escopo das propostas de melhorias pode ser expandido para outros processos, além de verificação e validação, como elicitação de requisitos e projetos de arquitetura.

5.1 Ameaças a validade da pesquisa

De acordo com Yin et al. (2003), existem quatro testes utilizados para estabelecer a qualidade de qualquer pesquisa social empírica, e como os estudos de caso fazem parte desse tipo de pesquisa, esses testes são relevantes ao se realizar estudos de caso. Considerando a validade do construto, validade externa e confiabilidade da pesquisa, e o escopo deste trabalho, tem-se:

- Validade do construto: O uso de guias já utilizados, testados e aprovados, como o MPS.BR e o CMMI-DEV, ao invés de desenvolver as próprias técnicas, é uma maneira de mitigar a subjetividade do autor sobre as informações acerca do processo de verificação e validação do projeto.

- Validade externa: O trabalho atual **não** tem objetivo de generalizar os resultados. Todos os resultados obtidos e suas conclusões são aplicáveis **apenas** ao contexto do projeto escolhido.

- Confiabilidade: O uso do protocolo de estudo de caso e sua consequente descrição na metodologia deste trabalho é uma forma de assegurar a confiabilidade do estudo.

5.2 Limitações da pesquisa

Apesar do planejamento e da inspiração em modelos de referência, esta pesquisa possui limitações que devem ser observadas.

Conforme discutido na Seção 3.1.2, a equipe de desenvolvimento utiliza conceitos de metodologias ágeis ativamente no projeto, como *Scrum* e *Extreme Programming*. Possivelmente devido a esse fator, a documentação do projeto está escassa, já que, como descrito no Manifesto Ágil, a metodologia ágil valoriza "software em funcionamento mais que documentação abrangente" (BECK et al., 2001). Portanto, existe a possibilidade de que alguns itens apontados como pontos fracos do processo de VV sejam implementados informalmente. Dessa forma, apesar de serem executados, não foram documentados adequadamente.

Mesmo depois de vários documentos terem sido analisados e entrevistas conduzidas, a coleta de mais dados provavelmente poderia ter revelado ainda mais problemas relacionados ao processo de VV do projeto. Nesse sentido, a quantidade de dados obtidos é considerada uma limitação deste trabalho.

Houve a preocupação em utilizar a representação contínua do CMMI junto à definição do processo do VV do MPS.BR. Essa forma de avaliação, entretanto, não está prevista no Método de Avaliação do MPS.BR. Dessa forma, corre-se o risco de a integração tenha sido feita de foma inadequada e incompleta.

Houve a intenção de realizar a análise de todos os achados da avaliação do processo, entretanto, devido ao tempo curto para execução desta pesquisa, algumas análises poderiam ser mais detalhadas. Com isso, o escopo da análise teve que ser reduzido.

Apesar das limitações observadas, o trabalho atual obteve sucesso em identificar pontos fracos no processo de verificação e validação do projeto, além de propor algumas melhorias viáveis de serem aplicáveis. Conforme discutido anteriormente, os resultados deste trabalho devem ser aplicados apenas ao projeto específico do "Órgão X", ou seja, não é possível generalizar para outros projetos e contextos.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISO/IEC 9126-1*: Engenharia de software - qualidade de produto, parte 1: Modelo de qualidade. Rio de Janeiro, 2003. Citado 2 vezes nas páginas 30 e 32.

Axivion. *Axivion Suite*. 2022. Disponível em: <https://www.axivion.com/produkte/axivion-suite/>. Citado na página 48.

BALTHAZAR, G. da R. Visão geral da qualidade de software. *Revista Eletrônica da Faculdade Metodista Granbery-http://re. granbery. edu. br-ISSN*, p. 0377, 1981. Citado na página 31.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <https://agilemanifesto.org/>. Citado na página 85.

BOEHM, B. W. Software engineering; r & d trends and defense needs. *Research Directions in Software Technology*, p. 1–9, 1979. Citado na página 44.

BOURQUE, P.; FAIRLEY, e. R. Guide to the software engineering body of knowledge: Version, 3.0. 2014. Disponível em: <www.swebok.org>. Citado 5 vezes nas páginas 21, 29, 30, 44 e 93.

