CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
UNIDADE DE PÓS-GRADUAÇÃO, EXTENSÃO E PESQUISA
MESTRADO PROFISSIONAL EM GESTÃO E TECNOLOGIA EM SISTEMAS PRODUTIVOS

FERNANDA PEREIRA LEITE AGUIAR

MELHORIA DO TESTE DE SOFTWARE EM PROCESSO ÁGIL DE DESENVOLVIMENTO DE MICRO SERVIÇOS: UMA PROPOSTA DE ROADMAP

São Paulo
Março/2023

FERNANDA PEREIRA LEITE AGUIAR

MELHORIA DO TESTE DE SOFTWARE EM PROCESSO ÁGIL DE DESENVOLVIMENTO DE MICRO SERVIÇOS: UMA PROPOSTA DE ROADMAP

Dissertação apresentada como exigência parcial para a obtenção do título de Mestre em Gestão e Tecnologia em Sistemas Produtivos do Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos, sob a orientação do Prof. Dr. Marcelo Duduchi Feitosa

São Paulo
Março/2023

FICHA ELABORADA PELA BIBLIOTECA NELSON ALVES VIANA
FATEC-SP / CPS CRB8-8390

Aguiar, Fernanda Pereira Leite

A282m Melhoria do teste de software em processo ágil de desenvolvimento de micro serviços / Fernanda Pereira Leite Aguiar. – São Paulo: CPS, 2023.

235 f. : il.

Orientador: Prof. Dr. Marcelo Duduchi Feitosa

Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos) – Centro Estadual de Educação Tecnológica Paula Souza, 2023.

1. Sistemas de informação. 2. Desenvolvimento ágil de software. 3. Arquitetura de micro serviços. 4. Teste de software. 5. Roadmap. I. Feitosa, Marcelo Duduchi. II. Centro Estadual de Educação Tecnológica Paula Souza. III. Título.

FERNANDA PEREIRA LEITE AGUIAR

MELHORIA DO TESTE DE SOFTWARE EM PROCESSO ÁGIL DE DESENVOLVIMENTO DE MICRO SERVIÇOS: UMA PROPOSTA DE ROADMAP

gov.br
Documento assinado digitalmente
MARCELO DUDUCHI FEITOSA
Data: 13/04/2023 11:47:46-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcelo Duduchi Feitosa

Orientador – CEETEPS

gov.br
Documento assinado digitalmente
AURI MARCELO RIZZO VINCENZI
Data: 14/04/2023 18:17:52-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Auri Marcelo Rizzo Vincenzi

Examinador Externo – UFSCar

gov.br
Documento assinado digitalmente
MARILIA MACORIN DE AZEVEDO
Data: 13/04/2023 12:05:06-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Marília Macorin de Azevedo

Examinador Interno - CEETEPS

São Paulo, 28 de março de 2023

A Deus, pela graça e oportunidade de fazer este curso de mestrado profissional.

À minha família, presente em todas as horas, auxiliando nos momentos difíceis, e comemorando nos momentos alegres.

Às pessoas queridas que incentivam e motivam e, ao meu orientador, professor doutor Marcelo Duduchi Feitosa, pelo apoio e orientação.

AGRADECIMENTOS

Agradeço primeiramente, e acima de tudo e de todos, a Deus, que sempre me ajudou, ouviu minhas súplicas e me consolou nas horas mais difíceis, sempre presente, guiando minhas mãos e mente para o desenvolvimento dessa tarefa.

Também sou grata a minha família, que me apoiaram em todos os momentos, sempre presentes até nas horas mais difíceis, me deram o suporte necessário e me incentivam a prosseguir nessa importante etapa da minha vida.

Agradeço às pessoas que se fizeram presentes nesse momento, me ajudando sempre que possível, mostrando que tudo fica mais fácil e mais divertido quando tem-se verdadeiros amigos.

E a todos que de forma direta ou indireta participaram da construção não só deste trabalho, mas da minha vida acadêmica de forma geral.

Por fim, agradeço aos professores que tive o prazer de tê-los como orientadores, ao professor doutor Marcelo Okano e ao professor doutor Marcelo Duduchi, por todo o apoio e suporte na condução desta pesquisa. Agradeço também a todos os professores da casa que, direta ou indiretamente, contribuíram com este trabalho.

Tenho muitos motivos para agradecer!

E um homem verdadeiramente humilde é consciente da diminuta extensão de seu próprio conhecimento, da grande extensão de sua ignorância e da insignificante extensão de seu entendimento comparado com o entendimento de Deus.

Jonathan Edwards

RESUMO

AGUIAR, F. P. L. **Melhoria do teste de software em processo ágil de desenvolvimento de micro serviços**: uma proposta de *roadmap*. 236 F. Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2023.

Esta pesquisa, no contexto do projeto de pesquisa de Tecnologias Digitais em Sistemas Produtivos da linha de pesquisa de Sistemas de Informação e Tecnologias Digitais do Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos do Centro Estadual de Educação Tecnológica Paula Souza, tem como objetivo desenvolver um guia orientador para melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços por meio de um *roadmap*. Para isto, tornou-se necessário a revisão da literatura e pesquisa qualitativa nos temas de engenharia de software, métodos ágeis, teste de software e *roadmap*. Realizou-se também, uma pesquisa de campo que buscou analisar se as práticas de teste de software presentes na literatura são realmente as práticas utilizadas pelos profissionais da área. Elaborou-se então, a construção do RoMTeS, *roadmap* de melhoria de teste de software, composto pelas etapas de Esclarecimentos, Definições, Averiguações, Melhorias e Aferição. Para validação, o RoMTeS foi aplicado em três organizações, uma de pequeno porte, uma de médio porte e uma de grande porte, em que trouxe resultados positivos. Por meio do RoMTeS foi possível identificar ameaças e oportunidades para a melhoria dos testes de software, bem como aplicar planos de ação específicos para cada organização e equipe, considerando seu nível de maturidade em teste de software.

Palavras-chave: Sistemas de Informação. Engenharia de Software. Desenvolvimento Ágil de Software. Arquitetura de Micro Serviços. Teste de Software. Maturidade em Teste de Software. *Roadmap*.

ABSTRACT

AGUIAR, F. P. L. **Improvement of software testing in agile development processes for microservices**: a *roadmap* proposal. 236 F. Dissertação (Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos). Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2023.

This research, in the context of the Digital Technologies in Productive Systems research project of the Information Systems and Digital Technologies research line of the Professional Master's in Management and Technology in Productive Systems at the Paula Souza State Center for Technological Education, aims to develop a guiding roadmap for improving software testing in agile development processes of microservices. To achieve this, it was necessary to review the literature and conduct qualitative research on software engineering, agile methods, software testing, and roadmap. A field research was also conducted to analyze whether the software testing practices present in the literature are actually used by professionals in the field. The RoMTeS, a roadmap for software testing improvement was developed, consisting of the stages of Clarifications, Definitions, Investigations, Improvements, and Verification. To validate it, the RoMTeS was applied in three organizations, one small-sized, one medium-sized, and one large-sized, which brought positive results. Through the RoMTeS, it was possible to identify threats and opportunities for improving software testing, as well as to apply specific action plans for each organization and team, considering their level of maturity in software testing.

Keywords: Information Systems. Software Engineering. Agile Software Development. Micro Services Architecture. Software Testing. Maturity Test Model. Roadmap.

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

ADS	Adaptative Software Development
CMMI	Capability Maturity Model Integration
CI/CD	Continuous Integration/Continuous Delivery
DSDM	Dynamic System Development Method
FDD	Feature Driven Development
ISTQB	International Software Testing Qualifications
MPSBR	Melhoria de Processos do Software Brasileiro
TMMi	Test Maturity Model integration
XP	eXtreme Programming

SUMÁRIO

INTRODUÇÃO

A transformação digital traz diversos benefícios e modernização para a sociedade como um todo. Hoje, a tecnologia encontra-se tão enraizada no cotidiano que se torna difícil imaginar a existência humana sem a sua presença. Os diversos softwares que nos cercam todos os dias, seja para agendar uma reunião de trabalho, para verificar a previsão do tempo e, até mesmo, os softwares utilizados para construção de outros softwares, todos eles mudaram radicalmente nossa forma de viver.

Em meio a este mundo pautado pela tecnologia, contendo sistemas de informação, pode-se perceber a importância do teste de software para suas mais diversas finalidades, seja para alinhar expectativas em relação ao software entregue ser o software que o cliente espera, detectar problemas e defeitos, assim como manter uma estabilidade do sistema antes que estes cheguem ao cliente final.

A questão que norteia este trabalho é, quais aspectos a considerar na promoção de melhorias em realização de teste de software em processos ágeis de desenvolvimento de micro serviços? Esta questão se torna cada vez mais importante, à medida que as melhorias aplicadas no processo de teste de software estão diretamente relacionadas a visão que o cliente/usuário da aplicação terá da organização, além de, estar relacionado à melhoria de processos internos.

Assim, o objetivo geral deste estudo é desenvolver um guia orientador para melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços por meio de um *roadmap* e, como objetivos específicos, descrever as melhores práticas de teste de software encontradas na literatura, verificar se as melhores práticas de teste de software encontradas na literatura são efetivamente usadas no cotidiano das organizações, dando oportunidade para os respondentes indicarem outras melhores práticas de teste de software em desenvolvimento ágil de micro serviços que presenciem nas organizações que atuam, construir o *roadmap* para melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços, contendo as melhores práticas encontradas no primeiro objetivo específico, aplicar o *roadmap* em empresa de pequeno, médio e grande porte, que trabalhem com o desenvolvimento software utilizando arquitetura de micro serviços, avaliar o *roadmap* por meio da sua utilização com entrevistas aos gestores e pesquisa com profissionais que atuam no ambiente de desenvolvimento e testes de software e, comunicar a pesquisa por meio da dissertação e publicação de artigos.

A construção do *roadmap* como guia orientador para melhoria de teste de software se abrange etapas como esclarecimentos, definições, averiguações, melhorias e aferição e, sua

avaliação nas organizações se dá por meio de análise qualitativa, por entrevista com os envolvidos no processo de sua implantação.

O presente estudo se justifica em decorrência ao aumento do uso de sistemas de computadores nos mais diversos segmentos da atividade humana. Assim, há necessidade de assegurar que os sistemas computacionais estão devidamente testados. Dessa forma, o estudo acerca dos softwares mostra-se essencial, estabelecendo novas técnicas, ferramentas, critérios e métodos para corroborar nas fases da produção de software (Coelho et al. 2012).

Para entender quais são as práticas e aspectos necessários para considera-se na promoção das melhorias na realização de teste de software, realizou-se uma pesquisa bibliográfica com o objetivo de se encontrar as principais práticas de teste de software encontradas na literatura. A partir destas práticas selecionadas, realizou-se um questionário com pessoas que atuam no ambiente de desenvolvimento ágil de software para analisar se, as práticas encontradas na literatura realmente são as práticas de teste de software aplicadas nas organizações e, para verificar se existe alguma prática de teste de software comumente aplicada nas organizações, porém, não mencionadas na literatura.

Após esta análise e validação das práticas de teste de software encontradas na literatura e da validação das práticas realmente aplicadas no dia a dia das organizações, realizou-se a construção de um *roadmap* como guia orientador de melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços, que aborda etapas como esclarecimentos, definições, averiguações, melhorias e aferições.

Nomeado de RoMTeS, *Roadmap* de Melhoria de Teste de Software, este *roadmap* foi aplicado em três organizações que desenvolvem seus próprios sistemas. Estas organizações atuam com o desenvolvimento de micro serviços em ambiente ágil de desenvolvimento de software e, são responsáveis pelo desenvolvimento e testes de suas aplicações. Por questões de privacidade, a primeira organização, é mencionada neste trabalho como empresa Alfa, empresa de pequeno porte, atuando no mercado com foco em soluções para análise de áudio, a segunda empresa, mencionada como empresa Beta é uma empresa de médio porte e está atuando no mercado a mais de 20 anos com atuação focada em outsourcing de impressão e, a terceira empresa, é mencionada neste trabalho como empresa Gama, empresa de grande porte atuando a mais de 20 anos no setor financeiro.

A avaliação do RoMTeS se dá por meio de entrevistas com os gestores das organizações mencionadas acima, para análise da melhoria do seu processo de teste de software.

Para descrever este processo, tem-se no primeiro capítulo deste trabalho, a fundamentação teórica, que é responsável pela introdução dos assuntos base para melhor

compreensão deste trabalho. Dentre os principais termos tem-se os sistemas de informação e o processo de desenvolvimento de software, o desenvolvimento ágil de software, métodos ágeis, a arquitetura em micro serviços, o teste de software e a conceituação de *roadmap*.

No Capítulo 2 será abordado a metodologia utilizada neste trabalho, bem como definição de tema, questão de pesquisa, objetivos e a justificativa de importância do tema trabalhado.

O terceiro capítulo apresenta os resultados e discussões, contendo o levantamento bibliográfico, as práticas de teste de software encontradas na literatura, a pesquisa de campo que valida as práticas de teste de software encontradas na literatura como realmente utilizadas no dia a dia das organizações e, trata também, da construção do *roadmap* RoMTeS.

O quarto capítulo trata da descrição do *roadmap* RoMTeS, bem como sua implantação nas três organizações, a descrição do processo de melhoria de teste de software e sua avaliação por parte dos gestores e equipes das organizações que aceitaram sua aplicação.

O quinto capítulo é destinado a implantação do RoMTeS em uma organização de pequeno porte, o sexto capítulo é referente a implantação do RoMTeS em uma organização de médio porte e, o sétimo capítulo é destinado a implantação do RoMTeS em uma organização de grande porte.

O oitavo capítulo refere-se às considerações finais e trabalhos futuros, seguido pelos Apêndices 01 a 04, que são os questionários aplicados para validações, relação do RoMTeS com modelo de maturidade TMMi e, o roteiro de entrevistas com especialistas em teste de software sobre o RoMTeS. O primeiro anexo mostra o questionário para análise de práticas de teste de software presentes na literatura e o segundo anexo mostra o roteiro de entrevista destinado aos gestores das organizações que implantaram o RoMTeS e suas equipes, para realização de análise qualitativa de suas opiniões sobre a implantação do RoMTeS.

1. FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho aborda os sistemas de informação e o processo de desenvolvimento de software, o desenvolvimento ágil de software, a arquitetura de micro serviços e o teste de software e sobre os conceitos de *roadmap*. Temas necessários para a contextualização da pesquisa.

1.1. Sistemas de Informação e o Processo de Desenvolvimento de Software

O processo de desenvolvimento de software é a maneira pela qual produzimos o software abrangendo desde a metodologia com seu modelo de ciclo de vida do software e técnicas adjacentes até as ferramentas utilizadas e a equipe de desenvolvimento que está criando o software (SCHACH, 2010).

Este processo abrange todas as etapas necessárias para produzir um software, desde a escolha da metodologia e modelo de ciclo de vida até as ferramentas e equipe de desenvolvimento envolvidas no projeto (SCHACH, 2010). Esse processo envolve diversas atividades que visam criar um produto de software capaz de automatizar, melhorar ou inovar algum processo no mundo real. Embora existam vários modelos de processo de desenvolvimento de software, todos eles compartilham atividades comuns, como a análise de requisitos, a codificação e a etapa de testes (SOMMERVILLE, 2011).

O processo de desenvolvimento de software consiste em uma série de atividades que têm como objetivo criar um produto de software capaz de automatizar, melhorar ou inovar algum processo realizado manualmente no mundo real. Esse processo envolve atividades que vão desde a análise de requisitos até a codificação e testes. Embora existam vários modelos de processo de desenvolvimento de software que definem a ordem, sequência e intensidade dessas atividades, algumas são comuns a todos os modelos e a qualquer projeto de desenvolvimento de software. Dentre essas atividades, podem ser mencionadas as seguintes (SOMMERVILLE, 2011):

- Especificação de Software: Definição das funcionalidades, requisitos, restrições e de qual a finalidade do software. Geralmente nesta fase do processo, o desenvolvedor interage mais com o cliente a fim de definir as características do software a ser construído.
- Projeto e Implementação: O software é projetado e produzido de acordo com as especificações construídas na etapa de anterior. Nesta fase são feitos modelos e

diagramas que serão implementados em alguma linguagem de programação para a construção do software.

- Validação de Software: O software construído é validado com o usuário final a fim de garantir que todos os itens da especificação foram atendidos.
- Evolução do Software: O software precisa evoluir, para ser útil e customizável para o cliente, sendo assim pode ter novas funcionalidades e com isto o software pode evoluir para algo mais robusto.

O processo de desenvolvimento de software é crucial para a construção de um software eficiente e confiável. Ele estabelece os alicerces para gerenciar o desenvolvimento por meio do uso de técnicas e métodos de engenharia. Quando um modelo (ou metodologia) de desenvolvimento é adotado, ele auxilia diretamente na sincronização, fornecendo a todos os membros da equipe de desenvolvimento uma terminologia comum para tarefas e atividades (AUDY, 2008).

A partir destas definições pode-se considerar que de forma geral um processo de software padrão pode ser visto como um conjunto de atividades, métodos, ferramentas e práticas que são utilizadas para construção de um software. Na definição de um processo de software devem ser consideradas as seguintes informações: atividades a serem realizadas, recursos necessários, artefatos requeridos e produzidos, procedimentos adotados e o modelo de ciclo de vida utilizado (FALBO, 1998).

Uma aplicação de software passa por diversas fases durante o seu desenvolvimento até a sua conclusão final. Cada uma dessas fases pode ser conduzida de diferentes maneiras, formando assim um processo de software.

As fases do processo de desenvolvimento de software são os passos que transformam a ideia no produto final acabado. Estas fases desempenham um papel essencial no processo de desenvolvimento de software, possibilitando ao gerente de projetos controlarem o processo de desenvolvimento. (PETERS, 2001).

Peters (2001) define uma visão tradicional do processo de software que é vista como uma sequência linear de seis atividades: engenharia de sistema, análise de requisitos, projeto, codificação, testes e manutenção.

Para Braude (2004) as principais fases de um processo de software são: análise de requisitos, projeto, implementação, testes e manutenção.

Rezende (2005) define que o processo de software é um ciclo natural de vida e que abrange as seguintes fases: concepção, construção, implantação, implementação, maturidade e utilização plena, declínio, manutenção e morte.

Para Schach (2010) são seis as fases do processo de desenvolvimento: levantamento de necessidades, análise e especificação, projeto, implementação, manutenção e entrega.

Pressman (2011) entende como sendo cinco as atividades genéricas no processo de desenvolvimento de software: comunicação, planejamento, modelagem, construção e emprego.

Embora os diversos autores adotem diferentes número de fases para o processo de desenvolvimento de software, todos concordam sobre as atividades essenciais que compõem esse processo. De acordo com Sommerville (2011), Braude (2004), Pressman (2011), Schach (2010), Peters (2001) e Rezendes (2005), essas atividades são fundamentais para garantir a qualidade do software produzido e incluem desde a definição dos requisitos até a implantação do software, passando pela análise, projeto, implementação e testes.

1.2. Desenvolvimento Ágil de Software

Determinar a metodologia de desenvolvimento de software a ser utilizada em um projeto é uma decisão crítica. Condições de mercado e pressões competitivas requerem que as organizações diminuam tempos de ciclo, reduzam custos e melhorem dos softwares (KAKKAR, 2006).

Por vezes, a decisão de uma metodologia pode ser baseada em um viés de marketing ou literatura, que apoiam práticas novas ou apoiadas pela indústria. Outras vezes, organizações podem contar com padrões já utilizados por questões de consistência e repetibilidade. É improvável que a escolha de uma metodologia de desenvolvimento de software seja uma questão simples ou determinística. Hawrysh e Ruprecht (2000) afirmam que uma única metodologia não é capaz de funcionar para todo o espectro de diferentes projetos, e a gestão do projeto deve definir a sua natureza e selecionar a metodologia de desenvolvimento mais adequada ao projeto em questão.

Nandhakumar e Avison (1999) argumentam que metodologias de desenvolvimento de software tradicionais são tratadas como uma “ficção necessária” para apresentar uma ilusão de controle ou prover um status simbólico, e são mecanicistas demais para serem de utilidade nas atividades detalhadas do dia a dia de um desenvolvedor de sistemas de uma organização. Segundo Awad (2015), profissionais julgavam que esta visão é centrada em processos frustrantes e vivenciavam problemas quando a taxa de mudanças era relativamente baixa. Como

resultado, diversos consultores desenvolveram independentemente metodologias e práticas para aceitar e reagir às mudanças inevitáveis que estavam vivenciando. Essas metodologias e práticas são baseadas em aprimoramentos iterativos e técnicas que foram introduzidas em 1975 e que mais tarde se tornaram conhecidas como métodos ágeis. Segundo Highsmith e Cockburn (2001, p.122):

> O que há de novo no desenvolvimento ágil não são as práticas que eles usam, mas o reconhecimento das pessoas como os principais impulsionadores do sucesso do projeto, associado a um foco intenso na eficácia e manobrabilidade. Isso produz uma nova combinação de valores e princípios que definem uma visão de mundo ágil.

Os métodos ou metodologias ágeis são caracterizados por ciclos de desenvolvimento mais curtos, alta interação com o cliente, entregas incrementais, redesign frequente com acomodação de mudanças necessária pela coleta dinâmica de requisitos dos usuários. Apesar de diversas metodologias de desenvolvimento de software partilharem dos princípios ágeis elaborados pelo Manifesto Ágil, elas diferem umas das outras em diversos parâmetros (MATHARU et al., 2015). Algumas das mais conhecidas metodologias ágeis são: *Scrum*, *Extreme Programming* (em português: Programação Extrema, também conhecido pela sigla XP), Desenvolvimento *Lean* (ou "enxuto), entre outras. Abaixo serão detalhados algumas destas metodologias.

1.3. Métodos Ágeis

O mercado atual é marcado por alta competividade entre as empresas por cliente, levando – as a buscar meios para oferecer customização e rapidez. (ALBUQUERQUE, 2002). Com isso, os intervalos entre lançamentos são reduzidos, de modo que a vida útil dos produtos e serviços também diminuam. Nesse contexto, a inserção de recursos tecnológicos aumenta as possibilidades de as empresas sobreviverem dentro desta realidade ao torná-las capazes de responder às demandas com maior rapidez (JORDÃO, 2004).

Embora exista há algumas décadas, a nomenclatura "Metodologias Ágeis" começou a se popularizar no Brasil recentemente, com a ideia de um método simples. A simplicidade, dentro das metodologias ágeis, contudo, implicam em organização e controle.

Para Highsmith (2004), agilidade diz respeito à criação e adaptação a mudanças, visando alcançar lucro em um ambiente empresarial complexo e agitado. Pode significar ainda ser capaz de manter um equilíbrio entre flexibilidade e a estabilidade. O autor ressalta também que a falta

de estrutura ou segurança pode resultar em turbulências e, por outro lado, o excesso de estrutura provoca rigidez.

No contexto de desenvolvimento de projetos, a abordagem ágil se tornou mais intensa quando, no começo dos anos 2000, 17 autores e especialistas das diversas técnicas e metodologias ágeis, tais como: “eXtreme Programming (XP) , Scrum, Dynamic System Development Method (DSDM), Adaptive Software Development (ASD), Crystal, Feature Driven Development (FDD) e Lean Development, reuniram-se para debater e investigar o modelo de desenvolvimento de projetos considerando as técnicas e metodologias que existiam no mercado.

A partir dessa discussão, foi criado o Manifesto para Desenvolvimento Ágil de Software (Agile Manifesto, 2001), que instituiu um “framework” comum para processos ágeis valorizando os seguintes critérios: indivíduos e interações; funcionamento de “Software”, cooperação dos clientes, respostas às mudanças em detrimento de processos e ferramentas, extensa documentação, negociação em contratos, subordinação a um plano, respectivamente (PEREIRA, *et al.*, 2007).

Diante das constantes mudanças que acontecem no trabalho remoto, as metodologias tradicionais não são mais suficientes para o gerenciamento de projetos e equipes. Nesse contexto, surgem então as metodologias ágeis.

A metodologia ágil deriva da manufatura ágil. Esta surgiu como uma forma de resposta do setor de manufaturas americano às fábricas japonesas durante a década de 1980, que tinham como preceito a eficiência na produção, tendo assim um gasto mais enxuto em sua produção. Houve, portanto, uma grande preocupação por parte do congresso americano devido à queda da lucratividade e perda de espaço no mercado internacional, sendo necessária a criação de um programa federal que se tornou a base da manufatura ágil (FRANCO, 2007).

A manufatura ágil foi um conjunto integrado de estratégias de negócios, modelos, estudos e melhorias em práticas de gestão desenvolvidos durante seis anos de pesquisas pelo *Agility Fórum de Iacocca Institute*, entre os anos de 1991 e 1998 com foco nas 150 empresas líderes de diversos setores influentes na economia americana como: Boing, Goodyear, IBM, Kodak, Texas Instruments e Xerox (FRANCO, 2007).

No início do ano de 2001, o movimento ágil se formalizou dentro da indústria por meio do “Manifesto para ao desenvolvimento ágil de software”, também comumente conhecido como o “Manifesto Ágil”. O documento teve como intuito defender a necessidade de apresentar novas formas de desenvolver softwares, em contraponto aos modelos tradicionais (FRANCO, 2007).

As metodologias tradicionais, que costumam ser mais fechadas e direcionadas a planejamentos, são eficientes em situações em que as condições do sistema apresentam estabilidade e previsibilidade. Em realidades sujeitas a modificações e adaptações recorrentes, as abordagens estáticas não se mostram eficientes, exigindo a adoção de metodologias ágeis (SOARES,2004).

Grande parte das metodologias ágeis não são totalmente inovadoras (COCKBURN *et al.*,2001). O ponto de distinção entre estas e os métodos tradicionais são os valores e o enfoque. As metodologias ágeis priorizam as pessoas, e não mais os processos.

Como pode ser visto por meio do "Manifesto Ágil" que descreve diversos princípios e valores, que conforme os estudos realizados, seriam relevantes na forma de desenvolver softwares. A metodologia ágil não busca a previsão do futuro, mas aceita a mudança e, sendo flexível, adequa-se às transformações.

Conforme apontado por Beck et. al (2011), os modelos ágeis possuem infinitos processos em diversos formatos, porém todos regidos pelos mesmos princípios (Quadro 1), de modo a garantir que tenham como ideal de funcionamento um processo com o mínimo de burocracias e o máximo de fluidez.

Quadro 1: Princípios do Software Ágil

A maior prioridade deve ser gerar satisfação do cliente por meio de uma entrega sem atrasos e contínua de um software com valor.
Estar pronto para modificações mesmo que estas ocorram ao fim do desenvolvimento. É importante entender que processos ágeis são resilientes e que o foco principal deve estar nas vantagens competitivas que o cliente deve obter com o sistema
Realizar a entrega de um software com a garantia de funcionabilidade, realizando verificações e manutenções periódicas. Tendo como base uma escala de semanas e meses.
O projeto deve ser desenvolvido juntamente às pessoas relacionadas ao negócio diariamente durante todo o curso do projeto.
É necessário que a construção do projeto seja realizada por pessoas engajadas a ele. Devem ser colocados à disposição o ambiente e o suporte necessário para isso, além de se criar vínculos de confiança baseados em trabalho.
A forma mais eficiente e eficaz de se compartilhar informações dentro de um time é em uma conversa franca entre indivíduos.
O software funcional é a premissa de progresso.
A sustentabilidade do ambiente depende de um processo ágil que conte com a capacidade de patrocinadores, desenvolvedores e usuários em se manterem de forma indefinida em passos constantes.
O foco na excelência técnica e em um bom design quando feita de forma contínua tende a aumentar a agilidade.
A simplicidade deve ser vista como a chave para maximizar a quantidade de trabalho que não precisou ser realizado
Times auto-organizáveis tendem a criar melhores arquiteturas, requisitos e projetos.
É necessário que sejam realizadas constantes reavaliações em equipe para verificar a efetividade do trabalho. Posteriormente devem, em conjunto, se ajustar e otimizar seu comportamento.

Fonte: Beck *et al.* (2011)

Seus valores, portanto, podem ser sintetizados em quatro premissas:

- As interações entre os indivíduos mais que processos e ferramentas;
- Softwares com foco em funcionalidade mais que documentação;
- Movimento de colaboração junto ao cliente mais que negociações com foco no contrato;
- Ser responsivo a mudanças mais que seguir um plano pré-determinado. (RUSSO, *et*. al, 2021)

Ao que se compreende sobre metodologias ágeis, nota-se que estas foram criadas com o intuito de diminuir os processos burocráticos e dar foco nas pessoas e em seus resultados aos clientes. Desse modo, tais metodologias podem conferir maior eficiência na gestão de projetos de software, o que torna esse trabalho relevante.

Dentre as metodologias comumente utilizadas o Scrum tem destaque e será apresentado nesta seção de forma detalhada. Compreende-se por uma estrutura de desenvolvimento de software ágil incremental e iterativa para gerenciar produtos em desenvolvimento. Foi definido pela primeira vez como "uma estratégia de desenvolvimento de produto flexível e holística, com uma equipe de desenvolvimento funciona como uma unidade para alcançar um objetivo comum" em 1986 por Hirotaka Takeuchi e Ikujiro Nonaka no *New Product* Jogo de Desenvolvimento.

Esta estratégia desafia os pressupostos da "abordagem tradicional e sequencial" para o produto em desenvolvimento e incentiva as equipes a se auto-organizarem, incentivando a estreita colaboração online de todas as equipes e membros, bem como a comunicação face a face entre todos os membros da equipe e disciplinas do projeto (JAKOBSEN; SUTHERLAND, 2009).

Um princípio-chave do Scrum é a “volatilidade dos requisitos”, ou seja, reconhece o fato de que durante os processos de produção, os clientes podem mudar de ideia sobre o que querem e precisam. Esses desafios imprevistos não podem ser facilmente abordados de maneira preditiva ou planejada tradicional e, portanto, é uma vantagem do Scrum/ágil.

Para Sutherland (2005), o Scrum assume uma abordagem empírica - aceitando que o problema não pode ser totalmente definido, em vez disso concentra-se em responder aos requisitos emergentes e se adaptar às tecnologias em evolução e às mudanças no mercado. Além disso, é capaz de empregar processos de tomada de decisão em tempo real com base em eventos e informações reais. Isto exige equipes especializadas e capazes de autogestão, comunicação e tomada de decisão.

Trata-se de uma técnica ágil que pode ser aplicada a praticamente qualquer projeto; no entanto, é mais comumente utilizada em software de desenvolvimento. O processo é adequado

para projetos com mudanças rápidas ou altamente emergentes (JAKOBSEN; SUTHERLAND, 2009).

Schwaber (2002) descreve o Scrum como um framework que pode ser usado para construir produtos complexos. É "uma estratégia flexível de desenvolvimento de produto". Ele não prescreve nenhuma das práticas comuns de engenharia, pessoas, gerenciamento de riscos ou outras práticas. É possível e às vezes desejável também, considerar práticas não-Scrum que podem estar intimamente ligadas ao sucesso desta metodologia.

Diversos estudos comprovam a eficiência da aplicabilidade, provando ser bem-sucedido para projetos ágeis, mesmo que não seja uma prática explícita do Scrum. É frequentemente implementado em conjunto com métodos como *Extreme Programming* que influencia em seu sucesso. O que o Scrum fornece é o *feedback* para que seja possível melhorar resultados (WOODWARD, 2010).

Por exemplo, se alguém quer produtividade pode ter uma equipe co-localizada, o Scrum permite identificar a melhor maneira possível. Se, porventura, o processo é iniciado com uma equipe dispersa e compara sua produtividade com outra co-localizada, as conclusões podem ser alcançadas. Dessa forma é possível tomar decisões inteligentemente com base nas conclusões recebidas e pode realizar alterações de acordo com seu desempenho (SUTHERLAND, 2005).

Assim, a especialidade Scrum está na melhoria contínua do processo. Quando utilizado corretamente, ou seja, seguindo todos seus requisitos, para inspeção e adaptação. Notoriamente, um gestor então buscaria averiguar o que o Scrum está tornando transparente e assim, faria mudanças para otimizar os resultados. Presumivelmente, as mudanças são justificadas pelo custo (JAKOBSEN; SUTHERLAND, 2009).

Considerando o Scrum como um framework para gestão de projetos sua aplicabilidade é impulsionada com o uso de ferramentas, garantindo o comprometimento de todos com os pilares do Scrum, que são a transparência, a inspeção e a adaptação (KELLY, 2012).

Para que as empresas fossem competitivas, elas precisavam reduzir seus custos, acelerar o desenvolvimento de produtos e focar na satisfação de seus clientes. Conhecimento em gerenciamento de projetos focado em habilidades essenciais nas áreas de orçamento, programação e alocação de recursos. As ferramentas de gerenciamento de projetos são fundamentais para auxiliar nesse processo, pois possuem recursos de planejamento, programação, alocação de recursos, comunicação e documentação (KELLY, 2012).

Gerenciar projetos e equipes às vezes pode parecer esmagador. Felizmente, existem várias soluções gratuitas de gerenciamento de projetos para ajudar os líderes de projeto a

gerenciar seu fluxo de trabalho. Uma dessas soluções é o Trello, uma ferramenta baseada em nuvem que utiliza o método Kanban de gerenciamento de projetos (JOHNSON, 2017).

Com o Trello, os usuários podem organizar visualmente projetos em quadros, dividir projetos em grupos e subdividir grupos em tarefas. A interface amigável do Trello o torna ideal para uma ampla variedade de usuários, desde indivíduos gerenciando projetos pessoais, como reformas de casas, até organizações gerenciando vários grandes projetos e equipes.

Johnson (2017) afirma que o Trello requer apenas uma conexão com a Internet, eliminando a necessidade de os usuários instalarem software ou inserir chaves de produto. Os usuários registrados podem criar um número ilimitado de quadros e designar um quadro por projeto. Os usuários podem então atribuir vários grupos de tarefas (listas) a cada quadro e atribuir subgrupos (cartões) a cada lista. Os usuários podem criar cartões adicionando-os manualmente ou copiando e colando listas de texto existentes do Microsoft Word ou Excel.

Os usuários podem coordenar facilmente as equipes e atribuir cartões aos membros da equipe (membros do conselho) enviando e-mails de convite diretamente do quadro do projeto. Uma vez que um membro do conselho tenha recebido um cartão, o cartão refletirá a designação e as iniciais do membro do conselho aparecerão no cartão.

Outra ferramenta que apresenta grande performance é o Slack, desempenhando um papel cada vez mais significativo na forma como os desenvolvedores colaboram e se comunicam. Em apenas dois anos, conquistou mais de 1,1 milhão de usuários diários e mais de 900 mil integrações. O Slack não apenas facilita as mensagens e o arquivamento da equipe, mas também oferece suporte a uma ampla variedade de integrações com serviços e bots externos (SINGH, 2008).

Entender como e por que os desenvolvedores usam o Slack para apoiar seu trabalho é essencial para obter *insights* sobre software moderno, práticas de desenvolvimento e seus aspectos colaborativos. A mídia social revolucionou a maneira como os desenvolvedores trabalham, colaboram e se comunicam (DABBISH, 2012).

Singh (2008) ressalta que o Slack é uma plataforma de comunicação moderna para equipes que está sendo adotada de forma ampla e rápida pelas equipes de desenvolvimento de software. O Slack e suas integrações estão desempenhando um papel cada vez mais significativo no desenvolvimento de software, substituindo o e-mail em alguns casos e interrompendo os processos de desenvolvimento de software.

O software Asana é outra modalidade de plataforma móvel e aplicativo desenvolvido para simplificar o gerenciamento de trabalho em equipe, tarefas e projetos ajudando a organizar,

controlar e gerenciar o projeto. Foi desenvolvido em 2008 pelo cofundador do Facebook Dustin Moskovitz e ex-Google e o engenheiro do Facebook Justin Rosentein (KELLY, 2012).

Para Kelly (2012) a ideia veio da necessidade de melhorar a produtividade de seus funcionários. Esta ferramenta permite que as equipes criem projetos, atribuam tarefas aos membros da equipe, especifiquem prazos, comuniquem tarefas, gerem monitoramento de progresso, relatórios, anexos de arquivos, calendários etc.

A Asana passou por várias mudanças desde seu lançamento e a versão em português foi lançada em fevereiro de 2018. Este aplicativo possui integrações com outras ferramentas como Gmail, Microsoft Outlook, Dropbox, Google Drive, entre outros. As principais vantagens deste software consistem em uma segmentação e priorização de projetos; permite gerenciar para cada projeto suas permissões; *dashboards* personalizáveis; possui um portal de comunicação; o armazenamento é seguro; interface simples e intuitiva (LE, 2014).

Assim como as demais, PangoScrum é uma ferramenta capaz de gerenciar projetos ágeis, deve-se acrescentar que teste ágil é um termo que deriva da palavra teste e podemos compreender como serviço que atua ou se desenvolve com rapidez ou presteza. Esse *software* foi projetado com a ajuda de ferramentas sofisticadas que possuem as funcionalidades de medir e avaliar. A ideia de testes ágeis como processo é mostrar anomalias em seu desenvolvimento, minimizando a chance de o usuário de detectá-los em uso. O PangoScrum fornece métodos e funcionalidades que podem encontrar algum defeito ou vulnerabilidade em programas (GARCÍA RAMÍREZ, 2016).

1.4. Arquitetura de Micro Serviços

Newman (2015) discorre sobre serviços autônomos que trabalham em conjunto e que, na última década, os sistemas distribuídos se tornaram mais refinados, avançando de aplicações monolíticas de código pesado para micros serviços independentes.

O termo micros serviços é definido por Thönes (2015) como um *app* que pode ser implantado, dimensionado e testado de forma independente, seguindo o princípio da responsabilidade única. O propósito da responsabilidade única é que o aplicativo desenvolvido deve ser planejado para realizar apenas um conjunto mínimo viável de tarefas, sendo facilmente compreendido, modificável e substituível.

Os micros serviços devem ter foco, serem pequenos e realizarem uma única tarefa por conta própria, decompondo de forma funcional a aplicação em um conjunto colaborativo de serviços, em que cada serviço implementa um conjunto de funções relacionadas de forma

restrita, para implementar as regras de negócio. O estilo arquitetônico baseado em micros serviços é uma abordagem para o desenvolvimento de uma aplicação única, embasada em um conjunto de micros serviços, cada um desenvolvendo em seu próprio processo e se comunicando via um mecanismo leve, por vezes, uma API de recursos HTTP (FOWLER; LEWIS, 2014).

Segundo Fowler e Lewis (2014), o termo micros serviços se manifestou nos últimos anos para determinar um meio peculiar de desenvolver aplicações de software independentemente implementáveis, como uma suíte de serviços, embora este estilo de desenvolvimento ainda não esteja precisamente definido. Com os serviços implementáveis e escaláveis, é possível desenvolvê-los em diversificadas linguagens de programação e mantê-los e gerenciá-los por equipes diversas.

Newman (2015) define os princípios e objetivos estratégicos em conjunto às práticas de *design* e *delivery* para o desenvolvimento de software baseado em micros serviços, defendendo a construção de serviços autônomos pequenos que trabalham em conjunto. Os princípios dos micros serviços são:

- A modelagem deve voltar seu foco para o domínio de negócio;
- Partir da cultura da automação;
- Consagrar os detalhes da implementação;
- Dissociar todas as coisas;
- Dissipar as falhas;
- Deploy independente;

Os princípios abordados acima têm como objetivo, de acordo com Newman (2015), minimizar a inércia, realizando escolhas que beneficiem o *feedback* e mudanças rápidas e reduzindo as dependências entre as equipes. Para eliminar a complexidade acidental deve-se substituir os processos complexos, desnecessários e principalmente as integrações também demasiadamente complexas com o propósito de privilegiar o desenvolvimento.

Tanto a análise baseada em modelo quanto em gráfico introduz uma abstração ou uma representação intermediária do sistema para capturar certas preocupações. Embora possa haver uma sobreposição ou um gráfico possa ser entendido como um modelo, consideram as duas categorias separadas. Tanto a análise baseada em modelo quanto em gráfico introduz uma abstração ou uma representação intermediária do sistema para capturar certas preocupações do sistema (KNOCHE; HASSELBRING, 2018).

1.5. Teste de Software

Sommerville (2003) explica que o teste de um software é medido pelo grau em que atende seus requisitos, seu comportamento quando em funcionamento, a estrutura e a organização do programa fonte e a documentação associada. Há muitos atributos de teste de software, como: segurança, confiabilidade, documentação precisa e completa, capacidade de recuperação, robustez, testabilidade, modularidade, portabilidade, eficiência e facilidades de uso, de compreensão, de adaptação e de aprendizado. No geral, não é possível para qualquer sistema ser otimizado em relação a todos estes atributos, logo, deve-se definir quais deles são mais significativos para o produto a ser desenvolvido

Para este trabalho, foi considerada a definição de teste de software como parte integrante da vida, ou seja, pois compreende desde a concepção comercial até o consumo. Em suma, grande parte das pessoas já desfrutaram de experiências com algum tipo de software que não funcionaram de maneira adequada ou esperada. Quando isso ocorre, não resulta apenas na insatisfação do usuário, mas sobretudo em uma vasta imensidão de problemas como a perda de investimento, queda de reputação comercial e outros que podem estar associados a vida.

Para o International Software Testing Qualifications Board (ISTQB) (2011), o teste de software atua como um método avaliador de controle a fim de atenuar a probabilidade e recorrência de risco e falha em operação. É comum que ocorram equívocos a respeito da compreensão do teste ser considerado apenas um executor para verificar seus resultados. Muito mais além do que isso, o teste de software abrange um rol de estratégias que englobam desde a execução até a verificação.

Ademais, cabe salientar que também incluem tarefas como planejamento, análises, modelagem e até mesmo implementação de outras ferramentas, perspectiva de resultados, avaliação, relatórios de progresso. A ISTQB (2011) ainda ressalta que alguns testes podem envolver diversos componentes de um sistema que está sendo avaliado, atribuindo-o o nome de teste dinâmico. Diferentemente destes, alguns outros testes que não englobam a execução dos componentes ou sistemas que estejam sendo testados, são chamados de testes estáticos.

Os testes de *software* são técnicas de validação dinâmica, pois têm como um dos objetivos avaliar se o comportamento está de acordo com o esperado (BARTIE, 2002). Os testes de *software* podem ser definidos como uma série de atividades que expõem a presença de defeitos. Para que os testes sejam eficazes não basta apenas que eles sejam executados, é importante que sejam usadas estratégias para que eles sejam competentes (PRESSMAN, 2011).

O teste de software abrange uma série de objetivos no desenvolvimento de um software, como por exemplo: requisitos não funcionais, capacidade de manutenção, reutilização de código, estabilidade, desempenho e usabilidade (ENGHOLM, 2010).

O teste é parte fundamental no ciclo de vida de um software, e há sete princípios que devem ser seguidos segundo o International Software Testing Qualifications Board (2011), o primeiro princípio presume que os testes apontam a presença de falhas, isso porque testes conseguem identificar a existência de falhas, mas não pode garantir a ausência delas. Mesmo se nenhum erro for identificado em uma bateria de testes, não é possível afirmar que o software está livre de falhas.

O segundo princípio defende que o "teste exaustivo é impossível" a menos que a aplicação sendo testada tenha uma estrutura lógica muito simples e valores de entrada limitados, teste exaustivo é inviável pois seria extremamente custoso cobrir todos os cenários de execuções possíveis. Deve-se calcular o esforço dos testes baseando-se nos riscos e prioridades.

O terceiro princípio compreende o "teste antecipado" que nada mais é do que desenvolver um software, as atividades de teste devem começar o quanto antes. Assim que os requisitos ou modelagem do sistema estiverem prontos, é possível começar o trabalho de modelagem do plano de testes. O quanto antes uma falha for identificada no ciclo de vida de um sistema, mais barata e mais simples será a correção.

O quarto princípio visa tratar do agrupamento de falhas tendo em vista que a maioria das falhas encontradas durante a execução dos testes está concentrada em um número pequeno de módulos. Sempre existe uma área do software que é responsável pelo maior número de erros.

O quinto princípio trata do paradoxo do pesticida**.** Um conjunto de testes, se executado várias vezes, pode não mais detectar novas falhas. Para contornar esse problema, os casos de teste devem ser frequentemente revisados e atualizados. Eles devem ser reformulados para abordar novas áreas do sistema e assim aumentar a chance de detectar novas falhas.

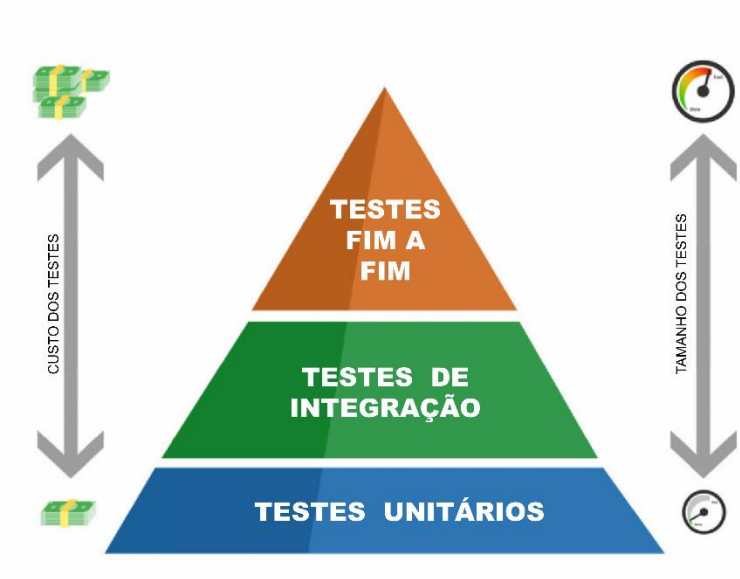
O sexto princípio compreende o cenário que o teste depende. Os testes devem ser elaborados de acordo com o tipo do software. Por exemplo, um sistema bancário deve ser testado de maneira diferente de uma rede social. Há questões de segurança que devem ser mais precisamente abordadas no primeiro caso. Da mesma forma que testes web são elaborados com foco diferente dos testes de aplicações desktop.

Por fim, o sétimo e último princípio discorre que a "ausência de erros é uma ilusão". Identificar e corrigir os problemas de um software não garantem que ele está pronto. Os testes foram elaborados para identificar todas as possíveis falhas? O sistema atende às necessidades e expectativas dos usuários?

A automatização de testes para o software requer a aplicação de padrões de métricas, para que se tenha em mente o que deve ser automatizado ou não: a pirâmide de teste. Mike Cohn (2009) surgiu com esse conceito em seu livro *Succeeding with Agile*, e trata-se de uma espécie de metáfora visual que suscita a compreensão e percepção de diferentes camadas de teste e determina quantos testes devem ser feitos em cada camada. A pirâmide de teste original de Mike Cohn determina que o conjunto de testes consista em três camadas, sendo elas: os testes de unidade, os testes de serviço e os testes de interface do usuário.

A pirâmide de testes demonstra uma maior quantidade de testes automatizados realizados na base, e menos testes conforme a pirâmide vai chegando ao topo (CAMPOS, 2019), conforme pode ser observado na Figura 1.

Figura 1: Pirâmide de testes



Fonte: Adaptado de Cohn (2019).

Devido à sua simplicidade, como observado na Figura 1, a essência da pirâmide de testes serve como uma regra no que tange a estabelecer seu próprio conjunto de testes. Sua melhor aposta é lembrar duas coisas da pirâmide de teste original de Cohn: escrever testes com granularidade diferente e quanto mais alto nível você obtiver, menos testes automatizados deve ter. Dadas as deficiências dos nomes originais, não há problema em criar outros nomes para suas camadas de teste, contanto que o mantenha consistente em sua base de código e nas discussões de sua equipe.

Iniciando pela base da pirâmide, os testes de unidade são aqueles de responsabilidade do desenvolvedor, mas há organizações que outorga essa tarefa aos analistas de testes para que

eles colaborem com o desenvolvedor. O teste consiste na separação do código em pequenas partes para avaliar se elas funcionam de forma separada (ZARELLI, 2020). Os testes de unidade podem ainda ser divididos em componentes maiores, que compreenda unidades coesas. Esses testes são desempenhados com a ajuda de ferramentas de depuração (SWEBOK GUIDE, 2004).

Em outro degrau, os testes de integração têm como objetivo avaliar se as unidades funcionam de forma integrada. É importante a realização dessa etapa para inibir que os defeitos passem para o próximo nível (ISTQB, 2018). Por fim, o teste *End to End* (E2E), ou em português, testes de Fim a Fim, que acontecem com o software executado em um âmbito próprio para a realização. São testes mais custosos de se criar e de se manter, porém são aqueles que garantem de fato a experiência do usuário que está consumindo seu serviço, pois eles simulam o ambiente real, isto é, usam a aplicação da forma que o usuário faria. Na maioria das equipes, são desenvolvidos pelo analista de testes e têm como objetivo simular o comportamento de um usuário (CAMPOS, 2019).

Com base na literatura do ISTQB (International Software Testing Qualifications Board)**,** os níveis de teste são grupos de atividades de teste que são organizados e geridos em conjunto. Cada nível de teste é uma instância do processo de teste, e são divididos da seguinte forma: teste de unidade, teste de integração e teste do sistema (ISTQB, 2018).

O Teste de unidade ou de módulo, é responsável pela verificação do funcionamento da menor unidade de um código testável da aplicação, independente da interação dela com outras partes do código. Esse tipo de teste é frequentemente realizado de forma isolada ao resto do sistema. É possível utilizar objetos simulados, virtualização de serviços, simuladores e controladores. O teste unitário pode cobrir funcionalidade (por exemplo: correção de cálculos), características não funcionais (por exemplo, busca de vazamentos de memória) e propriedades estruturais (por exemplo: teste de decisão) (CAMPOS, 2019).

Os testes de integração são esquecidos no momento de escrever testes para um projeto. Mesmo testando duas unidades que interagem entre si separadamente, usando *mocks*, virtualização, e concluindo que ambas estão funcionando como esperado. Ainda assim, é possível que as duas unidades não funcionem bem em conjunto**.** Os testes de integração são mais complicados de desenvolver, manter e mais lentos que os testes de unidade, pois eles testam funcionalidades inteiras, por vezes com persistência de dados.

Realizar testes de integração não é testar a lógica das unidades, mas testar como as diferentes unidades interagem entre si. Por outro lado, testes de integração são bem mais simples de desenvolver, manter e são bem mais rápidos que os testes de ponta a ponta.