BRERETON, P. et al. Using a protocol template for case study planning. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. [S.l.: s.n.], 2008. p. 1–8. Citado na página 25.

CMMI. *CMMI for Development, Version 1.3*. Pittsburgh, PA, 2010. Disponível em: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661>. Citado 3 vezes nas páginas 13, 35 e 36.

GPS. Manual de produção de software. In: UFG. [S.l.], 2010. Citado 7 vezes nas páginas 12, 13, 73, 74, 75, 77 e 80.

ISO. *ISO 9001:2000*: Quality management systems-requirements. [S.l.], 1994. Citado 2 vezes nas páginas 29 e 93.

ISO. *ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE*. [S.l.], 2005. Citado 2 vezes nas páginas 21 e 31.

ISO. *ISO/IEC 25010:2010, Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*. [S.l.], 2010. Citado 3 vezes nas páginas 11, 31 e 32.

JURISTO, N.; MORENO, A. M.; VEGAS, S. Limitations of empirical testing technique knowledge. In: *Lecture Notes on Empirical Software Engineering*. [S.l.]: World Scientific, 2003. p. 1–38. Citado na página 45.

KALINOWSKI, M. et al. Mps. br: Promovendo a adoção de boas práticas de engenharia de software pela indústria brasileira. In: *CIbSE*. [S.l.: s.n.], 2010. p. 265–278. Citado na página 37.

KRASNER, H. Using the cost of quality approach for software. *he Journal of Defense Software Engineering*, v. 11, n. 11, p. 6–11, 1998. Citado na página 29.

MARCILIO, D. et al. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. In: IEEE. *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. [S.l.], 2019. p. 209–219. Citado na página 47.

MELLO, M. S. de. *Melhoria de processos de software multi-modelos baseada nos modelos MPS e CMMI-DEV*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2011. Citado na página 35.

NDepend. *NDepend*. 2022. Disponível em: <https://www.ndepend.com/>. Citado na página 48.

OLIVEIRA, A.; PETRINI, M.; PEREIRA, D. L. Avaliação da adoção do cmmi considerando o custo de qualidade de software. *Gestão e Projetos: GeP*, Universidade Nove de Julho, v. 6, n. 1, p. 45–62, 2015. Citado 3 vezes nas páginas 11, 29 e 30.

PRESSMAN, R. S. *Engenharia de Software: Uma Abordagem Profissional*. [S.l.]: Palgrave macmillan, 2016. ISBN 978-85-8055-534-9. Citado 3 vezes nas páginas 32, 33 e 93.

QUALIDADEBR. *MPS.BR*. 2022. Disponível em: <https://qualidadebr.wordpress.com/2008/08/23/mpsbr/>. Citado 2 vezes nas páginas 11 e 39.

RSC. *Rational Unified Process: Templates*. 2002. Disponível em: <https://www.tesestec.com.br/pasteurjr/rup/process/templates.htm>. Citado 6 vezes nas páginas 13, 79, 97, 99, 100 e 101.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, Springer, v. 14, n. 2, p. 131–164, 2009. Citado na página 25.

SANTIAGO, M. R. UNIVERSIDADE GAMA FILHO POSEAD. p. 243, 2011. Citado na página 29.

SOFTEX. Mps.br - melhoria de processo do software brasileiro: Guia de implementação – parte 4: Fundamentação para implementação do nível d do mr-mps-sw:2016. 2016. Citado 5 vezes nas páginas 44, 45, 46, 55 e 93.

SOFTEX. Mps.br - melhoria de processo do software brasileiro: Guia de implementação – parte 1: Fundamentação para implementação do nível g do mr-mps-sw:2016. 2016. Citado na página 69.

SOFTEX. Mps.br - melhoria de processo do software brasileiro: Guia geral mps de software. 2021. Citado 14 vezes nas páginas 11, 37, 38, 39, 40, 49, 53, 54, 57, 61, 64, 67, 69 e 70.

SOFTEX. Mps.br - melhoria de processo do software brasileiro: Guia de avaliação. processo e método de avaliação ma-mps. 2021. Citado 3 vezes nas páginas 13, 40 e 41.

SOFTEXRECIFE. Mpt.br - melhoria de processo de teste brasileiro: Guia de referência do modelo. 2011. Citado 5 vezes nas páginas 11, 13, 42, 43 e 53.