O teste de sistema se concentra no comportamento e nas capacidades de todo um sistema ou produto. Geralmente considerando as execuções das tarefas de ponta a ponta do sistema e os comportamentos não funcionais exibidos ao executar tais tarefas. O teste do sistema geralmente produz informações que são usadas pelos *stakeholders* para tomar decisões de liberação. O teste do sistema também pode satisfazer requisitos ou padrões legais e regulatórios. Esse tipo de teste é o mais comum de se implantar, pois simula a experiência do usuário. São, portanto, muito importantes, pois são os testes mais próximos do que o usuário realmente vai encontrar ao utilizar a aplicação (KAUR, 2012).

Geralmente esse tipo de teste é realizado no ambiente que antecede o ambiente de produção. O teste de sistema deve focar no comportamento geral, funcional e não funcional, de ponta a ponta do sistema como um todo.

As saídas do desenvolvimento do produto ao longo dos estágios de análise, projeto e codificação devem ser medidas, observadas e gerenciadas para que possam ser verificadas em relação a critérios pré-especificados. Além disso, os esforços de desenvolvimento de produtos devem ser atualizados em cada estágio para reduzir custos e manter ou melhorar a competitividade do mercado (KNOCH, 2018).

Para que os testes sejam realizados, é necessário que se saiba em que serão aplicados, e, por isso, é necessário compreender como são classificadas essas características (ISTQB, 2018).

É possível executar qualquer um dos tipos de teste mencionados acima em qualquer nível de teste**,** mas não é necessário ter todos os tipos de testes representados em todos os níveis.

O teste manual significa testar um aplicativo manualmente por um ser humano. Um especialista testador que executa testes manuais garante que um aplicativo esteja funcionando corretamente seguindo as condições descritas nos casos de teste. O testador avalia o *design*, a funcionalidade e o desempenho do aplicativo verificando vários elementos. Testes manuais são recomendados quando se utilizam testes exploratórios, testes de usabilidade e testes de aceite. O teste manual é muito útil quando não é possível implementar o teste automatizado, ou quando os custos de automatização são proibitivos (CAMPOS, 2019).

A importância das medições SQA é evidenciada em normas internacionais como a ISO 9000-3 que é uma diretriz para o desenvolvimento de software, e a ISO 9001 (ISO,2000). Embora tais padrões e modelos destacam a importância das medições de software, faltam diretrizes detalhadas sobre a realização de medições SQA e os objetivos de tais programas de medição

As falhas do produto podem ser evitadas a partir da realização de medições de software. Essas medições alertam os desenvolvedores e engenheiros sobre erros e, assim, os mesmos

podem ser evitados antes e durante os estágios iniciais do lançamento do produto, além de auxiliar no monitoramento do desenvolvimento de software (STAVRINOUDIS; XENOS, 2000).

1.6. Maturidade em Teste de Software

O termo "maturidade" refere-se à ideia de que empresas maduras realizam as coisas de forma sistemática e eficiente (SIQUEIRA, 2005). Um modelo de maturidade funciona como um guia para a empresa, permitindo que ela avalie sua posição atual e crie um plano para alcançar um nível superior de excelência. Ao seguir um modelo de maturidade, a empresa pode identificar áreas de melhoria e implementar mudanças graduais e contínuas que levam a um aumento na qualidade e eficiência de seus processos (SIQUEIRA, 2005). O conceito de modelos de referência é fundamental para que as empresas possam caracterizar seus processos de forma estruturada. Esses modelos são elaborados com base em melhores práticas e normas, que servem como um guia para que a empresa possa identificar seus pontos fortes e fracos e traçar um plano de melhoria contínua. Embora os modelos de referência ofereçam diretrizes sobre as práticas que devem ser adotadas, eles não definem a forma como o processo será integrado na empresa, ficando a cargo da organização escolher como melhor aplicá-los em seu contexto (CMMI, 2023).

De acordo com o CMMI (2023), esses modelos, além de corroborar com o desenvolvimento de processos, possibilitam que as empresas tenham maturidade para analisar o processo que já está em uso na busca por progressos. Desenvolvido em 1997, pela Carnegie Mellon University em parceria com a Systems Engineering Institute - SEI, o CMM foi o primeiro modelo de maturidade criado e teve grande impacto na melhoria dos processos de desenvolvimento de software. Ele oferece um meio de avaliar o processo de desenvolvimento de software fornecendo diretrizes de melhores práticas que podem ser adotadas (CMMI, 2023).

O CMMI (Capability Maturity Model Integration) é uma evolução do CMM que tem como objetivo ajudar as organizações a melhorar o desenvolvimento e a manutenção de processos de produtos e serviços.

De maneira geral, o CMMI é um modelo de referência que tem como objetivo ajudar as organizações a melhorar seus processos de desenvolvimento de produtos e serviços. Ele fornece diretrizes e melhores práticas para avaliar, desenvolver e aprimorar os processos de negócios,

resultando em produtos e serviços mais eficientes e de maior qualidade e é utilizado mundialmente como um padrão de referência para a gestão de processos.

Ao analisar os modelos de maturidade em teste de software, torna-se importante mencionar o modelo de processo de software brasileiro (MPS.BR). O MPS.BR é um modelo de melhoria de processo de software desenvolvido no Brasil, com base em normas e padrões internacionais, que visa aprimorar a qualidade e a maturidade dos processos de software das organizações. Ele é dividido em sete níveis de maturidade, com critérios de avaliação para cada nível, e apresenta um conjunto de processos que devem ser implementados para que a organização possa atingir um determinado nível de maturidade.

Base no CMMI, o MPS.BR surgiu em dezembro de 2003 como necessidade de melhoria de processo do software brasileiro (SOFTEX, 2023). Ele define um modelo de referência para melhoria e avaliação de processos de software de forma a atender as necessidades de negócio de empresas brasileiras (SOFTEX, 2011).

O MPS.BR oferece uma maneira mais adequada a realidade das organizações nacionais, comparada com os padrões internacionais como o CMMI. O MPS.BR tem como foco o desenvolvimento de software de forma geral, incluindo também, a disciplina de testes de software, semelhante ao modelo CMMI (SOFTEX, 2023).

O modelo de referência MPS.BR está organizado em guias de implementação que indicam o que deve ser implementado, além de um guia geral que descreve o modelo. Uma vez que é crescente o número de empresas que fornecem serviços exclusivamente em teste de software, o MPS.BR oferece um guia de implementação específico para organizações do tipo fábrica de teste, nome comumente destinado a organizações que atuam oferendo o serviço de teste de software (MPS.BR, 2011). Além disso, foi criado um modelo exclusivamente para processo de teste, o Melhoria do Processo de Teste Brasileiro (MPT.BR, 2011).

Outro modelo de maturidade complementar ao CMMI é o Modelo de Maturidade em Teste de Software Integrado (TMMi), este modelo, de acordo o TMMi Foundation (2010) é um modelo de maturidade para processos de teste de software, que fornece uma estrutura para avaliar e melhorar a qualidade dos processos de teste e tem como objetivo oferecer diretrizes e melhores práticas para o gerenciamento de testes de software em diferentes estágios de maturidade.

O TMMi possui cinco níveis de maturidade, que vão desde o inicial, onde as atividades de teste são informais e não padronizadas, até o otimizado, onde os processos de teste são contínuos e altamente otimizados. O modelo é amplamente utilizado em todo o mundo para melhorar a qualidade do processo de teste e aumentar a eficácia dos testes de software.

Cada nível de maturidade é composto de algumas áreas de processos que indicam onde uma organização deve focar para melhorar o seu processo de teste e, consequentemente, atingir o nível de maturidade desejado. Áreas de processos identificam as questões que devem ser satisfeitas para que se possa alcançar determinado nível de maturidade. Vale ressaltar que, para cada nível, todas as áreas de processos devem ser satisfeitas e cada área de processo identifica um conjunto de atividades de teste que devem ser realizadas. A imagem do TMMi pode ser observada por meio da Figura 2.

Figura 2: Modelo de Maturidade em Teste de Software TMMi

Fonte: Adaptado de TMMi (2023).

Como observado na Figura 2, o TMMi fornece um caminho para avaliar e melhorar a maturidade dos processos de teste de software em uma organização. O modelo é dividido em cinco níveis, sendo que cada nível representa um estágio de maturidade crescente nos processos de teste.

O primeiro nível é o nível inicial, no qual a organização não tem processos de teste definidos e os testes são executados de forma ad hoc e não planejada.

O segundo nível é o nível gerenciado, no qual a organização começa a implementar processos de teste mais estruturados e documentados, embora ainda não estejam totalmente integrados em toda a organização.

O terceiro nível é o nível definido, no qual a organização já possui processos de teste bem definidos, documentados e integrados em toda a organização.

O quarto nível é o nível medido, no qual a organização já possui indicadores de desempenho de teste e está continuamente monitorando e melhorando seus processos de teste com base nesses indicadores.

Por fim, o quinto nível é o nível otimizado, no qual a organização já possui processos de teste altamente maduros e otimizados, e está continuamente buscando novas formas de melhorá-los.

1.7. Roadmap

Genericamente, um *Technologic Roadmap* (TRM), que mais bem traduzido para a literatura da língua portuguesa figura como Mapa Tecnológico, trata-se de um artefato visual, na grande maioria das vezes, orientado dentro de um espaço de tempo, em diferentes perspectivas, principalmente comercial e tecnológica, se relacionam (PHAAL, 2004). O termo *Roadmapping* é um neologismo em inglês que, segundo Bray e Garcia, (1997), designa um processo de planejamento tecnológico para identificar, selecionar e desenvolver as alternativas tecnológicas que atendessem ao conjunto de necessidades de produção das empresas. Em outras palavras, *roadmapping* refere-se ao conjunto de ações e tarefas que constituem o processo por meio do qual é produzido o artefato visual denominado *roadmap*.

Com a popularização do termo TRM, este se tornou um substituto, ou até mesmo uma metáfora, para o que a maior parte da literatura sempre se referiu como planejamento de recursos para ciência e tecnologia (KOSTOFF e SCHALLER, 2001). O *roadmap* nada mais é, portanto, que o artefato visual resultante do *roadmapping*, isto é, o processo de criação do *roadmap* em si.

Para Phaal et al. (2004), que abordam o tema de maneira mais específica, o conceito de *roadmap* genérico pode ser definido basicamente como um gráfico contendo uma linha temporal, em que algumas camadas são comprimidas incluindo uma perspectiva comercial e tecnológica. Mais especificamente, o TRM permite que a evolução de mercados, produtos e tecnologias possa ser explorada de maneira conjunta com links e descontinuidades entre as várias perspectivas apresentadas.

Bray & Garcia (1997) atentam para a importância de compreender que o processo de *roadmapping* é obrigatoriamente orientado por uma necessidade e não por uma solução e que, portanto, o *roadmapping* seria um processo de planejamento tecnológico orientado à necessidade com o intuito de auxiliar a identificar, selecionar e desenvolver alternativas tecnológicas que satisfaçam um conjunto de necessidades de um determinado produto.

De forma prática e direta, o processo de *roadmapping* pode ser entendido pela definição trazida por Phaal (2003a), que afirma que este se configura uma técnica flexível amplamente empregada nas indústrias para apoiar planejamentos estratégicos e de longo alcance.

A abordagem deste processo fornece um meio estruturado e gráfico para explorar e comunicar os relacionamentos entre os mercados em evolução e desenvolvimento, os produtos e as tecnologias ao longo do tempo. Pressupõe-se que a técnica de *roadmapping* possa "(...) ajudar as organizações a sobreviver em ambientes turbulentos por prover um foco na varredura do ambiente e formas de rastrear o desempenho individual das tecnologias, incluindo as potencialmente disruptivas" (PHAAL; FARRUKH; PROBERT, 2004). Os autores ressaltam ainda que os mapas tecnológicos (TRM) são enganosamente simples em termos de formato, mas seu desenvolvimento apresenta desafios significativos. Em particular, o escopo é geralmente amplo, abrangendo uma série de complexas interações conceituais e humanas.

Os *roadmaps* têm se mostrado uma ferramenta bastante poderosa para auxiliar organizações de modo geral a preverem seu futuro com relação ao desenvolvimento de tecnologia e sua relação com o mercado. Para Bray e Garcia (1997), tanto no nível corporativo quanto industrial, o processo de *roadmapping* apresenta vários usos potenciais e consecutivos benefícios resultantes deste processo, sendo os três principais destes abordados a seguir.

Primeiramente, o *roadmapping* pode ajudar a desenvolver um consenso sobre um conjunto de necessidades e as tecnologias necessárias para satisfazer essas necessidades. Em segundo lugar, fornece um mecanismo para ajudar os especialistas a prever a evolução tecnológica em áreas específicas. E por último, pode fornecer uma estrutura para ajudar a planejar e coordenar os desenvolvimentos de tecnologia, tanto dentro de uma pequena empresa, quanto em um grande parque industrial.

A estrutura genérica de um *roadmap* é composta de três camadas. Apesar dos *roadmaps* assumirem diferentes formas, todos procuram basicamente responder a três "simples" questões (simples de propor, mas não de se responder): a) "Para onde estamos indo?"; b) "Onde nos encontramos agora?"; e c) "Como podemos chegar lá?". Coelho, Botelho e Tahim (2012) corroboram a ideia de Phaal et al (2004) no que se refere a explicação da relação entre as camadas de um *roadmap* genérico.

A camada superior diz respeito aos propósitos que a organização aspira, juntamente com fatores que influenciam esses propósitos. Normalmente, dentro do cenário empresarial, esta camada compete tanto perspectivas internas quanto externas (neste caso tanto mercado, como negócios). Se trata basicamente do "*Know-why*", isto é, saber os propósitos que direcionam a criação e execução do processo de *roadmapping*, expondo nesta camada, portanto, o porquê fazer o mesmo. Os principais temas compreendidos nesta camada são usualmente: mercado, consumidores, competidores, ambiente, indústria, negócio, tendências, motivação, ameaças, objetivos, marcos e estratégia.

Na sequência da explicação, tem-se a camada do meio do *roadmap*, a qual compreende os mecanismos dos quais os propósitos citados na camada superior serão alcançados. Tipicamente, os assuntos incluem: produtos, serviços, aplicações, capacidades, desempenho, características, componentes, famílias de produtos, processos, sistemas, plataformas, oportunidades, requisitos e riscos, ou seja, temas tangíveis, que compreendem a pergunta "*Know-what*" (saber o quê), ligados diretamente com a geração de receita para a organização.

As camadas do meio focalizam no desenvolvimento do produto, escolhendo o caminho pelo qual a tecnologia é empregada para atender o mercado e as necessidades do cliente. Estas camadas são cruciais, pois funcionam como uma ponte entre o propósito e os recursos, determinando o que fazer e quais decisões tomar.

Por último, a camada inferior trata dos recursos, os quais devem ser arranjados e integrados a fim de se atender às demandas estipuladas nas camadas superiores. São os mecanismos de entrega que tratam de como fazer "*Know how*". Nesta, os principais assuntos abordados são referentes a habilidades, parcerias, fornecedores, instalações, infraestrutura, organizações, normas, ciência, financiamentos e projetos de P&D.

Dando continuidade à abordagem dos tipos de *roadmaps*, tem-se uma abordagem classificatória mais recente proposta por Kappel (2001), na qual o autor compreende que, com a significativa popularização do termo, quaisquer documentos orientados a predições futuras de tecnologia são denominados *roadmaps*, torna necessária uma melhor definição dos conceitos. Propõem assim, portanto, uma taxonomia, a qual poderá ser no decorrer desse estudo irá tratar da abordagem os aspectos das definições atuais que cercam o tema.

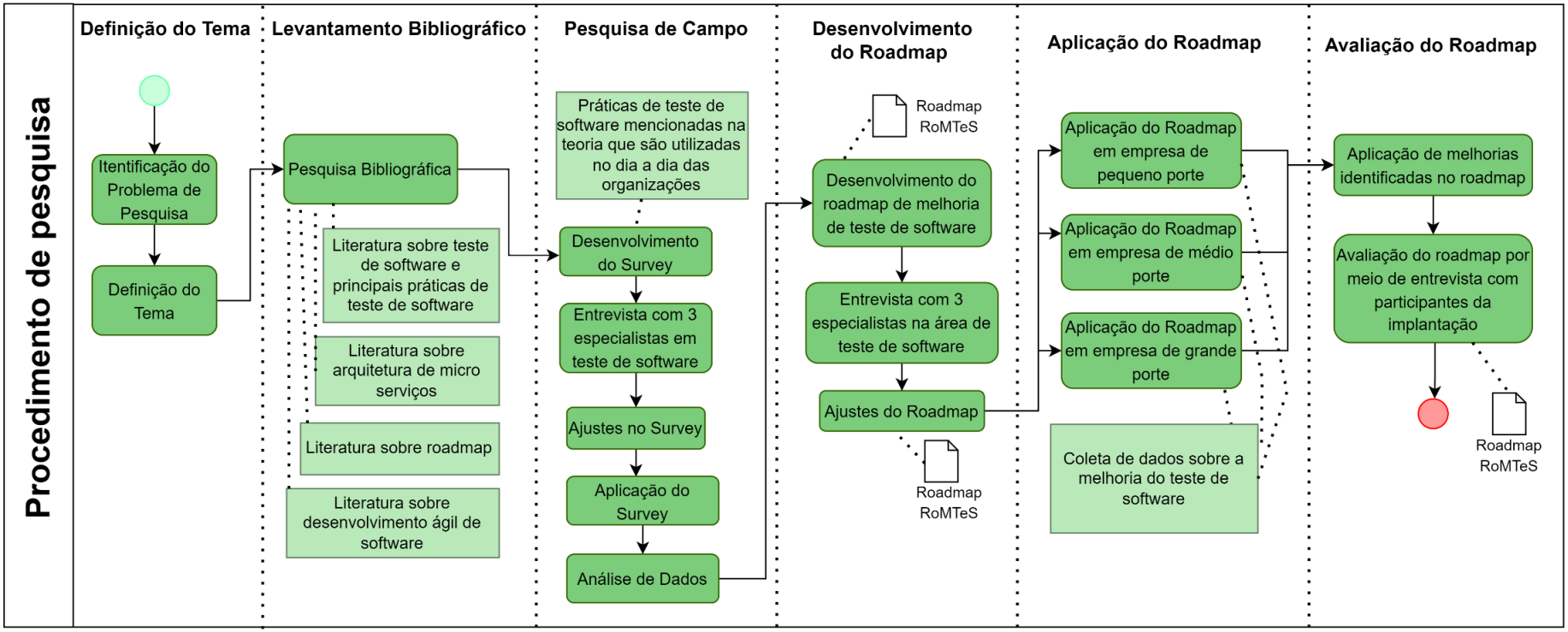
A visão de Kappel (2001) quanto à classificação dos termos é feita a partir de uma diferenciação, primeiramente orientada ao propósito do processo de mapeamento (eixo horizontal), tanto em nível macro (nível industrial), quanto à nível micro (empresarial) do *roadmap* e quanto à aplicação do *roadmap* (eixo vertical), e há uma diferenciação abordando o posicionamento das empresas dentro da indústria e suas respectivas tendências específicas.

Assim, torna-se interessante expor os diferentes tipos de *roadmaps* tecnológicos existentes e suas principais características através da classificação proposta por esses autores. Por meio do estudo realizado, os autores identificaram mais de 40 tipos de TRM em uso nas indústrias e organizações, os quais foram agrupados em 16 áreas e subdivididos e categorizados em dois grupos quanto a seus tipos e formatos.

2. METODOLOGIA

Para o presente estudo, foram seguidas algumas etapas de desenvolvimento: definição do tema, levantamento bibliográfico, pesquisa de campo, desenvolvimento de *roadmap*, aplicação do *roadmap* e avaliação do *roadmap*, as quais podem ser observadas na Figura 3.

Figura 3: Metodologia de pesquisa



Fonte: O Autor

Por meio da pesquisa bibliográfica buscou-se contextualizar os termos desenvolvimento de software, seu uso, micro serviços e testes de software. Segundo Gil (2002, p. 44), "[...] a pesquisa bibliográfica é desenvolvida com base em material já elaborado, constituído principalmente de livros e artigos científicos".

Na pesquisa de campo foi feito o levantamento e análise de grupos respondentes, cálculo do tamanho da amostra dos respondentes. Para Marconi e Lakatos (2003), a pesquisa de campo quantitativo-descritiva consiste em investigações empíricas, que objetivam o delineamento ou análise das características principais ou decisivas de um fenômeno, a avaliação de programas ou ainda o isolamento de variáveis principais ou chave.

Assim, para o estudo aqui apresentado foram (serão) trabalhadas três empresas, de pequeno, médio e grande porte, que terão seus nomes preservados neste estudo. A empresa de pequeno porte atua com a análise de áudios, utilizando aplicações com inteligência artificial e, para este estudo, tratada como empresa Alfa. A empresa de médio porte é uma empresa outsourcing de impressão, fundada a 20 anos, para este estudo, tratada como empresa Beta. Por fim, a empresa de grande porte, é uma empresa do setor bancário, com atuação no mercado a mais de 20 anos e, para este estudo, tratada como empresa Gama. Todas as empresas aqui estudadas têm em comum o fato de que elas desenvolvem seus softwares utilizados em toda a organização.

Considerando que esta pesquisa busca a aplicação de questionário, criação e implantação de *roadmap* e, análise de resultados de maneira qualitativa, por meio de entrevistas com liderança e equipe de desenvolvimento sobre a aplicação do *roadmap* e dos resultados obtidos podemos caracterizá-la como uma pesquisa qualitativa.

2.1. Definição do Tema

Os testes de software estão presentes em todo o ambiente de desenvolvimento de software, tendo atividades realizadas por desenvolvedores e atividades realizadas por testadores, todas as atividades relacionadas ao teste de software são muito importantes para entregas consistentes de software, independente de quem as executa. Segundo Bartié (2002), os testes "têm por objetivo identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo".

Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado. O seu

objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento.

2.2. Questão de Pesquisa

Quais aspectos a considerar na promoção de melhorias em realização de teste de software em processos ágeis de desenvolvimento de micro serviços?

2.3. Objetivo geral

Como já vistos na introdução, o objetivo geral deste trabalho é desenvolver um guia orientador para melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços por meio de um *roadmap*.

2.4. Objetivos específicos

- Descrever as melhores práticas de teste de software encontradas na literatura;
- Verificar se as melhores práticas de teste de software encontradas na literatura são efetivamente usadas no cotidiano das organizações, dando oportunidade para os respondentes indicarem outras melhores práticas de teste de software em desenvolvimento ágil de micro serviços que presenciem nas organizações que atuam;
- Construir o *roadmap* para melhoria de teste de software em processos ágeis de desenvolvimento de micro serviços, contendo as melhores práticas encontradas no primeiro objetivo específico;
- Aplicar o *roadmap* em empresa de pequeno, médio e grande porte, que trabalhem com o desenvolvimento software utilizando arquitetura de micro serviços.
- Avaliar o *roadmap* por meio da sua utilização com entrevistas aos gestores e pesquisa com profissionais que atuam no ambiente de desenvolvimento e testes de software.
- Comunicar a pesquisa por meio da dissertação e publicação de artigos.

2.5. Justificativa de Pesquisa

A utilização crescente de sistemas de computadores em diversos setores da atividade humana tem gerado uma demanda alta por testes de software. Contudo, ao analisar os diversos métodos de teste de software, deve-se atentar ao paradoxo do pesticida, ou seja, se os mesmos testes forem repetidos por diversas vezes, chegará um momento em que eles não encontrarão mais defeitos nas aplicações ISTQB(2011).

Por isso, é importante explorar novas técnicas, ferramentas, critérios e métodos que possam melhorar o processo de teste de software, garantindo a qualidade e a segurança dos sistemas computacionais. A criação de um *roadmap* de melhoria de teste de software se faz necessária para que se possa mapear o nível de maturidade em testes de software, identificar as práticas de testes de software utilizadas e qual sua frequência de uso e, priorizar as práticas de teste de software a serem implementadas, a fim de aumentar o nível de maturidade em teste de software e, consequentemente, a qualidade das aplicações.

Por meio da implantação de um guia orientador de melhoria de teste de software por meio de um *roadmap*, é possível que as organizações possam obter sistemas mais confiáveis e seguros, melhorando a eficiência e a eficácia das práticas de teste de software.

2.6. Levantamento Bibliográfico

Buscando fundamentar os resultados que serão discutidos nos próximos tópicos, o referencial teórico aqui apresentado teve como objetivo auxiliar na concretização das hipóteses propostas e atuarem como argumentos literários. O conteúdo foi extraído a partir das bases de pesquisas Scopus e Web of Science por meio das seguintes palavras-chaves: engenharia de software, métodos ágeis, teste de software, *roadmap,* bem como seus correspondentes em língua inglesa "*software engineering*", "*agile methods*", "*software testing*" e "*roadmap*". Como filtros, foram limitados conteúdos ao idioma português e inglês, com publicação nos últimos 5 anos que compreendem entre o período de 2017 a 2021, limitando-se a artigos científicos e livros.

O levantamento bibliográfico feito teve sua base no teste de software e identificação das principais práticas de teste de softwares recomendadas pela literatura. Bem como levantamento bibliográfico sobre o processo de desenvolvimento de software utilizando métodos e metodologias ágeis. Buscou-se pela literatura sobre *roadmap,* visto que o produto desta dissertação será um *roadmap* de melhoria de teste de software. Pesquisou-se também sobre

arquitetura de micro serviços, visto que é a arquitetura de desenvolvimento utilizada nas organizações que o *roadmap* RoMTeS será implementado.

2.7. Pesquisa de Campo

A pesquisa de campo foi feita por meio da elaboração de um questionário contendo as práticas de teste de software encontradas na literatura e, aplicando aos profissionais que atuam no ambiente de desenvolvimento de software, a fim de se analisar se as práticas de teste de software mencionadas na literatura são, realmente, as práticas de teste de software aplicadas nas organizações que utilizam métodos ágeis de desenvolvimento de software.

Antes da divulgação do questionário, realizou-se a entrevista com 3 profissionais experientes na área para validação de conhecimento, análise do questionário e sugestões, antes de sua divulgação aos respondentes.

Estas entrevistas foram feitas para validação se as questões e práticas de teste de software mencionadas no questionário fazem sentido para pessoas que atuam no ambiente ágil de desenvolvimento de software.

Este questionário foi feito tendo como base as práticas de teste de software encontradas na literatura e, aplicado a profissionais que atuam no ambiente de desenvolvimento ágil de software.

2.8. Desenvolvimento do *Roadmap*

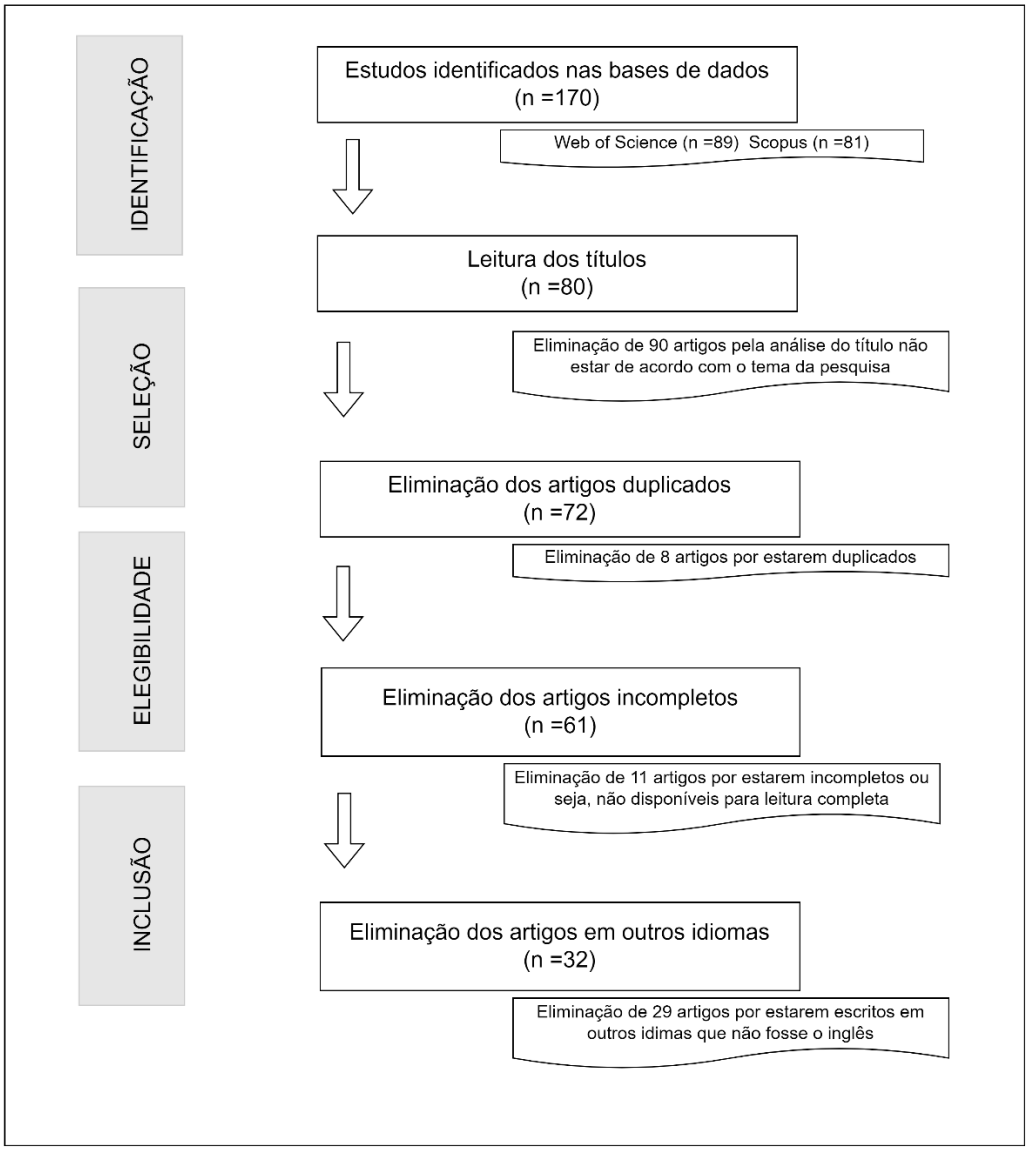
O RoMTeS, desenvolvido com o objetivo de auxiliar as organizações a melhorarem seu nível de maturidade em teste de software e, como consequência, melhorarem seus processos relacionados a testes de software e, aumentarem a qualidade de suas aplicações.

Este *roadmap* é composto pelas etapas de esclarecimentos, definições, averiguações, melhorias e aferições. Ele busca responder às questões de para onde estamos indo, no sentido de melhoria de teste de software, onde nos encontramos agora, como uma visão da organização antes da aplicação do *roadmap* e, como última questão, quais os passos necessários para se chegar à melhoria dos testes de software tendo como base o modelo de maturidade em testes de software TMMi para análise do nível de maturidade em teste de software das organizações.

3. RESULTADOS E DISCUSSÃO

Para o presente estudo, realizou-se a pesquisa bibliográfica por meio das bases de pesquisa Web *Science e Scopus*, para análise e seleção dos artigos, foi realizado um filtro nos artigos selecionados que será representado pelo diagrama de prisma abaixo. Foram utilizados como palavras-chave de busca, os seguintes termos: engenharia de software, métodos ágeis, teste de software, *roadmap*, bem como seus correspondentes em língua inglesa "*software engineering*", "*agile methods*", "*software testing*" e "*roadmap*". Como podem ser observadas no Quadro 2.

Quadro 2: Fluxo de material selecionado. Modelo Prisma adaptado



Fonte: O Autor

Como é possível visualizar no Gráfico 2, a etapa de identificação dos artigos contou com pesquisas realizadas nas bases Scopus e Web of Science, a qual utilizou das palavras-chave "*software engineering*", "*agile methods*", "*software testing*" e "*roadmap".* Foram encontrados um total de 170 artigos, dos quais 89 foram encontrados na base Web of Science e 81 encontrados na base Scopus.

A etapa de seleção dos artigos incluiu a leitura do título de cada um dos 170 artigos, com a seleção dos artigos que poderiam ser utilizados no trabalho, esta seleção removeu um total de 90 artigos, o que resultou em um total de 80 artigos selecionados pelo título.

A etapa de elegibilidade contou com a exclusão de artigos duplicados, ou seja, os artigos presentes tanto na base Scopus quanto na base Web of Science removendo um total de 8 artigos por estarem duplicados.

Esta etapa também contou com a eliminação de artigos que não estavam de acordo ao tema após a leitura do resumo de cada um destes artigos. Esta etapa resultou na eliminação de 11 artigos.

Como última etapa, a etapa de inclusão, foram eliminados artigos em outros idiomas que não fossem o inglês e, com o texto inacessível, ou seja, que não estivessem abertos para leitura e análise do material completo. Esta etapa eliminou materiais que não fossem artigos científicos, o que reduziu uma quantidade de 29 artigos.

Desta maneira, o total de artigos analisados foram 32 artigos científicos publicados nas bases Scopus ou Web of Science e escritos no idioma inglês.

3.1. Discussão dos artigos

Ao analisar a literatura, foi possível observar que grande parte dos estudos que compreendem métodos ágeis e qualidade de *software* são aplicadas em empresas de iniciativa privada. Com a inquirição de publicações foi possível identificar algumas frentes de pesquisa que estão ligadas à pesquisa e algumas que são dependentes dela, norteando e sendo utilizadas de apoio para o desenvolvimento deste estudo.

Dos artigos selecionados e filtrados pelo prisma P sobre as práticas de teste de software, apresentam um foco, como observado em Carvalho (2013), de obter a qualidade por meio da melhoria do processo que é desenvolvido o *software*, para isso, há a implementação de práticas documentadas em alguns processos dos níveis de maturidade, gerando ferramentas sistêmicas.

Ao analisar os artigos do ponto de vista de teste de software, buscou-se identificar as principais práticas de teste de software e, como os autores definem cada uma destas práticas.

Em relação a prática de desenvolvimento orientado a testes, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes e Nagappan *et al.* (2008) em seu artigo sobre a melhoria da qualidade por meio do desenvolvimento orientado a testes definem o termo como uma técnica de design de software na qual os testes automatizados são escritos antes do código de produção, permitindo que o desenvolvedor obtenha feedback rápido sobre a qualidade do seu código e, por ser uma abordagem guiada pelos requisitos do usuário e incentivar a melhoria contínua do código, com o objetivo de fornecer feedback rápido sobre a qualidade do código e reduzir a quantidade de defeitos introduzidos no software.

Sobre a prática testes unitários, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes e Nagappan *et al.* (2008) em seu artigo sobre a melhoria da qualidade por meio do desenvolvimento orientado a testes definem o termo testes unitários com testes automatizados cuja funcionalidade é testar o código de forma isolada, a fim de verificar o comportamento de uma unidade de código isoladamente

Sobre a prática testes unitários baseados em *Mocks*, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes, Nagappan *et al.* (2008) em seu artigo sobre a melhoria da qualidade por meio do desenvolvimento orientado a testes e Freeman & Pryce (2014) em seu artigo sobre a arte do teste unitário com exemplos em C# definem o termo testes unitários baseados em *mocks* como sendo o *mock* um objeto simulado utilizado em testes unitários para simular o comportamento de objetos reais e permitir que os testes sejam executados de forma isolada. Para os autores, esta é uma técnica que permite que testes de unidade sejam executados em isolamento, sem depender de outros componentes do sistema, ou seja, sem dependências externas.

Sobre a prática testes de mutação, os autores Offutt (2002), em seu artigo sobre a métodos de cobertura de teste de software, Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes e Delgado & Vargas (2016), em seu estudo do mapeamento sistemático do teste de mutação definem o teste de mutação como uma técnica que envolve a criação controlada alterações no código-fonte(mutantes) de um programa

e a execução de testes unitários em cada mutante para avaliar a capacidade dos testes unitários de detectar as mutações e falharem e, de acordo os autores, esta prática pode ser utilizada também para avaliar a qualidade dos testes de unidade e identificar deficiências nos casos de teste existentes.

Sobre a prática de análise estática de código, os autores Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes, Dewayne et al. (2010), em seu artigo sobre análise estática de software orientado a objetos e Bashar, et al. (2020), em seu estudo empírico da eficácia de ferramentas de análise estática para identificar vulnerabilidades de segurança definem o termo análise estática de código como um processo que examina o código-fonte de um programa sem executá-lo, para identificar erros de programação, violações de padrões de codificação, problemas de desempenho e vulnerabilidades de segurança.

Sobre a prática testes regressivos, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste, Myers et al. (2003), em seu estudo sobre a arte do teste de software, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes e Nagappan *et al.* (2008) em seu artigo sobre a melhoria da qualidade por meio do desenvolvimento orientado a testes definem o termo testes regressivos como a reexecução com certa frequência de testes para garantir que as mudanças feitas no software não causaram falhas nem regressão em funcionalidades já testadas anteriormente, apesar de se perceber uma variância entre os autores quanto a abordagem de estratégias utilizadas na priorização de testes de regressão como priorização baseada em riscos potenciais, a priorização por testes que cobrem os cenários principais da aplicação e priorização baseada em histórico dos testes falhados.

Sobre ambiente de execução de testes, os autores Wohlin & Thelin (2002) em seu artigo sobre o papel do ambiente de teste no teste de software, Janzen & Saiedian (2008), em seu artigo sobre conceitos, taxonomia e direção do desenvolvimento orientado a testes e Samanta & Dasgupta (2012) em seu artigo sobre a importância do ambiente de teste no teste de software definem o termo ambiente de execução de testes como um conjunto de hardware, software e outros recursos necessários para executar o processo de teste de software. O ambiente de teste, de acordo os autores, deve ser projetado para atender aos requisitos específicos do sistema e deve fornecer um ambiente controlado e replicável para testar o software.

Sobre a prática de testes funcionais automatizados, os autores Srinivasan & Sripriya (2012) em seu artigo sobre um estudo comparativo de ferramentas de testes automatizados, Shweta et al. (2016) em seu artigo sobre teste funcional automatizado em aplicações móveis

definem o termo teste funcional automatizado como ferramentas que automatizam a execução de testes de software para verificar se um aplicativo está funcionando de acordo com as especificações funcionais. Essas ferramentas simulam a interação do usuário com o aplicativo e verificam se o comportamento do aplicativo está correto. De maneira geral, os autores definem o termo como o processo de automatizar a execução de testes e a verificação se estes testes funcionam de acordo com as especificações funcionais.

Sobre a esteira de integração contínua e entrega contínua, os autores Duvall (2007) em seu artigo sobre integração contínua e o aumento da qualidade de software e diminuição de riscos e Humble & Farley (2011), em seu artigo sobre entrega contínua e a confiabilidade do software por meio das etapas de build, testes e entrega contínua definem o termo integração contínua como uma prática que envolve integrar regularmente o código de todos os desenvolvedores em um repositório compartilhado, seguido por compilação automática e execução de testes automatizados para detectar problemas o mais cedo possível no ciclo de vida do desenvolvimento. Em relação a entrega contínua, os autores também foram unanimes ao definirem o termo como conjunto de práticas que visam produzir software de alta qualidade de forma rápida e confiável. Isso é alcançado através da automação de todo o processo de build, testes e *deploy* do software, permitindo que a entrega do software seja feita de forma contínua e com menos erros.

Sobre a prática de testes de integração, os autores Myers et al. (2003) em seu livro sobre a arte do teste de software, Duvall (2007) em seu artigo sobre integração contínua e o aumento da qualidade de software e diminuição de riscos e Kang et al. (2009) em seu artigo sobre considerações formais em teste de software definem o termo testes de integração como o processo de testar a integração entre diferentes componentes de um sistema para verificar se eles funcionam corretamente juntos.

Sobre a prática de testes de desempenho (*performance*), os autores Myers et al. (2003) em seu artigo sobre a arte do teste de software, Kang et al. (2009) em seu artigo sobre considerações formais em teste de software e Garg & Kumar (2019) em seu estudo empírico sobre testes de desempenho, analisando oportunidades e ameaças definem o termo testes de desempenho como uma atividade que visa verificar se o software atende às expectativas de desempenho estabelecidas pelos usuários e especificadas nos requisitos do sistema.

Sobre a prática de testes de estresse (*stress*), os autores Hetzel (1993) em seu estudo sobre a arte do teste de software e Kang et al. (2009) em seu artigo sobre considerações formais em teste de software definem testes de estresse como um teste de software que envolve a avaliação do comportamento do software sob condições extremas de carga ou pressão a fim de

descobrir os limites do software e, para identificar os pontos fracos em sua estrutura. Esta prática de testes, de acordo os autores, envolve a injeção de falhas no sistema para avaliar a sua capacidade de recuperação.

Sobre a prática de testes não funcionais, os autores Hetzel (1993) em seu estudo sobre a arte do teste de software e Kang et al. (2009) em seu artigo sobre considerações formais em teste de software definem testes não funcionais como um conjunto de testes onde são avaliados atributos não funcionais do software, como desempenho, segurança, usabilidade e confiabilidade.

Sobre a prática de testes exploratórios, os autores Hetzel (1993) em seu estudo sobre a arte do teste de software, Wohlin & Thelin (2002) em seu estudo sobre o papel do ambiente de teste de software e Graham & Veenendaal (2008) em seu estudo sobre gerenciamento de testes direcionados aos negócios definem testes exploratórios como um método de testes de software que enfatiza a liberdade e responsabilidade do testador em explorar o software de forma manual, a fim de descobrir falhas e bugs, e relatar os resultados.

Sobre a documentação de testes, os autores Hetzel (1993) em seu estudo sobre a arte do teste de software, Wohlin & Thelin (2002) em seu estudo sobre o papel do ambiente de teste de software, Myers & Badgett (2003), eu estudo sobre a arte do teste de software, Beck & Gamma (2003) em seu artigo sobre desenvolvimento orientado a testes e Graham & Veenendaal (2008) em seu estudo sobre gerenciamento de testes direcionados aos negócios definem que a documentação de testes inclui planos de teste, relatórios de teste, scripts de teste e resultados de teste, assim como deve ser clara, concisa e fornecer informações úteis aos usuários e, atualizada à medida que os testes são modificados ou atualizados.

Em relação a prática de revisão por pares, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste e Kang et al. (2009) em seu artigo sobre considerações formais em teste de software definem a revisão por pares como uma prática em que o código é avaliado por outros membros da equipe de desenvolvimento, em vez de ser revisado somente pelo autor, o que pode ocorre de forma forma informal ou formal e, tem como objetivo encontrar erros, melhorar a qualidade do código, compartilhar conhecimentos e promover a colaboração e comunicação entre os membros da equipe de desenvolvimento.

Sobre a gestão de ciclo de vida da aplicação, os autores Duvall (2007) em seu artigo sobre integração contínua e o aumento da qualidade de software e diminuição de riscos e Humble & Farley (2011), em seu artigo sobre entrega contínua e a confiabilidade do software por meio das etapas de build, testes e entrega contínua definem a gestão de ciclo de vida da aplicação como o processo de gerenciar todas as fases do desenvolvimento de software desde

a concepção até a implementação e manutenção do sistema, o que inclui análise de requisitos, projeto de software, desenvolvimento de código-fonte, testes, implantação, monitoramento e manutenção do sistema.

Em relação a pirâmide de testes, os autores Beck & Gamma (2003), em seu artigo sobre desenvolvimento orientado a teste e Kang et al. (2009) em seu artigo sobre considerações formais em teste de software afirmam que a pirâmide de testes é geralmente composta por três níveis: Testes Unitários na base, Testes de Integração no meio e Testes de Sistema no topo. Os autores concordam ao afirmar que é uma ferramenta utilizada para descrever a estratégia de testes, apresentando uma hierarquia dos testes.

3.2. Práticas de teste de software

Ao analisar-se a literatura sobre testes de software, foram encontrados alguns tipos de teste de software muito mencionados. Com base na apostila Syllabus v3.1, ou seja, a apostila de estudo do ISTQB para o nível de *Certified Tester Foundation Level* (CTFL), que fornece conhecimentos essenciais de teste, explica terminologias e conceitos utilizados na área de teste de software. A CTFL desempenha grande importância para todas as práticas e abordagens de entrega de software que incluem Waterfall, Agile, DevOps, e Continuous Delivery.

Com base nos materiais pesquisados e, nas definições dos autores, realizou-se a análise das práticas de teste software presentes em seu conteúdo, elaborando-se assim, a Tabela 1, contendo as práticas de teste de software e os autores estudados que abordam as práticas de teste de software.

Elaborou-se também a Tabela 2, com base nos materiais pesquisados e, nas definições dos autores realizou-se a análise das definições que cada autor pesquisado deu a cada prática de teste de software.

Tabela 1: Práticas em Teste de Software - Autores

Prática	Autores	Prática	Autores
Pirâmide de teste	BECK, K., & GAMMA, E. (2003) Kang, K. C. Cohen, S. G. Hess, J. A. (2009)	**Testes de integração**	MYERS, G. J. SANDLER, C. BADGETT, T. (2003) DUVALL, P. (2007) Kang, K. C. Cohen, S. G. Hess, J. A., Novak, W.E. Peterson, A.S. (2009)
Desenvolvimento orientado a testes	BECK, K., & GAMMA, E. (2003) JANZEN, D. S., & SAIEDIAN, H. (2008) NAGAPPAN, N. MAXIMILIEN, E. M., BHAT, T. WILLIAMS, L. (2008)	**Testes não funcionais**	HETZEL, B. (1993) Kang, K. C. Cohen, S. G. Hess, J. A. Novak, W.E. Peterson, A.S. (2009)
Análise estática de código	JANZEN, D. S., & SAIEDIAN, H. (2008) DEWAYNE, E. PERRY, D. E, LEBLANC. D, BINKLEY. (2010) BASHAR, M. R. DASH, S. K. ROY, P. S. (2020)	**Testes de desempenho**	MYERS, G. J. SANDLER, C. BADGETT, T. (2003) Kang, K. C. Cohen, S. G. Hess, J. A. Novak, W.E. Peterson, A.S. (2009) GARG, S. KUMAR, R. (2019)
Testes unitários	BECK, K., & GAMMA, E. (2003) JANZEN, D. S., & SAIEDIAN, H. (2008) NAGAPPAN, N. MAXIMILIEN, E. M., BHAT, T. WILLIAMS, L. (2008)	**Testes de estresse**	HETZEL, B. (1993) Kang, K. C. Cohen, S. G. Hess, J. A., Novak, W.E. Peterson, A.S. (2009)
Testes de unidade baseado em mocks	BECK, K., & GAMMA, E. (2003) JANZEN, D. S., & SAIEDIAN, H. (2008) NAGAPPAN, N. MAXIMILIEN, E. M., BHAT, T. WILLIAMS, L. (2008) FREEMAN, S. PRYCE, N. (2014)	**Testes exploratórios**	HETZEL, B. (1993) WOHLIN, C. THELIN, T. (2002) GRAHAM, D. J. Veenendaal, E. V. (2008)
Testes de mutação	OFFUTT, J. (2002) BECK, K., & GAMMA, E. (2003) JANZEN, D. S., & SAIEDIAN, H. (2008) DELGADO, R. N. & VARGAS, A. G. (2016)	**Documentação de teste**	HETZEL, B. (1993) WOHLIN, C. THELIN, T. (2002) MYERS, G. J. SANDLER, C. BADGETT, T. (2003)
Testes regressivos	BECK, K., & GAMMA, E. (2003) JANZEN, D. S., & SAIEDIAN, H. (2008) NAGAPPAN, N. MAXIMILIEN, E. M., BHAT, T. WILLIAMS, L. (2008)	**Revisão por pares**	BECK, K., & GAMMA, E. (2003) Kang, K. C. Cohen, S. G. Hess, J. A. (2009)
Ambiente de execução dos testes	WOHLIN, C. THELIN, T. (2002) JANZEN, D. S., & SAIEDIAN, H. (2008) SAMANTA, D. D. DASGUPTA, S. (2012)	**Esteira de CI/CD**	DUVALL, P. (2007) HUMBLE, J. FARLEY, D. (2011)
Testes funcionais automatizados	SRINIVASAN, S. SRIPRIYA, S. (2012) SHWETA, R. PANDEY, S. K. DUBEY, S. R. (2016)	**Gestão de ciclo de vida de aplicação**	DUVALL, P. (2007) HUMBLE, J. FARLEY, D. (2011)