SOMMERVILLE, I. *Engenharia de Software*: 9° edição. [S.l.: s.n.], 2011. ISBN 978-85-7936-108-1. Citado 9 vezes nas páginas 11, 21, 32, 33, 34, 44, 45, 46 e 47.

SonarSource S.A. *SonarQube*. 2022. Disponível em: <https://www.sonarqube.org/>. Citado na página 47.

THÖNES, J. Microservices. *IEEE software*, IEEE, v. 32, n. 1, p. 116–116, 2015. Citado na página 50.

UNTERKALMSTEINER, M. et al. Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Transactions on Software Engineering*, IEEE, v. 38, n. 2, p. 398–424, 2011. Citado na página 33.

XOTESLEM, V. C. Elaboração de estratégia de testes em equipe de voluntários geograficamente distribuída. 2021. Citado na página 53.

YIN, R. K. et al. Design and methods. *Case study research*, v. 3, n. 9.2, 2003. Citado na página 84.

Apêndices

APÊNDICE A – Termos e Definições

Neste apêndice estão reunidas para acesso rápido as principais definições de termos, definições e conceitos abordados ao longo do trabalho, principalmente no Capítulo 2. As referências citadas fornecem detalhes mais aprofundados a respeito de cada definição.

- Qualidade de software: “Grau com o qual um conjunto de características intrínsecas cumpre requisitos (ISO, 1994)”
- Processo de software: “Série de atividades, ações e tarefas necessárias para a produção de um produto de software de alta qualidade” (PRESSMAN, 2016)
- Teste de software: “Verificação dinâmica que um programa realiza comportamentos esperados em um conjunto finito de casos de teste, selecionados do domínio de execução geralmente infinito” (BOURQUE; FAIRLEY, 2014)
- Verificação: “Garantia de que produtos de uma fase particular do processo estão consistentes com os requisitos desta fase e da fase anterior” (SOFTEX, 2016a)
- Validação: ”Processo com objetivo de assegurar que o software que está sendo desenvolvido é o software correto conforme os requisitos do usuário“ (SOFTEX, 2016a)

APÊNDICE B – Termo de Consentimento Livre e Esclarecido

O conteúdo do termo de consentimento livre e esclarecido que será assinado pelos responsáveis do projeto é descrito a seguir:

Termo de Consentimento Livre e Esclarecido - TCLE
Título da pesquisa: Influência da Testabilidade na Qualidade em Uso de Software
Estudante: Renan Cristyan Araújo Pinheiro
Matrícula do estudante: 17/0044386
Orientadora: Dra. Fabiana Freitas Mendes

Prezado

Solicito a sua colaboração para a pesquisa de desenvolvimento do trabalho de conclusão de curso intitulada **"Influência da Testabilidade na Qualidade em Uso de Software"** de responsabilidade do estudante Renan Cristyan Araújo Pinheiro.

A pesquisa tem por objetivo investigar como as características de testabilidade de software podem influenciar a qualidade em uso do ponto de vista do usuário final. A participação nesta pesquisa consistirá no acesso a dados relacionados a projetos da empresa previamente acordados. Para avaliação da testabilidade, solicita-se acesso às métricas disponíveis no Sonar, além de acesso aos testes implementados para análise. Nenhum trecho de código-fonte ou referência ao projeto será documentado no trabalho. Além disso, solicita-se acesso a informações acerca da usabilidade, qualidade em uso e quaisquer registros disponíveis sobre a interação do usuário final com o sistema.

Para o(s) projeto(s) participantes os benefícios esperados pela execução desta pesquisa são:

1. Análise detalhada da testabilidade do produto;

2. Análise da opinião do usuário final (qualidade em uso) sobre o sistema;

3. Sugestões de melhorias possíveis de serem aplicadas no projeto, após resultados das análises;

4. Pesquisa sobre a relação entre testabilidade de software e qualidade em uso. O trabalho será disponibilizado futuramente na biblioteca de monografias da Universidade de Brasília, sendo disponível publicamente.

A empresa não terá nenhuma despesa ou gasto. Além disso, os nomes da empresa, do(s) projeto(s) e dos participantes da pesquisa serão mantidos em sigilo para assegurar a privacidade, e nenhuma informação além dos dados mencionados anteriormente será utilizada ou divulgada.

Os dados fornecidos serão utilizados única e exclusivamente para fins desta pesquisa, e os resultados serão publicados na biblioteca de monografias da Universidade de Brasília através do link <https://bdm.unb.br>.

Em caso de dúvidas, entre em contato com os membros da pesquisa.

Agradeço pela sua cooperação.

APÊNDICE C – Modelo de Plano de Teste

Neste apêndice é fornecido um modelo simplificado de plano de teste para o projeto do Órgão X, baseado no modelo fornecido pelo RUP (RSC, 2002), com adaptações. Deve-se notar que, como o projeto não possui no momento um plano de teste estabelecido, o modelo atual está simplificado para facilitar sua implementação. A medida que o plano for integrado ao projeto, as seções sugeridas pelo modelo do RUP podem ser inclusas.

C.1 Capa

A capa do documento deve apresentar as seguintes informações: Nome do projeto, nome do documento (plano de teste) e versão atual. Além disso, é interessante incluir uma tabela com o histórico de versionamento, descrevendo data, versão, alterações principais e autores (Tabela 16).

Tabela 16 – Modelo de histórico de versões

Data	Versão	Descrição	Autor
<dd/mm/aaaa>	<x.y>	<Resumo das principais alterações>	<Responsável pela versão>

C.2 Sumário

Apresentar as seções e subseções implementadas no documento.

C.3 Introdução

C.3.1 Propósito

Aqui deve ser feita uma breve descrição sobre o propósito do plano de teste. Por exemplo, pode-se dizer que o plano de teste pretende agrupar toda informação necessária para guiar os esforços de teste do projeto, além de descrever de uma forma geral como os testes devem ser planejados, implementados, executados e analisados.

Também podem ser apresentados os principais objetivos que o plano de teste pretende alcançar, como identificação de produtos de trabalho a serem testados, abordagens e estratégias definidas para executar os testes, listagem de recursos necessários para executar os testes e artefatos gerados ao final da execução do plano de teste.

C.3.2 Escopo

Apresentar resumidamente quais níveis de teste serão abordados no plano de teste, como testes unitários, de integração, de sistema, entre outros. Apresentar também quais tipos de teste serão descritos, como, por exemplo, testes de usabilidade, funcionalidade, confiabilidade, desempenho, segurança, carga, estresse, entre outros. Deixar claro quais níveis e tipos de teste serão explicitamente excluídos do plano de teste.

C.3.3 Público-alvo

Descrever para quem se destina o plano de teste. No caso do projeto do Órgão X, o plano de teste pode ser utilizado por desenvolvedores, testadores e gerentes de projeto. Apresentar justificativas e como o plano de teste pode ser útil para cada perfil.

C.3.4 Terminologia e acrônimos

Apresentar os principais termos, acrônimos e abreviações presentes ao longo do documento.

C.3.5 Referências

Apresentar as referências utilizadas ao longo do projeto: livros, artigos, sites, blogs, documentos do próprio projeto ou da organização.

C.4 Background

Descrever o projeto que será testado, abordando resumidamente seu propósito, funcionalidades, problemas que resolve e a arquitetura geral do sistema. Caso essas informações já existam em outros documentos, incluir as referências.

C.5 Itens a serem testados

Apresentar uma lista com os principais produtos de trabalho a serem testados, agrupando-os por categorias. Destes produtos de trabalho, podem ser listados o plano de projeto, o documento de arquitetura, os protótipos de alta fidelidade, o código-fonte, entre outros.

C.6 Abordagem de teste

Utilizar os itens de trabalho selecionados e os tipos de testes planejados para detalhar como os testes serão realizados.

C.6.1 Requisitos de teste

Listar os requisitos para cada tipo de teste. Por exemplo, verificar o desempenho do sistema com 1000 usuários simultâneos, verificar as permissões de acesso de conteúdo de determinado perfil de usuário, entre outros. Os requisitos de teste devem estar alinhados aos requisitos de usuário e do sistema.