Fonte: O Autor

Tabela 2: Práticas em Teste de Software - Definições

Prática	Definição com base nos autores	Prática	Definição com base nos autores
Pirâmide de teste	A pirâmide de testes tem três níveis: Testes Unitários, Testes de Integração e Testes de Sistema. Testes de menor nível são executados com mais frequência e testam unidades individuais do código-fonte, enquanto os testes de nível mais alto avaliam o sistema inteiro. Essa estrutura é importante para uma estratégia de testes eficiente e eficaz.	Testes de integração	Teste de integração é o processo de testar a integração entre diferentes componentes de um sistema para verificar se funcionam corretamente juntos. Essa prática é importante no ciclo de vida do desenvolvimento de software, pois permite identificar e corrigir problemas de integração antes que o software seja entregue aos usuários finais.
Desenvolvimento orientado a testes	O desenvolvimento orientado a testes (TDD) é uma técnica em que os testes automatizados são escritos antes do código de produção, permitindo ao desenvolvedor obter feedback rápido sobre a qualidade do código. Essa abordagem é guiada pelos requisitos do usuário, incentiva a melhoria contínua do código e reduz a quantidade de defeitos no software.	Testes não funcionais	Os testes não funcionais avaliam atributos críticos do sistema, como desempenho, segurança, usabilidade e confiabilidade, que são essenciais para garantir a qualidade do software além da funcionalidade. Eles são importantes porque muitas vezes esses atributos são negligenciados durante o processo de desenvolvimento, e a avaliação desses atributos é crucial para a qualidade do software.
Análise estática de código	A análise estática de código examina o código-fonte de um programa para identificar erros, violações de padrões, problemas de desempenho e vulnerabilidades de segurança. Essa técnica é amplamente utilizada no desenvolvimento de software para garantir a qualidade do código e evitar problemas no produto final. A análise estática é uma técnica eficiente e econômica, identificando problemas antes da execução do código, ajudando a avaliar a conformidade com padrões e melhores práticas, e melhorando a legibilidade e manutenibilidade do código.	Testes de desempenho	O teste de desempenho, também conhecido como performance testing, verifica se o software atende às expectativas de desempenho estabelecidas pelos usuários e especificadas nos requisitos do sistema. A avaliação é feita medindo o tempo de resposta, uso de recursos e outros aspectos da eficiência do sistema em situações típicas e excepcionais.
Testes unitários	Testes unitários são testes automatizados que verificam o comportamento de uma unidade de código de forma isolada, como uma classe ou método. Eles garantem que cada unidade de código esteja funcionando conforme o esperado e ajudam a detectar e corrigir problemas mais cedo no processo de desenvolvimento, evitando que se propaguem para outras partes do código.	Testes de estresse	O teste de estresse, também conhecido como stress testing, avalia o comportamento do software sob condições extremas de carga ou pressão, identificando limites e pontos fracos em sua estrutura para melhorar seu desempenho. Essa técnica envolve a injeção de falhas no sistema para avaliar sua capacidade de recuperação.
Testes de unidade baseado em mocks	O teste de unidade baseado em mocks é a prática de simular objetos nos testes de unidade, utilizando mocks que imitam o comportamento de objetos reais. Essa técnica permite que os testes sejam executados de forma isolada, sem depender de outros componentes do sistema.	Testes exploratórios	O teste exploratório é um método de teste de software que enfatiza a liberdade e responsabilidade do testador em explorar o software manualmente para descobrir falhas e bugs. Esse teste é conduzido sem planos de teste ou roteiros prévios, permitindo que o testador use sua criatividade e experiência para encontrar problemas dinamicamente durante a execução do teste.
Testes de mutação	A técnica de mutação envolve a criação de alterações controladas no código-fonte de uma aplicação e a execução de testes unitários em cada mutante, a fim de avaliar a capacidade dos testes de detectar as mutações e falharem. Essa técnica pode ser usada para avaliar a qualidade dos testes de unidade e identificar deficiências nos casos de teste existentes.	Documentação de teste	Os autores concordam que a documentação de testes deve ser clara e concisa, fornecendo informações úteis aos usuários e sendo atualizada sempre que os testes forem modificados ou atualizados. Isso inclui planos de teste, relatórios de teste, scripts de teste e resultados do teste para garantir a rastreabilidade e transparência dos testes de software.
Testes regressivos	A prática de regressão de testes é a reexecução de testes para garantir que as alterações feitas no software não tenham causado falhas ou regressão em funcionalidades já testadas anteriormente. Existem diferentes estratégias para priorizar os testes de regressão, como a priorização baseada em riscos potenciais, a priorização dos testes que cobrem cenários principais da aplicação e a priorização baseada no histórico dos testes falhados.	Revisão por pares	Revisão de código é o método de avaliação do código realizado por membros da equipe de desenvolvimento, além do autor original. Esse processo pode ser formal ou informal e tem como objetivo identificar erros, melhorar a qualidade do código, compartilhar conhecimentos e promover a colaboração e comunicação entre os membros da equipe.
Ambiente de execução dos testes	O ambiente de execução de testes é composto por hardware, software e outros recursos necessários para executar o processo de teste de software. É importante que o ambiente seja projetado para atender aos requisitos do sistema e oferecer um ambiente controlado e replicável para testar o software. O ambiente de teste deve refletir as condições do ambiente de produção para ser eficiente e eficaz na detecção de defeitos no software.	Esteira de CI/CD	Integração contínua é uma prática de integrar regularmente o código de todos os desenvolvedores em um repositório compartilhado, seguido pela compilação automática e execução de testes automatizados para detectar problemas precocemente. Entrega contínua é um conjunto de práticas que automatiza todo o processo de construção, testes e implantação, visando produzir software de alta qualidade de forma rápida e confiável, permitindo uma entrega contínua com menos erros.
Testes funcionais automatizados	O teste funcional automatizado é o processo de automatizar a execução dos testes funcionais e verificar se os requisitos funcionais estão operando de acordo com as especificações. Essa abordagem utiliza ferramentas para simular a interação do usuário com o aplicativo e garantir que o comportamento do aplicativo esteja de acordo com suas especificações funcionais.	Gestão de ciclo de vida de aplicação	O processo de desenvolvimento de software é gerenciado desde a concepção até a implementação e manutenção do sistema. Isso inclui atividades como análise de requisitos, projeto de software, desenvolvimento de código-fonte, testes, implantação, monitoramento e manutenção do sistema.

Fonte: O Autor

Com base nos materiais pesquisados e, nas definições dos autores, realizou-se a análise das práticas de teste software presentes em seu conteúdo, elaborando-se assim, a Tabela 2, contendo as práticas de teste de software e as definições dadas pelos autores observados na Tabela 1.

3.3. Pesquisa de campo

Esta sessão do trabalho é destinada a descrição da pesquisa de campo realizada para identificar se as práticas de teste de software presentes na literatura são realmente as práticas presentes no ambiente de desenvolvimento de software e, para descobrir se existem práticas de teste de software aplicadas nas organizações que não foram encontradas no levantamento bibliográfico, a fim de, por meio destas práticas de teste de software, elaborar um *roadmap* de melhoria de teste de software.

3.3.1. Definição da amostra

O universo dessa pesquisa compreendeu profissionais da área de tecnologia da informação que trabalham no meio ao ambiente de desenvolvimento ágil de software. Por meio de um formulário via Google Forms, foi aplicado um questionário de opinião semiestruturado. O questionário foi enviado para 534 profissionais, dos quais 182 responderam até a presente data. A publicação do questionário se deu por meio de publicação do link aos contatos do LinkedIn, bem como disparado em grupos específicos da área no LinkedIn. Cada envio contou com um pedido de compartilhamento do formulário a outros respondentes.

A escolha deste tipo de coleta de dados foi motivada pelo alcance ampliado que abarca profissionais de diferentes lugares, idade e vivência, uma vez que é feito remotamente. O critério de inclusão nesta pesquisa compreendeu vínculo empregatício com alguma empresa na qual o respondente atue no ambiente de desenvolvimento de software ágil, além do termo de consentimento livre e esclarecido para participação no estudo.

3.3.2. Questionário

Este questionário tem como objetivo analisar se as práticas de teste de software presentes na literatura são, realmente, as práticas de teste de software utilizadas no cotidiano das organizações.

Com base nas práticas de testes dispostas nas Tabelas 1 e 2, elaboraram-se questões relacionadas a elas, de forma que seja possível o questionamento sobre quais práticas de teste de software são utilizadas nas organizações, representadas no Quadro 3.

Quadro 3: Práticas de teste de software

Desenvolvimento orientado a testes
Testes unitários
Testes unitários utilizam *mocks*
São aplicados testes de mutação
Testes unitários são necessários para conclusão da história
Análise estática de código
Testes regressivos
Há um ambiente específico para execução dos testes
O status de retorno das aplicações são validados
Os campos no banco de dados são validados
Testes funcionais automatizados
Testes automatizados estão integrados na esteira CI/CD
Erro nos testes acarreta falha na esteira CI/CD
Ocorre desativação de testes(*bypass*) na esteira CI/CD
Testes funcionais são necessários para conclusão da história
Testes de integração com banco de dados ou outras aplicações
Testes de desempenho (*performance)*
Testes de estresse (*stress)*
Testes não funcionais são necessários para conclusão da história
Testes exploratórios
São escritos cenários/planos de teste
Há padrões na escrita dos cenários/planos de teste
Revisão por pares dos cenários/plano de teste
São evidenciadas as execuções de teste
Ocorre a documentação dos defeitos encontrados
Resultados dos testes são utilizados para melhoria do produto
São estimadas horas para planejamento/execução dos testes
Os testes são baseados em requisitos de negócios
Gestão de ciclo de vida de aplicação (Exemplo: SilkCentral, AzureDevOps, Jenkins)
Execução dos testes são realizados com frequência
São planejados quantidade de testes por nível da pirâmide de teste

Fonte: O Autor

As práticas de teste de software, apresentadas no Quadro 3, foram agrupadas em perguntas, de forma a se analisar o conhecimento do respondente em relação as práticas de teste de software, bem como analisar a usabilidade destas práticas de teste de software no ambiente ágil de desenvolvimento de software.

O questionário foi elaborado tendo algumas abordagens para as questões, as quais foram divididas em sessões, aonde a primeira sessão é sobre a abordagem de caracterização dos respondentes, a segunda abordagem sobre planejamento de atividades na iteração, a terceira abordagem sobre conhecimento dos respondentes nas práticas em teste de software, a quarta abordagem sobre aplicabilidade das práticas de teste de software, a quinta abordagem é sobre o papel do testador ou da equipe de testes e, a última abordagem, sobre as métricas utilizadas nas organizações.

3.3.2.1. Abordagem de caracterização dos respondentes

Para caracterização dos respondentes, primeiramente, verificou-se o porte da empresa, utilizando a classificação de acordo com o SEBRAE (2013): microempresa (até 9 funcionários), pequena empresa (de 10 a 49 funcionários), média empresa (de 50 a 99 funcionários), grande empresa (acima de 99 funcionários).

Para analisar a extensão da organização, perguntou-se sobre a categorização nacional ou internacional e estados nacionais de atuação da empresa. Perguntou-se também, em relação à categorização do software, se é a principal atividade da organização (atividade fim) ou atividade necessária para ofertar produtos/serviços (atividade meio).

Para um melhor entendimento do tipo de software desenvolvido pela organização, perguntou-se pelo tipo de software desenvolvido, se específico para dispositivos móveis, para desktops, aplicações web, sistema operacional, software embarcado, de engenharia ou científico, software ERP, software de prateleira, de inteligência artificial, jogos e outros, com a opção para o respondente explicitar qual software sua organização desenvolve, caso não se encaixe em nenhuma das categorias anteriores.

A quantidade de membros na equipe de desenvolvimento também foi questionada e, para análise de pessoas focadas na atuação em testes de software e, se estas pessoas trabalham no mesmo local físico ou de forma distribuída, além de questionar sobre a função do respondente na equipe de desenvolvimento.

Perguntou-se também a função do respondente no time, se desenvolvedor, testador, gerente e, um campo de outros para que o respondente informe sua função, caso necessário.

Para saber qual a maturidade dos respondentes, verificou-se o tempo de experiência em ambiente de desenvolvimento ágil utilizando a escala likert com cinco níveis: 0 a 1 ano, 1 a 3 anos, 3 a 5 anos, 5 a 10 anos, acima de 10 anos.

Com o intuito de verificar qual método ou abordagem de desenvolvimento os participantes utilizam, criou-se uma pergunta com a opção de selecionar os métodos abordados na fundamentação teórica: Scrum, Dynamic Systems Development (DSDM), Feature Driven Development (FDD), Extreme Programming (XP), Crystal, Lean, Kanban e Outros.

3.3.2.2. Abordagem de planejamento de atividades na iteração

Para análise das atividades e práticas de teste realizadas dentro de uma iteração, perguntou-se dentro das práticas mencionadas na literatura, são utilizadas na iteração, como criação e execução de testes de histórias, testes em pares com outros testadores e desenvolvedores, validação do negócio, automação de novos testes de funcionais, execução de testes de regressão automatizados, demonstração para o *stakeholders*, criação dos ambientes de testes e, revisão dos cenários de teste pelo time e uma opção outros, em caso o respondente deseje mencionar uma opção diferente das apresentadas.

Perguntou-se também, sobre o tempo ideal de uma iteração, para que fosse possível tanto codificar, como executar os testes necessários e, relacionado ao mesmo assunto, perguntou-se ao respondente se os testes são executados durante toda a iteração, com as opções de uma até quatro semanas, além de uma opção sobre entrega de software em funcionamento todos os dias e, uma opção outros, em caso o respondente deseje mencionar uma opção diferente das apresentadas.

Além destas questões, perguntou-se também sobre a organização realizar testes durante toda a iteração, com respostas possíveis de sim ou não.

3.3.2.3. Abordagem de conhecimento sobre as práticas de teste de software

Para análise das práticas de teste de software, é necessário antes, analisarmos o nível de conhecimento do respondente nas práticas de teste de software levantadas na literatura, as quais são desenvolvimento orientado a testes, testes unitários, testes de mutação, análise estática de

código, esteira de integração CI/CD, teste funcional automatizado, testes regressivos, ambientes de execução de teste, documentação de testes, documentação de defeitos, validações/resultados esperados, gerenciamento de ciclo de vida da aplicação, execução de testes, *mocks*, testes de integração, testes de desempenho (*performance*), testes de estresse e pirâmide de testes. Estes conhecimentos foram medidos por meio da escala likert com quatro níveis: não conheço, conheço pouco e conheço bem.

3.3.2.4. Abordagem de aplicabilidade das práticas de teste de software

Para análise da aplicabilidade das práticas de teste de software encontradas na literatura, por meio de medição da escala likert com cinto níveis: não sei informar, nunca, raramente, frequentemente e sempre, perguntou-se ao respondente sobre as práticas de teste de software: desenvolvimento orientado a testes, testes unitários, testes unitários utilizam *mocks*, São aplicados testes de mutação, testes unitários são necessários para conclusão da história, análise estática de código, testes regressivos, há um ambiente específico para execução dos testes, o status de retorno das aplicações são validados, os campos no banco de dados são validados, testes funcionais automatizados, testes automatizados estão integrados na esteira CI/CD, erro nos testes acarreta falha na esteira CI/CD, ocorre desativação de validações (*bypass*) na esteira CI/CD, testes funcionais são necessários para conclusão da história, testes de integração com banco de dados ou outras aplicações, testes de desempenho (*performance*), testes de estresse, testes não funcionais são necessários para conclusão da história, testes exploratórios, são escritos cenários/planos de teste.

Há padrões na escrita dos cenários/planos de teste, revisão dos cenários/plano de teste, são evidenciadas as execuções de teste, ocorre a documentação dos defeitos encontrados, resultados dos testes são utilizados para melhoria do produto, são estimadas horas para planejamento/execução dos testes, os testes são baseados em requisitos de negócios e gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkins), execução dos testes são realizados com frequência e são planejados quantidade de testes por nível da pirâmide de testes.

Questionou-se também sobre a estratégia de automação relacionados à pirâmide de testes, com as opções: unitário (automação de testes de unidade), serviço (automação na camada de API-*Application programming interface*) e GUI (automação na camada de GUI-*Graphical*

user interface) e, uma opção outros, em caso o respondente deseje mencionar uma opção diferente das apresentadas.

Perguntou-se também sobre os fatores considerados na hora de automatizar um teste, tendo como opções: não se utiliza automação de testes, utilização para teste de regressão, facilidade de automação, ganhos em relação à automação de processo manual, necessidade de negócio, impacto na aplicação, feedback mais rápido quanto ao resultado dos testes (passou/falhou) e todos os testes são automatizados.

Outra questão feita foi sobre a organização da equipe de desenvolvimento, com as opções de: a equipe é integrada, mas com separação de papéis entre desenvolvedores e testadores, a equipe é integrada, com equipes multidisciplinares e sem separação de papeis, a equipe de teste é independente e, o respondente teve a liberdade para responder outro, caso trabalhe em um formato diferente dos propostos.

3.3.2.5. Abordagem sobre papel do testador ou equipe de testes

Para análise do papel e funções de um testador, questionou-se sobre a contribuição da equipe de teste no processo de desenvolvimento, com as opções de: promove feedback durante todo o processo de desenvolvimento, ajuda com conhecimento técnico de implantação do código do teste a ser realizado, assume a liderança nos testes de aceitação, assume a liderança nos testes de regressão, desenvolve os planos de testes, revela cenários de teste adicionais através de testes exploratórios, garante que a cobertura de teste é adequada, lidera os esforços de automação, lidera os esforços de testes de integração, executa testes de nível de sistema, mantém ambientes de teste e os dados disponíveis, identifica problemas e partes técnicas de teste, faz a priorização das funcionalidades e cenários que devem ser automatizados e, uma opção livre para que o respondente se sinta livre para escrever outra função.

Questionou-se também sobre a existência de práticas de teste de software utilizados pela organização que o respondente não observou no questionário, sendo esta questão, de texto livre.

3.3.2.6. Abordagem sobre métricas

Sobre métricas, o respondente é questionado sobre as métricas utilizadas no contexto de teste de software, com a opção de incluir outras métricas, as quais são: não são utilizadas métricas, cobertura de código (indica a relação entre o tamanho do código de testes e tamanho

do código em produção), quantidade de casos de teste e pontos de validação, porcentagem de pontos de validação de teste de unidade passando e falhando, quantidade de testes de aceitação por funcionalidade, porcentagem de validações de teste de aceitação passando e falhando, funcionalidade testadas e entregues, tempo de execução e quantidade de defeitos encontrados e uma opção livre, caso o respondente deseje informar outra métrica utilizada na organização que trabalha.

Como última pergunta do questionário, questionou-se ao respondente sobre seu interesse em receber os resultados da pesquisa, pedindo para que, em caso positivo, informasse seu e-mail.

3.3.3. Entrevistas

Após a criação do questionário, foram selecionados 3 especialistas na área de testes de software, os quais preencheram o questionário de forma a expressarem livremente sua opinião sobre as questões, abordagem e demais assuntos que achassem importante ser comentado.

Durante a pesquisa bibliográfica, foram encontradas na literatura uma grande variedade de práticas de teste de software, práticas estas, utilizadas para compor o questionário, a fim de se validar se realmente as práticas encontradas na literatura são ou não, conhecidas pelos profissionais que atuam no ambiente de desenvolvimento ágil de software e, se realmente são utilizadas no cotidiano das organizações.

Para realizar esta validação, antes de liberar o questionário aos respondentes, selecionou-se três especialistas em teste de software, pessoas com conhecimento e ampla experiência profissional para responderem ao questionário de forma a avaliar sua estruturação e conteúdo. Estas entrevistas-teste do questionário se deu por meio de quatro etapas com os entrevistados.

A primeira etapa foi o esclarecimento da pesquisa, motivações e justificativa, bem como apresentação do questionário para que o respondente preencha o questionário e realize comentários sobre toda e qualquer questão.

A segunda etapa, foi a leitura e resposta do questionário pelo respondente entrevistado, o qual teve liberdade de comentar sobre todas as questões apresentadas na pesquisa.

A terceira etapa foi anotar todas as considerações do respondente sobre todos os aspectos do questionário, tempo a ser respondido, questões feitas e entre outros.

A quarta etapa foi uma pergunta feita ao respondente, sobre o que ele sentiu falta no questionário, bem como se teria uma questão a ser removida ou acrescentada ao questionário.

Para manter a privacidade dos respondentes, os chamaremos de Respondente 01, Respondente 02 e Respondente 03.

3.3.3.1. Entrevista com Respondente 01

A primeira entrevista foi feita com profissional especialista na área de teste de software, com experiência de mais de oito anos atuando em ambiente de desenvolvimento de software ágil e com experiência em testes manuais, testes funcionais automatizados para plataformas web e desktop, além de experiência em testes não funcionais, como testes de estresse e testes de desempenho, com experiência profissional em empresas de grande porte de tecnologia, varejo e setor financeiro.

As considerações feitas pela Respondente01 foram pautadas ao tempo de resposta do questionário, que levou 25 minutos para ser respondido e, ela me alertou sobre a possibilidade de desistência de alguns respondentes pela demora em responder todas as questões.

Outro fator muito interessante levantado pela Respondente01 foi sobre termos em inglês, ela mencionou que não era necessário colocar alguns termos em inglês pois, quem atua realmente na área conhece os termos em português e, na leitura das questões e melhorando fluidez.

Ao final da entrevista-teste, perguntei à Respondente01 sua opinião de forma geral, se as questões faziam sentido para meu objetivo de validar se as práticas mencionadas na literatura realmente são utilizadas nas organizações e se ela teria alguma questão a acrescentar e sua resposta foi positiva para as duas questões, em relação ao objetivo da validação, ela respondeu que sim, o questionário iria auxiliar a descobrir se as práticas de teste de software mencionadas na literatura são utilizadas nas organizações e, em relação ao acréscimo de uma questão, ela comentou da importância de incluir uma questão aberta, para que o respondente possa incluir também, sua opinião sobre alguma prática que ache importante porém não tenha sido mencionada.

3.3.3.2. Entrevista com Respondente 02

A segunda entrevista foi feita com um profissional especialista na área de teste de software, foi feita após aplicação de correções e melhorias levantadas pela Respondente01.

A experiência deste profissional, Respondente02 é de dez anos atuando em ambiente de desenvolvimento de software ágil e com experiência em testes manuais, testes funcionais automatizados para plataformas web e desktop, além de experiência em testes não funcionais, como testes de estresse e testes de desempenho, experiência em criação e implantação de esteiras de integração CI/CD e virtualização de serviços. Este especialista, Respondente02 tem uma forte experiência profissional em empresas do setor financeiro, já atuou também em empresas de pequeno, médio e grande porte do varejo, empresas de tecnologia e empresas do setor automotivo.

As considerações feitas pelo Respondente02 foram pautadas no tempo de resposta do questionário, que levou 20 minutos para ser respondido, para ele, este tempo ainda era muito longo para um questionário.

Outro fator muito interessante levantado pelo Respondente02 foi a objetividade, ele comentou sobre quanto mais objetivo e direto, melhor na obtenção de respostas e sugeriu também, o mínimo possível de questões abertas, em texto, fator que, segundo o Respondente02, acarretaria problemas na tabulação destes dados.

O respondente02 também comentou a respeito do termo de consentimento livre e esclarecido, ao dizer que é necessário explicitar que, ao avançar com o questionário, o respondente acorda os termos mencionados e comentou da importância de ter um tempo previsto para responder o questionário, logo no início do mesmo.

Ao final da entrevista-teste, perguntei ao Respondente02 sua opinião de forma geral, se as questões faziam sentido para meu objetivo de validar se as práticas mencionadas na literatura realmente são utilizadas nas organizações e se ela teria alguma questão a acrescentar e sua resposta foi positiva em relação ao objetivo da validação, ele respondeu que o questionário iria auxiliar na descoberta se as práticas de teste de software mencionadas na literatura são utilizadas nas organizações e, em relação ao acréscimo de uma questão, ele comentou que não teria nenhuma questão para incluir, exceto, o agrupamento de algumas práticas de teste de software, para facilitar durante o processo de resposta do questionário.

3.3.3.3. Entrevista com Respondente 03

A terceira entrevista foi feita com um profissional especialista na área de teste de software, foi feita após aplicação de correções e melhorias levantadas tanto pela Respondente01 como pelo Respondente02.

A experiência deste profissional, Respondente03 é de sete anos atuando em ambiente de desenvolvimento de software ágil e com experiência em testes manuais, testes funcionais automatizados para plataformas web, além de experiência em análise de requisitos, construção de cenários de testes e validações com área de negócios. Esta especialista, Respondente03 tem experiência profissional em empresas do setor financeiro, já atuou também em empresas de pequeno, médio e grande porte do varejo, empresas de tecnologia e área farmacêutica.

A Respondente03 não fez muitas considerações, opinou na importância das questões, que achou muito válidas para o objetivo da pesquisa, de validar se as práticas de teste de software descritas na literatura realmente são as práticas de teste de software aplicadas no dia a dia das organizações.

Um ponto interessante levantado pela Respondente03 é que na primeira pergunta havia um questionamento sobre a experiência do respondente com desenvolvimento de software, a Respondente03 achou mais apropriado perguntar se o respondente tem experiência com o ambiente de desenvolvimento de software, pois assim, fica mais claro para o respondente que caso ele atue com outro papal dento do ambiente de desenvolvimento de software, além de desenvolver, pode também responder ao questionário.

Ao final da entrevista-teste, perguntei à Respondente03 sua opinião de forma geral, se as questões faziam sentido para meu objetivo de validar se as práticas mencionadas na literatura realmente são utilizadas nas organizações e se ela teria alguma questão a acrescentar e sua resposta foi positiva em relação ao objetivo da validação, ele respondeu que sim, o questionário iria auxiliar na descoberta se as práticas de teste de software mencionadas na literatura são utilizadas nas organizações e, em relação ao acréscimo de questões, ela comentou que não teria nenhuma questão para incluir, que havia gostado da forma como o questionário estava.

3.3.4. Respostas

A pesquisa de campo é um método de pesquisa preocupado com a compreensão e interpretação das interações sociais de grupos de pessoas, comunidades e sociedade, observando e interagindo com as pessoas em seus ambientes naturais. No caso, a análise foi alicerçada no ambiente laboral de programadores, basicamente, onde questões acerca da contenda da pesquisa foram disponibilizadas para resposta.

Os comentários e sugestões dos Respondente 01, Respondente 02 e Respondente 03 foram implementadas no questionário, o que contribuiu grandemente em sua composição.

Esta pesquisa foi disponibilizada a partir do dia 25 de novembro de 2022 por meio de um formulário do Google Forms, sendo divulgada no LinkedIn e, por meio de pedidos para responderem e reenviarem este questionário para cada contato.

A pesquisa, até o dia 25 de janeiro de 2023 teve um total de 182 respondentes.

A primeira questão feita foi se o respondente tem experiência com o ambiente de desenvolvimento de software utilizando métodos ágeis. Como resposta, tem-se o Gráfico 1.

Gráfico 1: Experiência com desenvolvimento de software

Fonte: O Autor

O primeiro aspecto a ser notado é se os programadores possuíam conhecimento ou até mesmo experiência com métodos ágeis para elaboração de softwares. A pesquisa apresentou que que 23,6% dos respondentes têm experiência em desenvolvimento tradicional de software e, 76,4% possuem experiência profissional em ambiente ágil de desenvolvimento de software. Isto representa que, grande maioria dos respondentes atuam com metodologias e práticas ágeis de desenvolvimento de software.