C.6.2 Tipos de teste

Para cada tipo de teste, criar uma subseção e detalhar: objetivos do teste, técnicas utilizadas, ferramentas necessárias, critérios de conclusão e observações. A Tabela 17 pode ser utilizada como ponto de partida. Caso outras informações sejam relevantes para o projeto, incluí-las na tabela.

Tabela 17 – Modelo de detalhamento dos tipos de teste. Adaptado de (RSC, 2002)

Objetivos do teste	<Listar os principais objetivos do teste realizado>
Técnicas utilizadas	<Apresentar as técnicas ou passos utilizados para execução do teste>
Ferramentas necessárias	<Apresentar as ferramentas necessárias para execução do teste>
Critérios de conclusão	<Listar os critérios que asseguram se o teste obteve sucesso ou não>
Observações	<Descrever quaisquer comentários pertinentes que não foram abordados nas linhas acima>

C.7 Recursos

Listar todos os recursos necessários para a execução dos testes: recursos humanos (Tabela 18), de software (Tabela 19) e de hardware (Tabela 20).

Tabela 18 – Recursos humanos necessários para execução do plano de teste. Adaptado de (RSC, 2002)

Perfil	Quantidade mínima recomendada	Responsabilidades e comentários
<Perfil 1>	<a>	<Listar responsabilidades do perfil 1>
<Perfil 2>	<b>	<Listar responsabilidades do perfil 2>
<Perfil n>	<c>	<Listar responsabilidades do perfil n>

C.8 Pontos de controle

Listar os principais marcos (milestones), como, por exemplo, fase de planejamento, fase de design, fase de implementação, fase de execução e fase de avaliação, e para cada

Tabela 19 – Recursos de software necessários para execução do plano de teste. Adaptado de (RSC, 2002)

Nome do produto	Versão	Categoria
<Software 1>	<a>	<Categoria do software 1>
<Software 2>	<b>	<Categoria do software 2>
<Software n>	<c>	<Categoria do software n>

Tabela 20 – Recursos de hardware necessários para execução do plano de teste. Adaptado de (RSC, 2002)

Recurso	Quantidade	Nome e tipo
<Recurso 1>	<a>	<Nome e tipo do recurso 1>
<Recurso 2>	<b>	<Nome e tipo do recurso 2>
<Recurso n>	<c>	<Nome e tipo do recurso n>

marco, registrar as datas planejadas de início e término, bem como as datas executadas de início e término (Tabela 21).

Tabela 21 – Cronograma das principais tarefas realizadas no processo de VV. Adaptado de (RSC, 2002)

Fase	Data de início planejada	Data de início executada	Data de término planejada	Data de término executada
Planejamento de testes	<a>	<b>	<c>	<d>
Design de testes	<a+1>	<b+1>	<c+1>	<d+1>
Implementação de testes	<a+2>	<b+2>	<c+2>	<d+2>
Execução de testes	<a+3>	<b+3>	<c+3>	<d+3>
Avaliação de testes	<a+4>	<b+4>	<c+4>	<d+3>

C.9 Riscos

Listar os riscos que podem prejudicar a execução do plano de teste. Para cada risco, descrever maneiras de evitá-lo e, caso ocorra, maneiras de tratá-lo. Se possível, organizar os riscos considerando suas probabilidades de acontecerem e suas severidades (Tabela 22).

C.10 Artefatos gerados

Referenciar os artefatos gerados após a execução dos testes, tais como relatórios de testes, registros de defeitos e relatórios de homologação.

Tabela 22 – Listagem dos riscos associados ao plano de teste. Adaptado de (RSC, 2002)

Risco	Probabilidade de ocorrência	Estratégia de mitigação	Severidade	Contingência
<Risco 1>	<Alta, média ou baixa>	<Estratégia para evitar o risco 1>	<Alta, média ou baixa>	<Estratégia para tratar o risco 1>
<Risco 2>	<Alta, média ou baixa>	<Estratégia para evitar o risco 2>	<Alta, média ou baixa>	<Estratégia para tratar o risco 2>
<Risco 3>	<Alta, média ou baixa>	<Estratégia para evitar o risco 3>	<Alta, média ou baixa>	<Estratégia para tratar o risco 3>

Pode ser criada uma seção para cada artefato, ou, caso existam muitos dados, pode haver apenas um pequeno texto explicativo com uma referência para outros documentos detalhando esses artefatos.