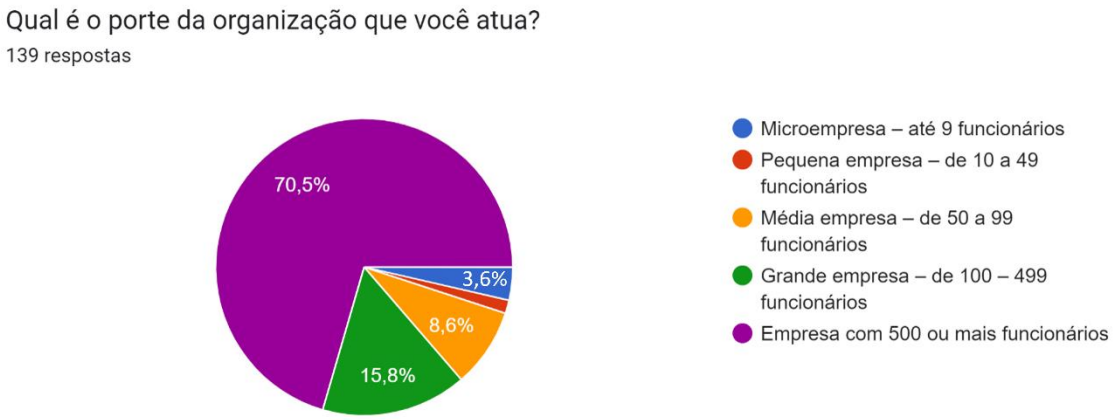
A diferença entre as metodologias ágeis e tradicionais está no foco e nos valores que cada uma delas prioriza. Enquanto as metodologias ágeis têm o foco nas pessoas, as metodologias tradicionais enfocam os processos e algoritmos. Além disso, as metodologias ágeis têm um menor investimento em documentação e priorizam mais a implementação. Essa metodologia espera que mudanças ocorram e está preparada para lidar com elas. À medida que

se aprende mais sobre a problemática a ser resolvida, melhores soluções são encontradas, sendo que a metodologia ágil se baseia em feedbacks e mudanças constantes.

Para análise dos dados, foram descartadas as respostas dos 43 respondentes que não atuam com o ambiente ágil de desenvolvimento de software, visto que as perguntas seguintes do formulário visam analisar características do ambiente ágil de desenvolvimento de software.

A segunda questão feita sobre o porte da organização que o respondente atua e, como resposta, tem-se o Gráfico 2.

Gráfico 2: Porte da organização

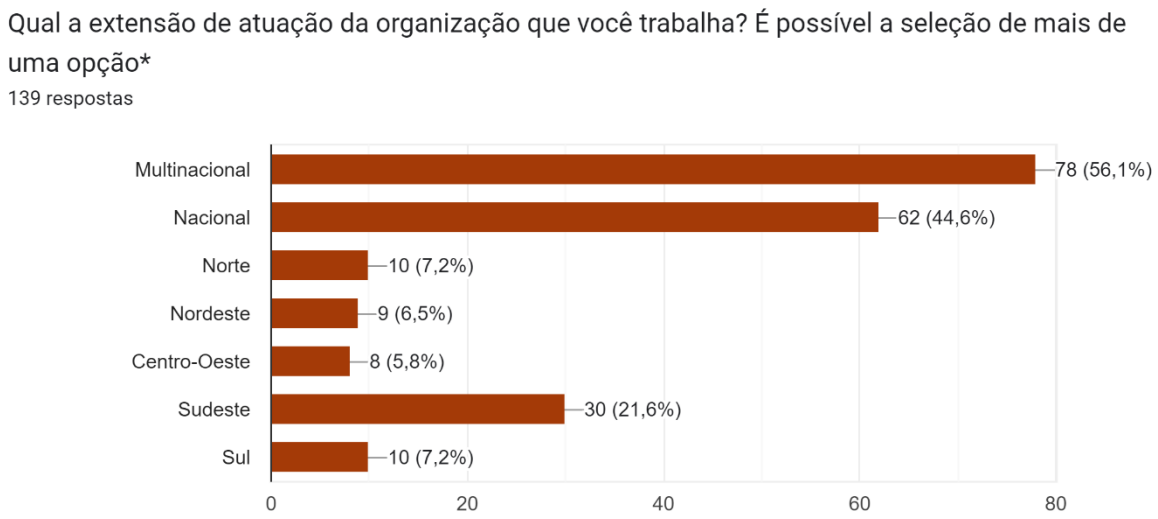


Fonte: O Autor

A resposta a esta questão demostra que mais da maioria dos respondentes da pesquisa, ou seja, 70,5% atuam em organizações com um quadro de funcionários igual ou maior a 500 funcionários. O tamanho do mercado global de software e serviços de negócios foi avaliado em US $ 429,59 bilhões em 2021 e deve se expandir a uma taxa de crescimento anual composta (CAGR) de 11,7% de 2022 a 2030. O volume crescente de dados corporativos e o aumento da automação dos processos de negócios em setores como varejo, manufatura e saúde estão impulsionando o crescimento do mercado. Além disso, a rápida implantação de software e serviços corporativos na infraestrutura de TI para melhorar a tomada de decisões, reduzir custos de estoque e aumentar a lucratividade também está contribuindo para o crescimento do mercado. Considerando tais pressupostos, não é de se espantar que quase 4/5 das empresas pesquisadas possui mais que 500 funcionários.

A terceira questão feita sobre a extensão da atuação da organização, sendo possível a seleção de mais de uma opção pelo respondente. Como resposta, tem-se o Gráfico 3.

Gráfico 3: Extensão da organização



Fonte: O Autor

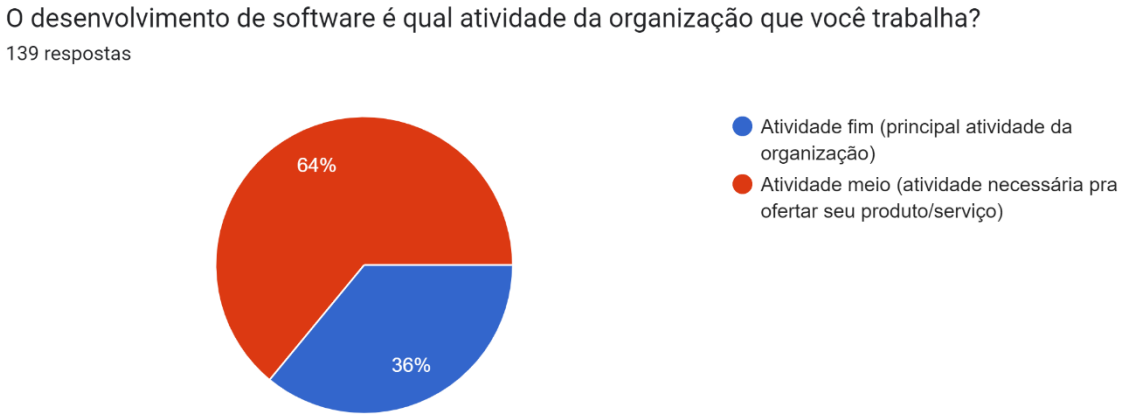
A resposta para esta questão indica que 56,1% dos respondentes, ou seja, mais da metade atua em organizações multinacionais e, 44,6% trabalham em organizações nacionais. Pode-se perceber também os respondentes informaram a opção Sudeste como área de atuação das organizações, o que indica uma forte atuação dos respondentes neste estado.

As multinacionais instalam-se em diferentes países em busca de vantagens locacionais importantes que contribuam para a diminuição dos custos de produção, como o melhor acesso às matérias-primas, a existência de legislações trabalhistas e ambientais mais fracas, a presença de mão de obra barata (SOUZA, 2017). Assim, as multinacionais foram apontadas em grande maioria pois há maiores chances de que essas utilizem de metodologias avançadas, como é o caso dos métodos ágeis.

Além disso, o Sudeste se destacou dentre as localidades e isso se dá porque 67,09% dos empregos gerados por multinacionais encontram-se na região Sudeste do país. Com número elevado de mão de obra e dinheiro em caixa, devido ao lucro da cafeicultura, a Região Sudeste logo se tornou a área mais industrializada e de maior concentração de população do país (SILVA, 2019).

A quarta questão feita sobre o desenvolvimento de software ter qual finalidade na organização que o respondente trabalha e, como resposta, tem-se o Gráfico 4.

Gráfico 4: Atividade de software na empresa



Fonte: O Autor

A resposta para esta questão indica que 64% dos respondentes, ou seja, mais da metade trabalham em organização aonde o desenvolvimento de software é a atividade meio, ou seja, atividade necessária para ofertar o produto/serviço, enquanto 36% dos respondentes selecionaram a opção de desenvolvimento de software da empresa que atuam como atividade fim, ou seja, principal atividade da organização,

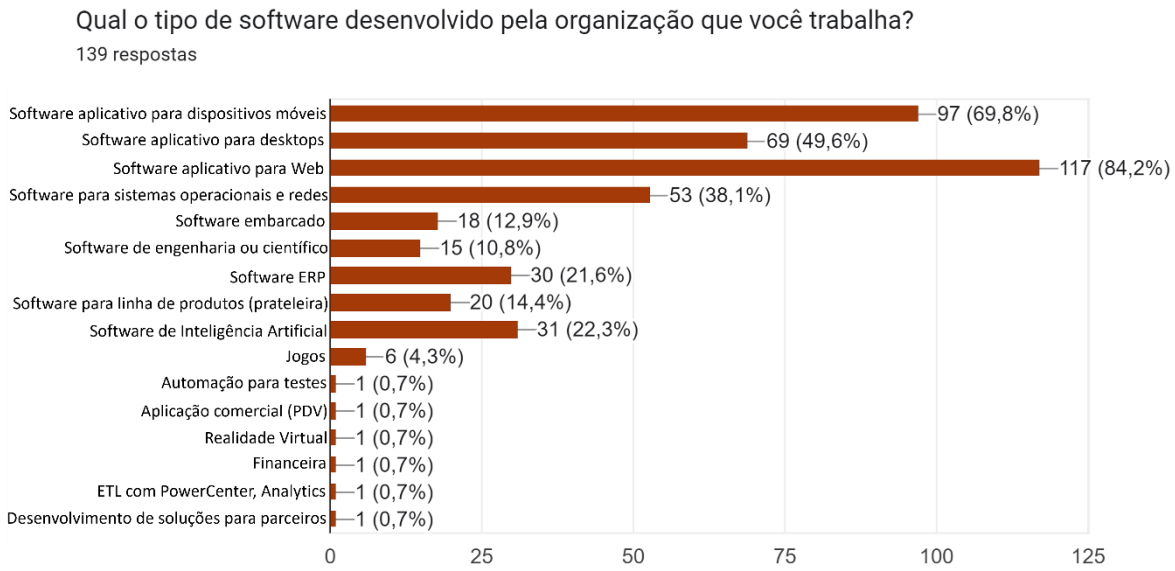
O desenvolvimento de softwares para empresas é muito importante, pois possibilita usar a tecnologia em favorecimento dos processos dela (PRESSMAN, 2011). Isso faz com que seja possível economizar dinheiro e ter os problemas resolvidos rapidamente, garantindo a excelência da gestão.

Muitas empresas estão adotando sistemas devido a várias razões, tais como: decepção com sistemas incompatíveis, incapacidade do Departamento de Tecnologia de Informação em realizar a integração entre os sistemas existentes atualmente na empresa e outros motivos que influenciam diretamente a competitividade da Empresa (SILVA, 2019). A tendência atual da área de sistemas de informações gerenciais é de não apenas visualizar a empresa isoladamente, mas toda a cadeia de suprimento, conseguindo realizar o planejamento estratégico e tático globalmente para a cadeia, além do operacional para a empresa (REZENDE, 2005).

Algumas empresas que realizavam somente vendas diretas, realizaram parcerias com outras empresas (muitas vezes brasileiras) para realizar vendas através de outros canais, procurando aumentar, assim, sua capilaridade (PETTERS, 2001).

A quinta questão feita sobre o tipo de software desenvolvido na organização que o respondente trabalha e, como resposta, tem-se o Gráfico 5.

Gráfico 5: Tipo de software desenvolvido pela organização



Fonte: O Autor

A resposta para esta questão indica os quatro tipos de softwares mais desenvolvidos, pelas organizações que os respondentes atuam, são softwares para dispositivos móveis, software aplicativo para desktops, software aplicativo para web, software para sistemas operacionais e redes, seguidos por software embarcado, software de engenharia ou científico, software ERP, software para linha de produtos (prateleira), software de inteligência artificial, jogos e outros.

Os sistemas de informação deixaram de ser algo opcional para as empresas, e se tornaram alvo das maiores necessidades que uma empresa pode possuir, o papel que esses sistemas possuem em relação à organização é de extrema importância, uma vez que nestes ficam as responsabilidades de manter, e gerar novos dados, que por sua vez podem se caracterizar como um dos maiores patrimônios da empresa (RUSSO, 2021). Dados sigilosos, informações de clientes, informações de produtos e estoque, financeiro, relatórios gerenciais e muitas outras informações ficam a cargo desses sistemas.

A atualidade do mercado está contida num mundo globalizado, e a internet tornou-se parte integrante desse novo sistema de negociação. Dia a dia cresce o número de usuários, serviços, informações e facilidades disponíveis online (REZENDE, 2005). Desenvolvimento

de softwares para web diferencia de um desenvolvimento desktop, exatamente no que diz respeito ao público-alvo, maneira de acesso e manutenção do mesmo.

A maneira de acesso torna-se mais acessível uma vez que é um sistema web e pode ser acessado através de navegadores de internet e com características multiplataforma já que não obriga o usuário final a possuir determinado sistema operacional, e sim apenas ter um navegador de internet.

E em relação à manutenção, fica mais simples a manutenção e atualização do mesmo já que não necessita o usuário final a executar procedimentos de instalação, facilita o suporte não necessitando diretamente visitas ao usuário final nem assistência remota, pois os arquivos estão armazenados em servidor web e sua manipulação afeta instantaneamente o usuário final (PEREIRA, 2007).

A sexta questão feita sobre a utilização da arquitetura de micro serviços no desenvolvimento de software e, como resposta, tem-se o Gráfico 6.

Gráfico 6: Utilização de arquitetura de micro serviço

Fonte: O Autor

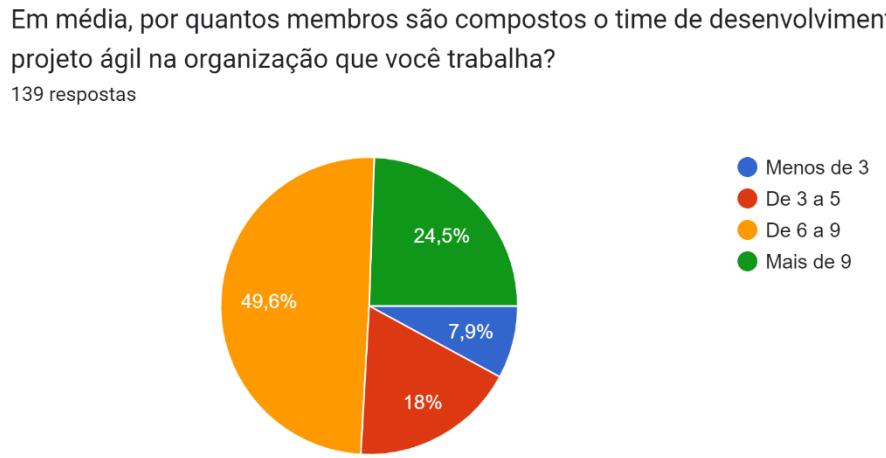
A resposta para esta questão indica que grande maioria, ou seja, 83,5% dos respondentes apontaram que sim, utilizam a arquitetura de micro serviços para desenvolvimento de software nas organizações em que atuam. 10,8% responderam que não sabem informar e, somente 5,8% informaram que não utilizam a arquitetura de micro serviços nas aplicações desenvolvidas em suas organizações.

Nesta arquitetura de micro serviços, o desenvolvimento de aplicativos em que um grande aplicativo é criado a partir de componentes ou serviços modulares. Cada módulo oferece suporte a uma tarefa ou meta de negócios específica e usa uma interface simples e bem definida, como uma interface de programação de aplicativos (API), para se comunicar com outros conjuntos de serviços.

O questionário mostrou que muitos dos respondentes utilizam a arquitetura de micro serviços em suas organizações durante o processo de desenvolvimento de software, o que significa que esta arquitetura é muito utilizada pelas organizações.

A sétima questão feita sobre o tamanho da equipe de desenvolvimento. Como resposta, tem-se o Gráfico 7.

Gráfico 7: Porte do time de desenvolvimento



Fonte: O Autor

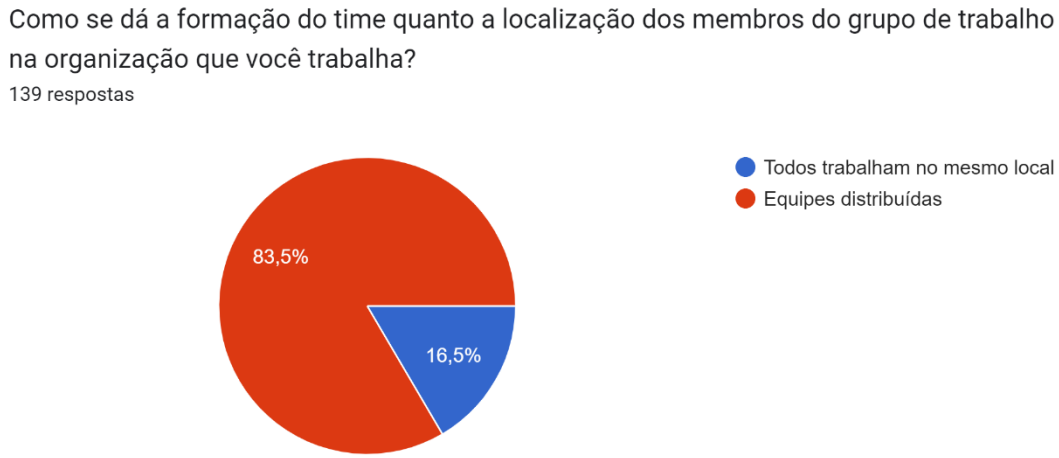
A resposta para esta questão indica que metade dos respondentes, ou seja, 49,6% disseram que suas equipes de desenvolvimento são compostas de 6 a 9 membros. Seguidos de 24,5% que responderam que suas equipes de desenvolvimento são compostas de mais de 9 membros. Seguidos da porcentagem de 18% dos respondentes que disseram que suas equipes de desenvolvimento são compostas de 3 a 5 membros e, uma pequena porcentagem de 7,9% dos respondentes, afirmaram que suas equipes de desenvolvimento são compostas de menos de 3 membros.

Por meio desta questão, percebe-se que grande parte das organizações seguem as boas práticas de desenvolvimento de software, que sugerem que, as equipes de desenvolvimento devem ter um tamanho que, duas pizzas sejam suficientes para alimentar o time. O tamanho

ideal, seria de três a nove pessoas de acordo com a literatura e não existem tarefas definidas para o programador, Tester, designer etc., onde cada um é responsável por uma atividade (MARCONI, 2001).

A oitava questão, feita sobre a formação do time quanto a localização dos membros do grupo de trabalho na organização que o respondente atua e, como resposta, tem-se o Gráfico 8.

Gráfico 8: Estratégia de formação do time



Fonte: O Autor

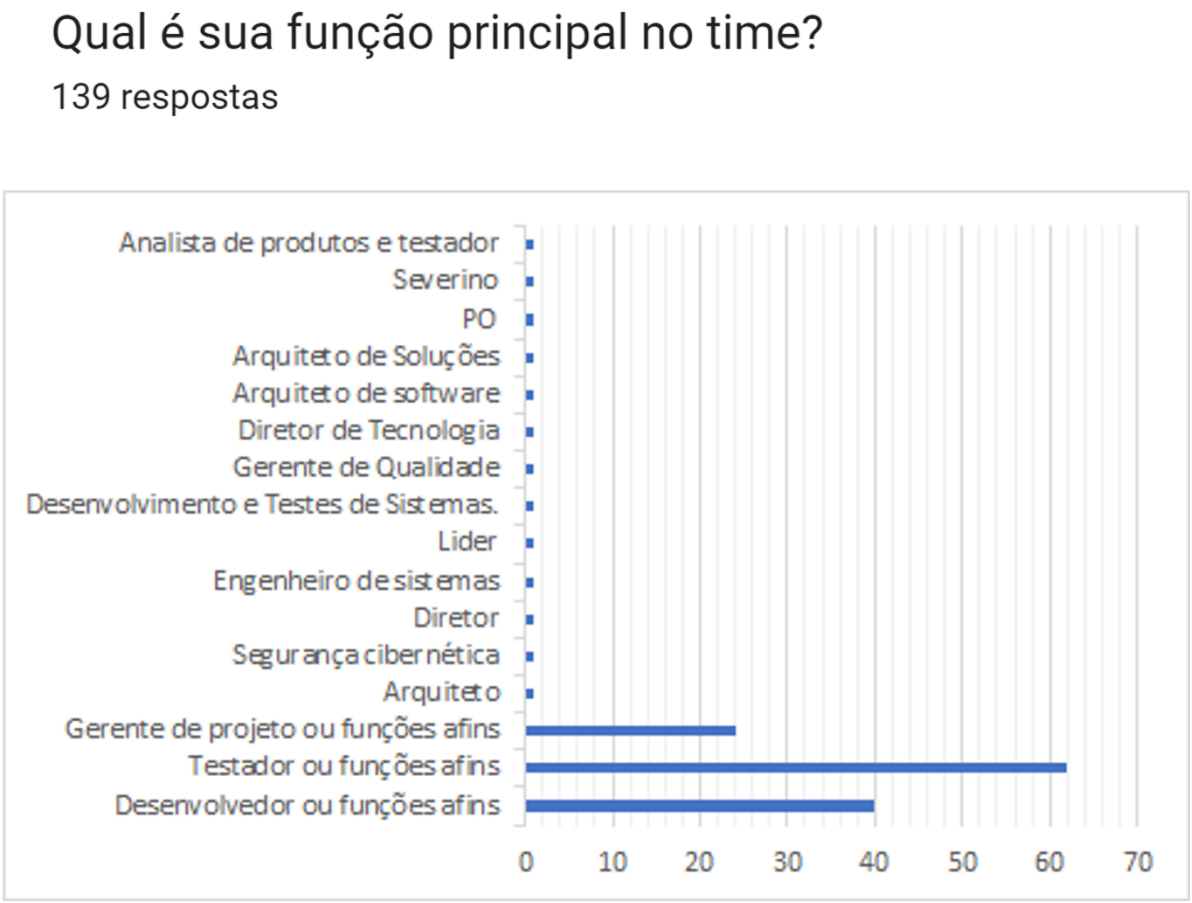
A resposta para esta questão indica que mais da metade dos respondentes, 83,5% dos respondentes atuam em equipes distribuídas, o que indica trabalho remoto com contribuição de pessoas de diversas localizações. Apenas 16,5% dos respondentes informaram trabalhar em equipes aonde todos os integrantes trabalhem em um mesmo local.

Pode-se levar em consideração o advento da Covid-19, a qual aumentou, em grau nunca visto antes, o trabalho remoto para diversas áreas de atuação, sendo que, algumas destas atividades continuaram remotas após a pandemia do Covid-19, enquanto outras voltaram suas atuações para o modelo de trabalho presencial.

Na área de tecnologia da informação, especialmente quando se fala das funções voltadas para as áreas de desenvolvimento de software, tiveram uma maior permanência ao home office mesmo após o período de quarentena devido a Covid-19, o que aumentou o número de funcionários trabalhando para uma mesma equipe em locais descentralizados, fator que justifica o grande número de respostas apontando para equipes distribuídas, ao invés de equipes centralizadas, atuando em um mesmo local físico.

A nona questão, feita sobre a função do respondente na equipe de desenvolvimento de software. Como resposta, tem-se o Gráfico 9.

Gráfico 9: Função no time



Fonte: O Autor

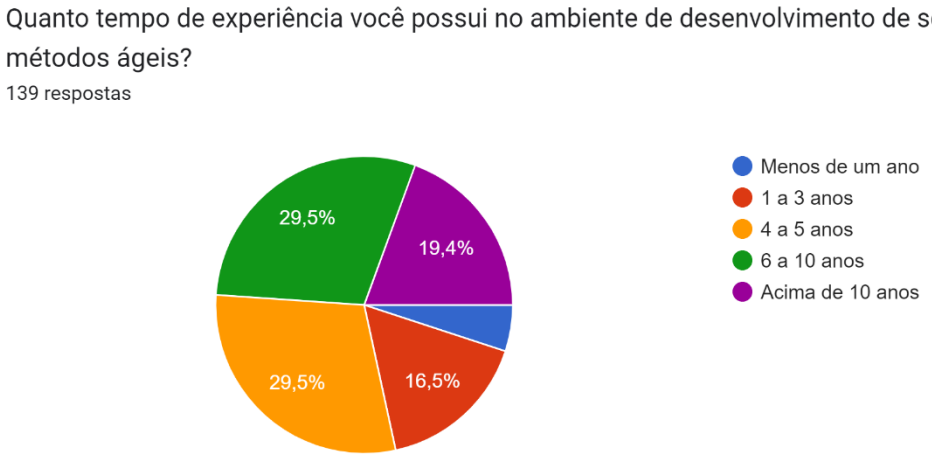
A resposta para esta questão indica que grande parte dos respondentes, 62 respondentes têm função de testador de software ou atividades afins. 40 respondentes têm função de desenvolvedor ou atividades afins e, 24 respondentes têm função gerente de projeto ou função semelhante. Os outros responderam informaram arquiteto de soluções, PO, Diretor e outros.

O grande número de respondentes que atuam com a função de testador de software ou funções similares, indica que, as respostas a seguir, de foco nas práticas de teste de software vão indicar um nível de conhecimento muito grande sobre a área, devido os respondentes atuarem com a função específica responsável pelos testes das aplicações. Isto indica que as pessoas respondentes do questionário fazem parte do dia a dia da construção dos sistemas e,

têm precisão e conhecimento para responderem sobre as práticas de teste de software aplicadas nas organizações que atuam com certa maturidade de conhecimento sobre testes de software.

A décima questão, feita sobre o tempo de experiência profissional do respondente em ambiente de desenvolvimento de software utilizando métodos ágeis. Como resposta, tem-se o Gráfico 10.

Gráfico 10: Tempo de experiência



Fonte: O Autor

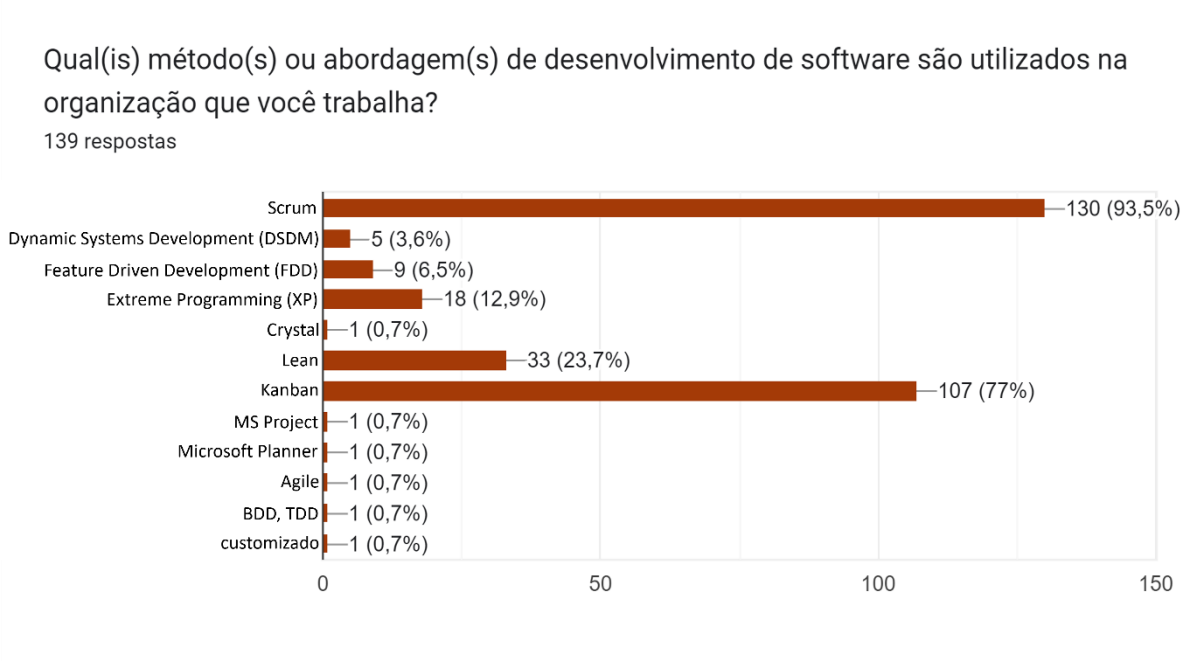
A resposta para esta questão indica a maturidade dos respondentes no quesito ao tempo de experiência na área, na qual podemos ver algumas faixas de experiências que nos traz confiança quanto ao nível de maturidade, visto que, 29,5% dos respondentes têm entre 6 e 10 anos de experiência na área. O segundo grupo de maior tempo de experiência é composto por 29,5% dos respondentes, que estão entre 4 e 5 anos de experiência profissional na área. O terceiro maior grupo, 19,4% possuem acima de 10 anos de experiência na área. O quarto maior grupo de respondentes, o que equivale a 16,5% respondeu que possui entre 1 a 3 anos de experiência e, o menor grupo, composto por 5% dos respondentes possui menos de um ano de experiência profissional na área. Este dado nos informa que os respondentes têm certo nível de maturidade para responder e opinar sobre os assuntos questionados neste formulário.

Esta questão indica o nível de maturidade dos profissionais que responderam ao questionário possuem sobre o assunto de testes de software e, de acordo as respostas, percebe-se que um quarto dos respondentes possuem de seis a 10 anos de experiência profissional da

área de atuação, o que nos remete a profissionais com conhecimento suficiente para não somente a responderem ao questionário, como também maturidade suficiente para incluírem sugestões viáveis para o ambiente profissional. O segundo maior grupo de respostas apontou que, uma experiência profissional de quatro a cinco anos e, o terceiro maior grupo de respostas abrangeu profissionais com mais de dez anos de experiência profissional na área, o que indica um nível de maturidade grande sobre o assunto para responder com propriedade sobre os assuntos relacionados a teste de software.

A questão sobre qual(is) método(s) ou abordagem(s) de desenvolvimento de software são utilizados na organização que o respondente trabalha. Como resposta, tem-se o Gráfico 11.

Gráfico 11: Adoção de métodos de desenvolvimento



Fonte: O Autor

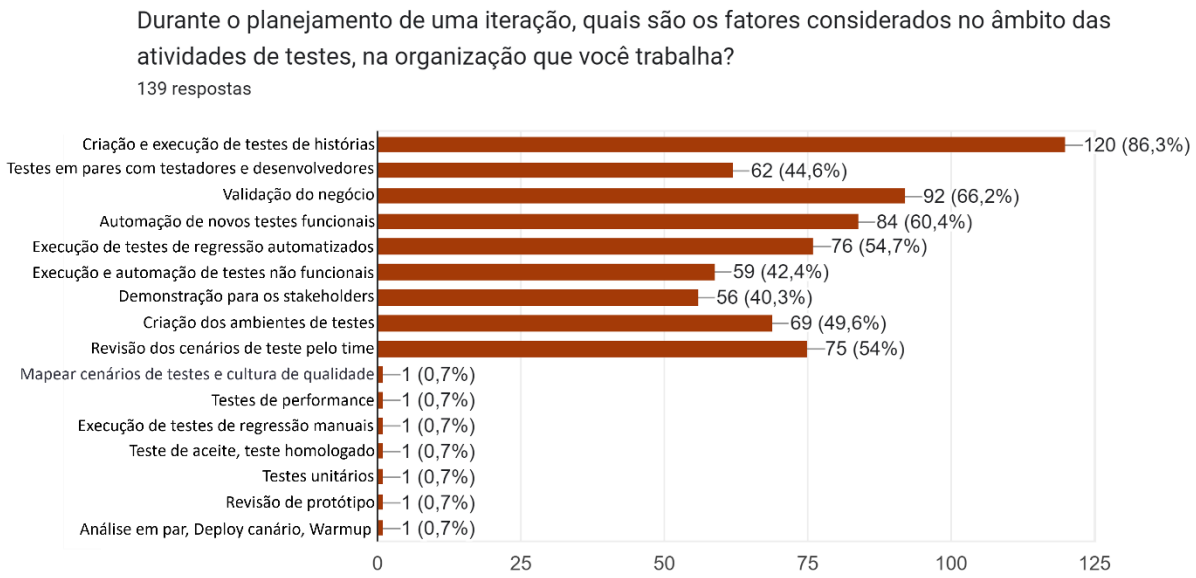
A resposta para esta questão indica quais os métodos ou abordagens de desenvolvimento de software são mais utilizadas nas organizações que os respondentes atuam, sendo os dois maiores grupos de respostas, Scrum e Kanban, seguidos de Lean e XP.

De acordo com levantamento do IGTI, o Scrum é o framework ágil mais famoso entre as empresas brasileiras, presente em 74% delas por sua eficiência na otimização de processos e sua versatilidade: ele se adapta facilmente a diversas áreas e não fica restrito apenas ao desenvolvimento de tecnologias.

O Quadro Kanban possibilita a visualização e a verificação do andamento de todo o ciclo de fabricação dos produtos, e é um método certeiro para identificação de prioridades de produção e tarefas a serem efetuadas (PETTERS, 2001).O que diferencia a XP das outras metodologias ágeis é que a XP enfatiza os aspectos técnicos do desenvolvimento de software. A programação extrema é precisa sobre como os engenheiros trabalham, pois, seguir práticas de engenharia permite que as equipes forneçam código de alta qualidade a um ritmo sustentável (PEREIRA, 2007).

A décima segunda questão, feita sobre os fatores considerados no âmbito das atividades de testes, tiveram como resposta, o Gráfico 12.

Gráfico 12: Período de codificação e execução



Fonte: O Autor

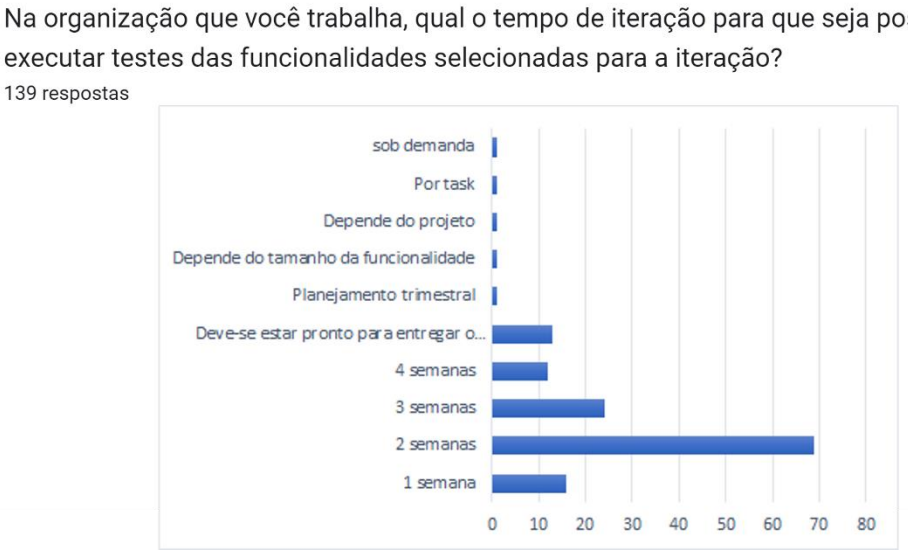
A resposta para esta questão indica quais os fatores levados em consideração no âmbito das atividades de teste de software são, principalmente, criação e execução de testes de histórias, validação do negócio, automação de novos testes funcionais, execução de testes de regressão automatizados, revisão dos cenários de teste pelo time, criação dos ambientes de teste, testes em pares com outros testadores e desenvolvedores, execução e automação de testes não funcionais e demonstração para os *stakeholders*.

Na produção de software, as atividades de testes são consideradas a forma mais importante de se assegurar ou controlar a qualidade do software (HERZLICH, 2007), mas

aplicar estas atividades dentro do contexto ágil, tem se mostrado desafiador para muitas empresas.

A décima terceira questão, feita sobre o tempo de uma iteração na organização em que o respondente trabalha teve como resposta, o Gráfico 13.

Gráfico 13: Tempo para iteração



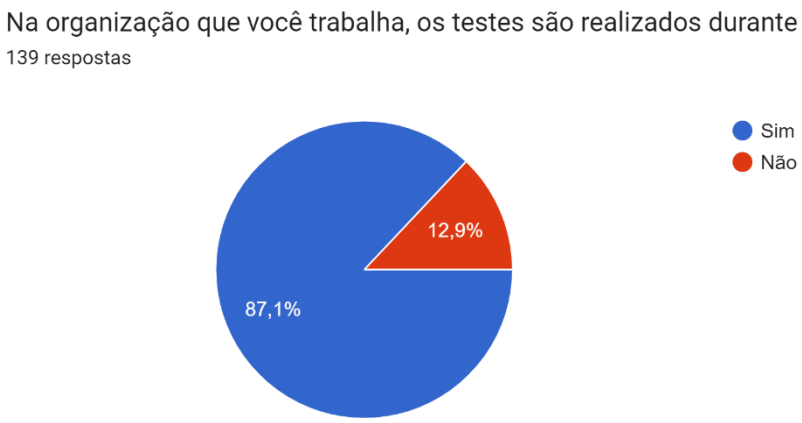
Fonte: O Autor

A resposta para esta questão indica que, 49,6% dos respondentes disseram que em suas organizações, o tempo de uma iteração equivale a 2 semanas de trabalho. 17,3% dos respondentes selecionaram o tempo de iteração equivalente a 3 semanas de trabalho, 11,5% dos respondentes afirmaram terem o equivalente a uma semana como tempo de uma iteração e, 9,4% dos respondestes afirmaram que deve estar pronto para entregar o software funcionando todos os dias.

Esta pergunta foi feita para identificação do período que as organizações consideram equivalentes a uma iteração. Usualmente, estes têm duração de duas ou três semanas, período no qual ocorrem a codificação, os testes e a correção dos defeitos de partes do produto em criação. Isto evita o acúmulo de problemas, pois estes são corrigidos em poucos dias ou horas após serem adicionados ao código.

A décima quarta questão, feita sobre a execução dos testes durante todo o período da iteração teve como resposta, o Gráfico 14.

Gráfico 14: Execução dos testes durante a iteração



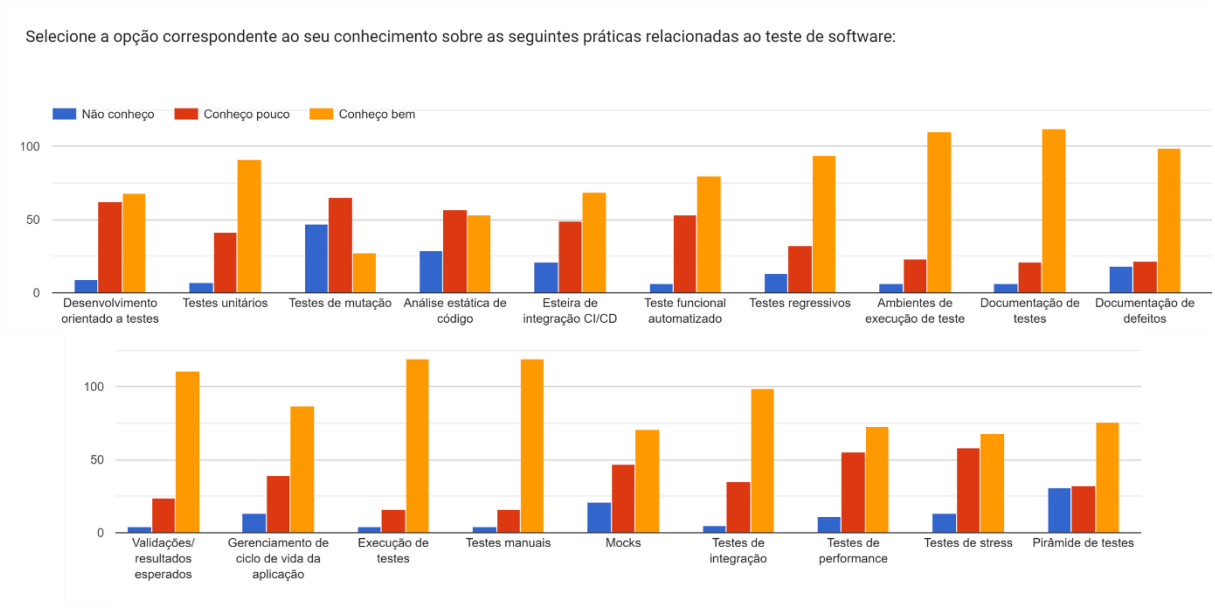
Fonte: O Autor

A resposta para esta questão indica que 87,1% dos respondentes disseram que em suas organizações, os testes de software são executados durante toda a iteração e, somente 12,9% dos respondentes afirmaram que a execução de testes não acontecerem em todo o período da iteração.

Testar durante todo o tempo da iteração evita que ao final do processo surjam erros ou falhas, e assim otimiza o tempo e evita um possível retrabalho com o software. Para que o testar ágil seja, de fato, incorporada por toda a equipe de entrega. Afinal, de acordo com o método ágil, faz muito mais sentido realizar testes que acompanham o processo, do que testar tudo na etapa final, já próximo da entrega do produto.

A décima quinta questão, feita sobre o nível de conhecimento dos respondentes sobre as práticas de teste de software levantadas pela literatura teve como resposta, o Gráfico 15.

Gráfico 15: Conhecimento dos respondentes sobre as práticas de teste de software



Fonte: O Autor

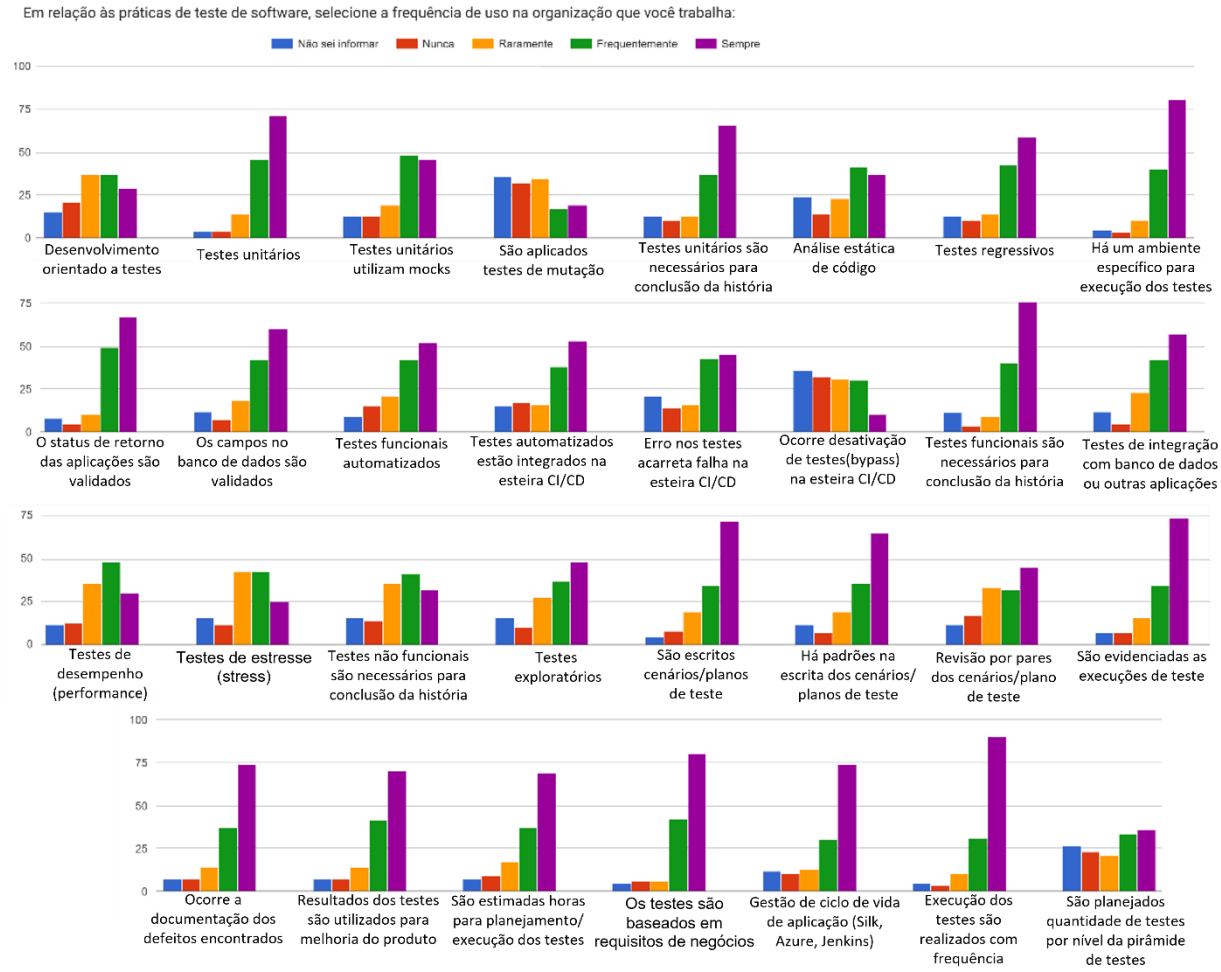
A resposta para esta questão indica quais as práticas de teste de software levantadas na literatura são realmente conhecidas pelos respondentes, dentre as quais tem-se as práticas testes unitários, testes regressivos, ambiente de execução de testes, documentação de testes, documentação de defeitos, validações/resultados esperados, execução de testes e testes manuais como as práticas de teste de software mais conhecidas pelos respondentes do questionário.

Neste gráfico percebe-se que muitos dos respondentes possuem bastante conhecimento em práticas de teste de software, isto pode se dar pelo motivo da divulgação desta pesquisa ter sido feita por meio da rede de LinkedIn pessoal do autor, o qual contém pessoas de mesma área de atuação, pessoas que atuam com a área de testes e desenvolvimento de software, as quais, encaminharam também para os seus contatos, pessoas com experiência na área de testes e desenvolvimento de software.

Com níveis de conhecimento como os relatados pelos respondentes, pode-se construir uma pesquisa sólida com uma visão das organizações que possuem testes de software em um nível estruturado e, com pessoas que conhecem e, algumas são até referência em testes de software, pelo nível de conhecimento relatado no questionário.

A décima sexta questão, feita sobre a aplicabilidade das práticas de teste de software levantadas na literatura, e sua frequência de uso nas organizações que o respondente trabalha. Como resposta, tem-se o Gráfico 16.

Gráfico 16: Práticas de teste de software aplicadas nas organizações



Fonte: O Autor

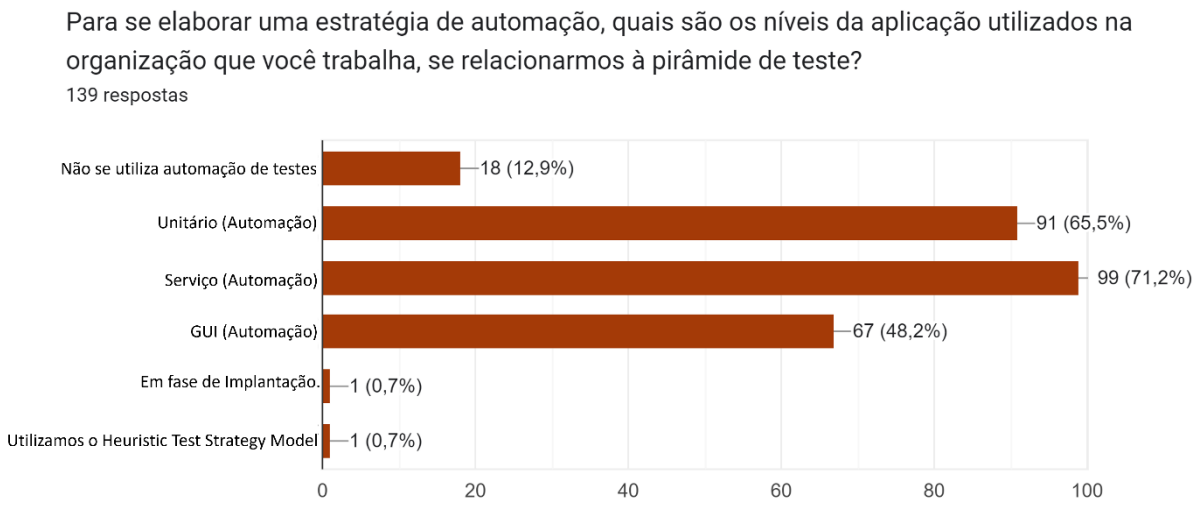
A resposta para esta questão indica quais são as práticas de teste de software mais utilizadas no ambiente de desenvolvimento de software das organizações que os respondentes atuam. Tendo como principais práticas de teste de software aplicadas nas organizações, a existência de um ambiente específico para execução dos testes e a execução dos testes são realizados com frequência.

Ao analisarmos as respostas dadas a esta questão, percebeu-se que, embora as respostas demonstrem certo nível de conhecimento dos entrevistados nas práticas de teste de software e, apesar de as organizações nas quais eles atuam sejam bem estruturadas e tenham certos níveis de maturidade no tocante aos testes de software, ainda tem-se uma divergência entre o conhecido e o aplicado, como o caso da questão específica sobre pirâmide de testes, aonde

grandemente maioria dos respondentes informou que conhece bem e, quando se analisa a aplicação da pirâmide de testes na prática, percebe-se um grande número de testes fim a fim, em relação aos testes de unidade e de integração, o que demonstra uma pequena diferença entre o conhecimento que os respondentes possuem da realidade aplicada nas organizações.

A décima sétima questão, feita sobre os níveis da aplicação relacionados a estratégia de automação. Como resposta, tem-se o Gráfico 17.

Gráfico 17: Níveis da aplicação



Fonte: O Autor

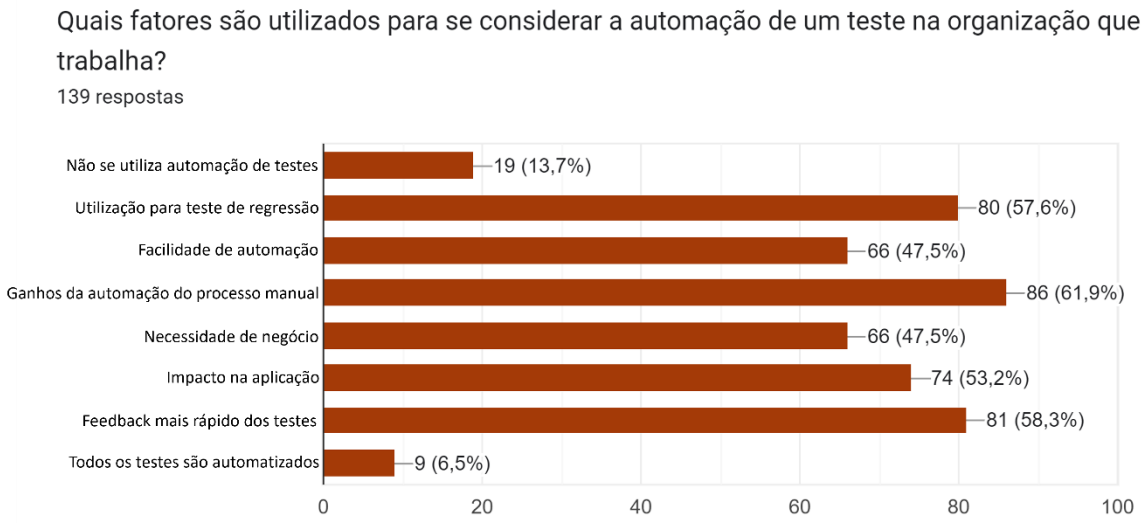
A resposta para esta questão indica que grande maioria dos respondentes, ou seja, 71,2% dos respondentes selecionaram a opção de camada de serviço, seguida da camada de unitário com 65,5% dos respondentes e, 48,2% dos respondentes selecionaram a opção de teste de interface como estratégias para automação dos testes. Com uma pequena parcela de 12,9% dos respondentes que selecionaram a opção de não se utiliza automação de testes.

Esta resposta representa que, de acordo a quantidade de respondentes informando nível alto de conhecimento em pirâmide de testes, sua aplicação ocorre de forma diferente do esperado, pois, os níveis da pirâmide automatizadas são, em certo grau, muito próximos, tendo pouca diferenciação entre testes de API e, um dos motivos que podem ter levado esta resposta foi o fato de que grande maioria dos entrevistados atuam com aplicações cuja arquitetura é em micro serviços. A arquitetura em micro serviços tem uma necessidade maior por testes de integração, os quais devem analisar os pontos de integração entre os componentes, chamados de micro serviços.

Observa-se também um alto número de respostas para a automação dos testes unitários, os quais validam o código da aplicação em um nível mais profundo, testando as menores partes possíveis do código, a fim de garantir que aplicação, em suas menores partes, realiza as funções para as quais foram desenvolvidas.

A décima oitava questão, feita sobre os fatores utilizados para se considerar automação de um teste. Como resposta, tem-se o Gráfico 18.

Gráfico 18: Fatores considerados para automação de um teste



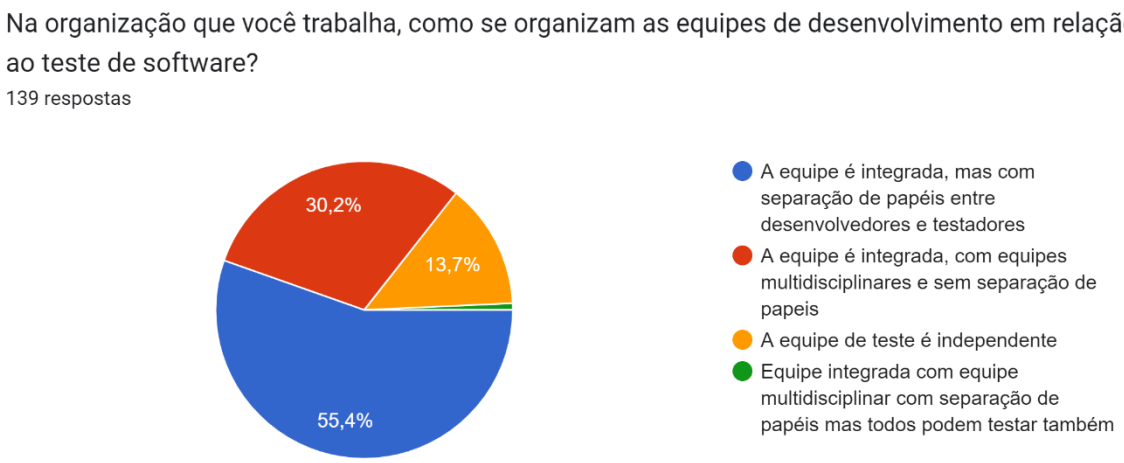
Fonte: O Autor

A resposta para esta questão indica que grande parte dos respondentes optaram pelas opções de ganhos em relação à automação de processo manual, utilização para teste de regressão e feedback mais rápido quanto ao resultado dos testes (passou/falhou).

O principal objetivo da automatização de processos é aprimorar o andamento do fluxo de trabalho em uma organização. Com a automatização, é possível reduzir custos, tempo, desperdícios, aumentar a produtividade, minimizar falhas e controlar, em tempo real, todos os processos do negócio. São diversos os ganhos em relação à automação de processo manual – gera indicadores extremamente úteis para a gestão, ganhos em produtividade, poupa recursos, reduz as possibilidades de infringir as regras de integridade do processo, reduz e oferece controles de integridade de um processo.

A décima nona questão, feita sobre os fatores utilizados para se considerar automação de um teste. Como resposta, tem-se o Gráfico 19.

Gráfico 19: Organização das equipes de desenvolvimento



Fonte: O Autor

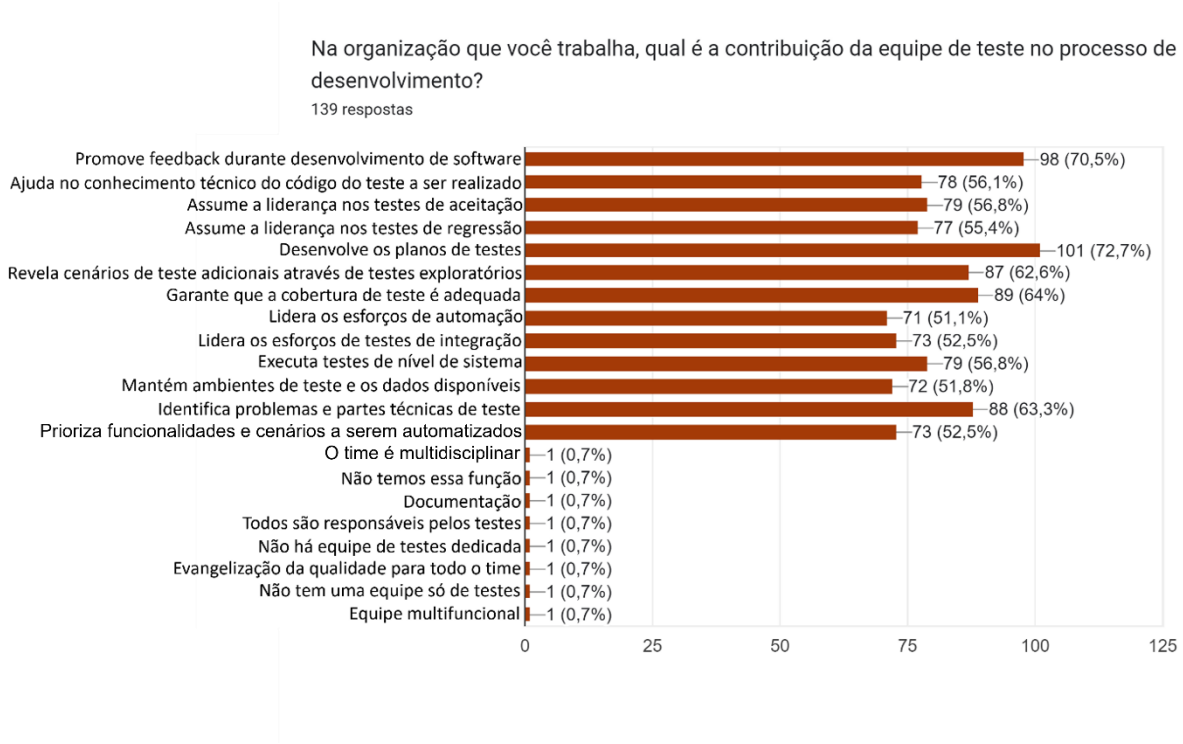
A resposta para esta questão indica que metade dos respondentes, ou seja, 55,4% dos respondentes optou pela opção que indica que a equipe de desenvolvimento é integrada, mas com separação de papéis entre desenvolvedores e testadores, seguidos de 30,2% que responderam que a equipe é integrada, com equipes multidisciplinares se sem separação de papeis e, 13,7% selecionaram a opção de que a equipe de testes é independente.

Uma única equipe com papéis diferenciados entre testadores e desenvolvedores permite identificar erros durante as etapas de desenvolvimento, garante a confiança do usuário final e sua satisfação ao utilizar o software, permite assegurar a qualidade do produto e seu funcionamento correto. É fundamental para manter a reputação do negócio no setor.

Uma razão pela qual os respondentes optaram pela opção de a equipe de testes ser independente é porque muitas pessoas que responderam o questionário trabalham em consultorias especializadas em testes de software, o qual as proporciona vivências em diversos tipos de projetos diferenciados entre si.

A vigésima questão, feita sobre a contribuição da equipe de testes no processo de desenvolvimento de software. Como resposta, tem-se o Gráfico 20.

Gráfico 20: Contribuição da equipe de testes



Fonte: O Autor

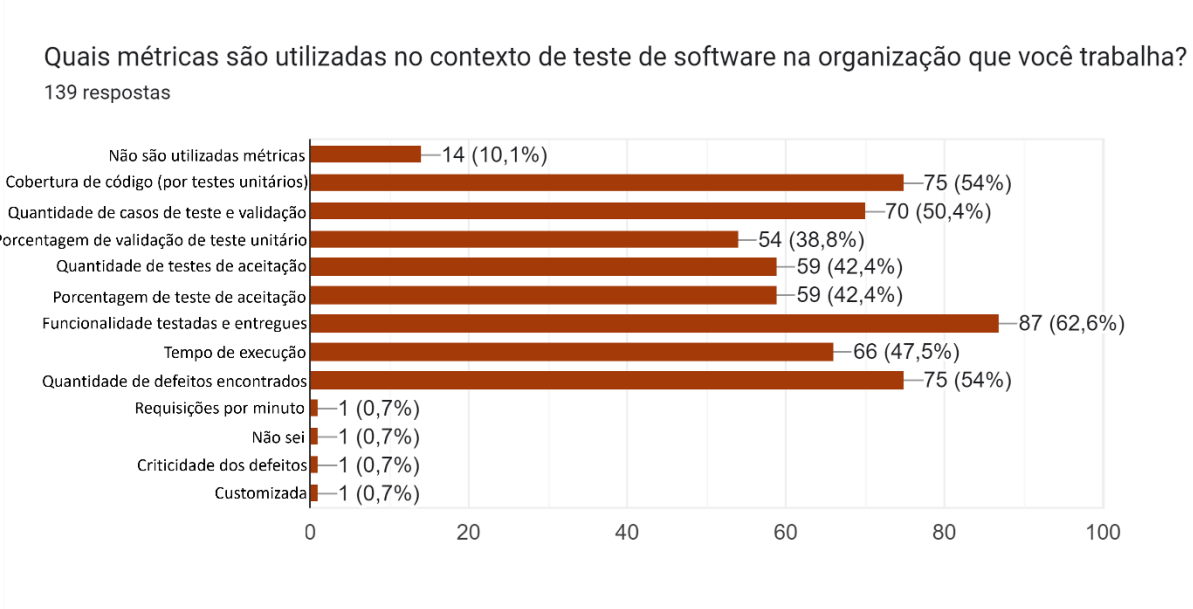
A resposta para esta questão indica que grande parte dos respondentes selecionaram as opções de que promove feedback durante todo o processo de desenvolvimento, ajuda com conhecimento técnico de implantação do código do teste a ser realizado, assume a liderança nos testes de aceitação, assume a liderança nos testes de regressão, desenvolve os planos de testes, revela cenários de teste adicionais através de testes exploratórios, garante que a cobertura de teste é adequada, lidera os esforços de automação, lidera os esforços de testes de integração, executa testes de nível de sistema, mantém ambientes de teste e os dados disponíveis e identifica problemas e partes técnicas de teste.

A vigésima primeira questão, foi feita de forma aberta, para que respondente possa colocar qual prática de teste de software sentiu falta no questionário.

As respostas foram em grande maioria, não. Teve um respondente que informou a opção de teste de usabilidade e outro que sentiu falta de ferramentas figma e mantis. Outro respondente comentou sobre TDD (Test Driven Development).

A vigésima segunda questão, foi feita para entender as métricas utilizadas nas organizações que os respondentes atuam. Como resposta, tempos o Gráfico 21.

Gráfico 21: Métricas de testes de software



Fonte: O Autor

As respostas foram as seleções das opções disponíveis e, alguns respondentes incluíram na categoria outros, métricas de teste de software utilizadas em suas organizações.

A última questão deste questionário foi informando se o respondente desejaria receber estas respostas por e-mail, e caso positivo, foi solicitado que ele informasse seu endereço de e-mail. Para manter a privacidade dos respondentes, estes dados não serão publicados no trabalho, mas, houve um total de 28 respondentes que desejaram receber os resultados desta pesquisa.

As métricas auxiliam o gestor a identificar quaisquer problemas e, dessa forma, fazer uma correção de rota sempre que essa ação for necessária. Assim é possível flexibilizar a empresa, evitando que o caminho traçado no planejamento acabe levando todos à ruína.

Aqui é apresentada uma análise preliminar da pesquisa sem ainda serem feitas análises de correlação entre os dados. Apesar disso é possível notar que todas as práticas consideradas no *Roadmap* são utilizadas.

3.3.4.1. Análise de dados

A análise de dados é fundamental para uma ampla variedade de campos e setores, desde negócios e finanças até ciência e tecnologia. Ela permite extrair *insights* significativos a partir de grandes volumes de informações, identificar tendências, padrões e anomalias, tomar decisões informadas, desenvolver estratégias eficazes e prever eventos futuros.

Com a análise dos dados do questionário, busca-se entender mais a fundo as características das respostas e obter informações sobre o nível de conhecimento dos respondentes em relação as práticas de teste de software e sua aplicação, ou seja, sua frequência de uso nas organizações.

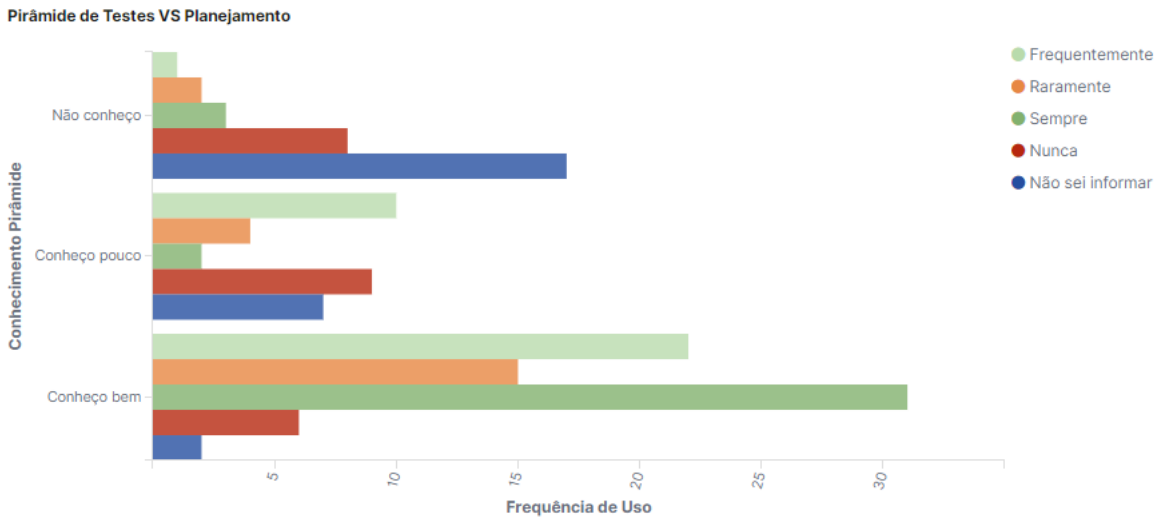
Para estas análises, o Gráficos 15 e Gráfico 16 serão utilizados a fim de se realizar comparações entre o conhecimento teórico dos respondentes ao conhecimento prático.

As análises buscaram comparar as informações de nível de conhecimento teórico e sua aplicabilidade.

3.3.4.2. Utilização da Pirâmide de Testes

Por meio do Gráfico 22, é possível visualizar a junção das respostas em relação ao nível de conhecimento em relação a pirâmide de testes e, a frequência de uso nas organizações para o planejamento da quantidade de testes por nível da pirâmide de testes, ou seja, por meio destas informações, será possível visualizar a relação entre conhecimento teórico e conhecimento aplicado, de acordo o Gráfico 22.

Gráfico 22: Análise de dados – Pirâmide de Testes



Fonte: O Autor

Com base no Gráfico 22, é possível observar cenários positivos e cenários negativos quanto ao alinhamento entre conhecimento teórico e conhecimento aplicado na pirâmide de testes.

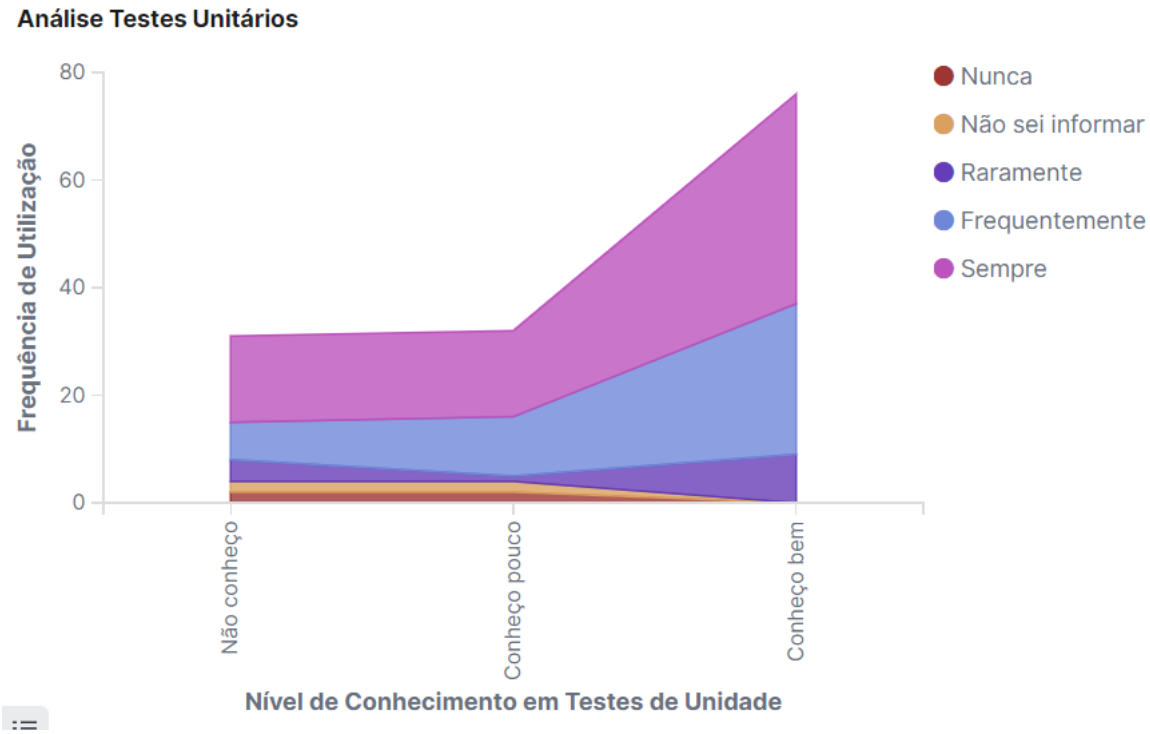
Como pode ser observado no Gráfico 22, pode-se observar o cenário de respondentes que informaram não conhecer a pirâmide de testes percebe-se a maioria optando pelas opções de não sei informar ou nunca aplicam o planejamento de testes utilizando a pirâmide de testes. Ao mesmo tempo, percebe que o mesmo grupo de respondentes que informaram não conhecer de pirâmide testes, selecionarem as opções que utilizam sempre, frequentemente e raramente, provocando uma inconsistência e divergência em relação a utilização de uma prática não conhecida.

Com a análise do cenário de respondentes que informaram conhecer a pirâmide de testes percebe-se a maioria optando pelas opções uso frequente e sempre para a aplicação do planejamento de testes utilizando a pirâmide de testes. Ao mesmo tempo, percebe que o mesmo grupo de respondentes que informaram conhecer de pirâmide testes, selecionarem as opções que utilizam raramente, nunca e, alguns respondentes, selecionaram a opção não sei informar. Este cenário apresenta divergências em relação ao conhecimento de uma prática e sua não utilização.

Por meio desta análise, percebe-se que as organizações precisam de um guia orientador de melhoria de teste de software para analisarem o processo de teste de software e, desta maneira, melhorar os processos que, podem ser melhorados simplesmente com a divulgação e conhecimento e aplicação do conhecimento em relação às práticas de teste de software.

A análise do cenário de conhecimento sobre pirâmide de testes e análise sobre a frequência de uso de testes unitários pode ser vista por meio do Gráfico 23.

Gráfico 23: Análise de dados – Testes de unidade



Fonte: O Autor

Com base no Gráfico 23, é possível observar cenários positivos e cenários negativos quanto ao alinhamento entre conhecimento teórico e conhecimento aplicado na pirâmide de testes.

A análise dos dados revela uma lacuna significativa na adoção e prática de testes unitários entre os entrevistados. De acordo com os resultados, apenas 54.68% dos entrevistados conhecem bem a pirâmide de testes, o que sugere que há uma oportunidade de melhoria na compreensão e aplicação de práticas de teste de software em geral.

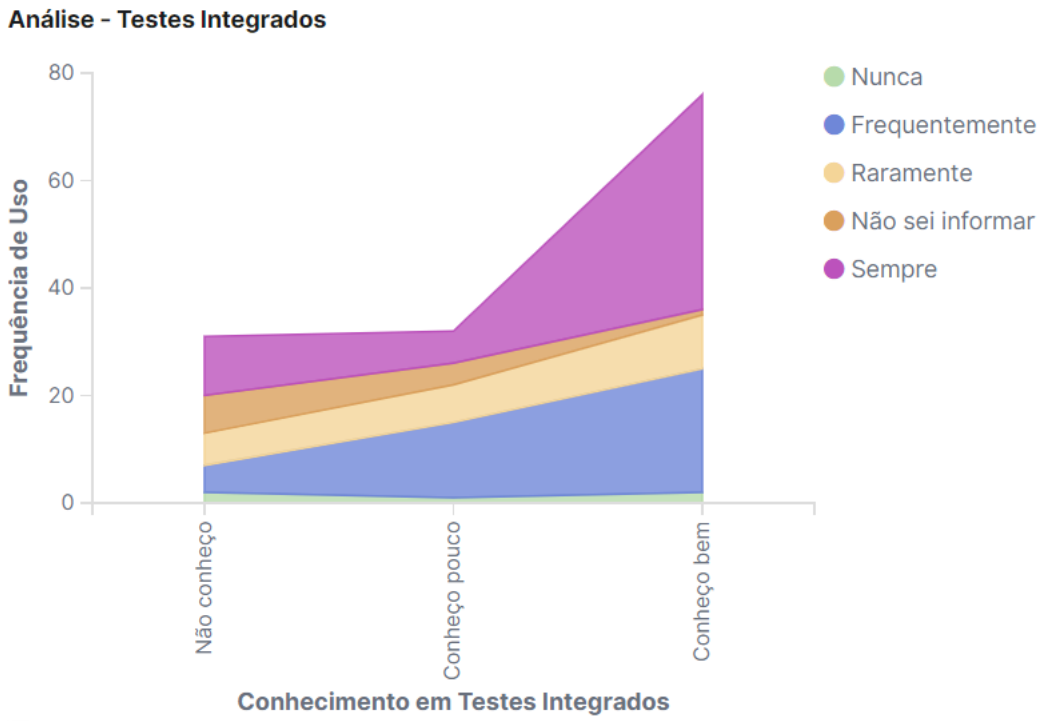
Dos entrevistados que conhecem bem a pirâmide de testes, 11.84% relataram raramente usar testes unitários, enquanto nos 23.02% que conhecem pouco, o percentual dos que raramente e nunca usam testes unitários é ainda maior, totalizando 51.08% e 2.88%, respectivamente.

Esses dados indicam claramente uma necessidade de melhorias na adoção e prática de testes unitários entre os entrevistados. Uma maior conscientização sobre a importância dos testes unitários e a aplicação adequada dessas práticas podem melhorar significativamente a qualidade do software e reduzir os custos de manutenção a longo prazo. Portanto, é importante

investir em treinamento e capacitação em testes unitários para preencher essa lacuna e garantir melhores práticas de teste de software em geral.

A análise do cenário de conhecimento sobre pirâmide de testes e análise sobre a frequência de uso de testes de integração pode ser vista por meio do Gráfico 24.

Gráfico 24: Análise de dados – Testes de Integração



Fonte: O Autor

Como observado no Gráfico 24, os resultados da pesquisa indicam que, entre os entrevistados que conhecem bem a pirâmide de testes, uma proporção significativa ainda não está aplicando adequadamente as práticas de teste de integração em suas atividades de desenvolvimento de software.

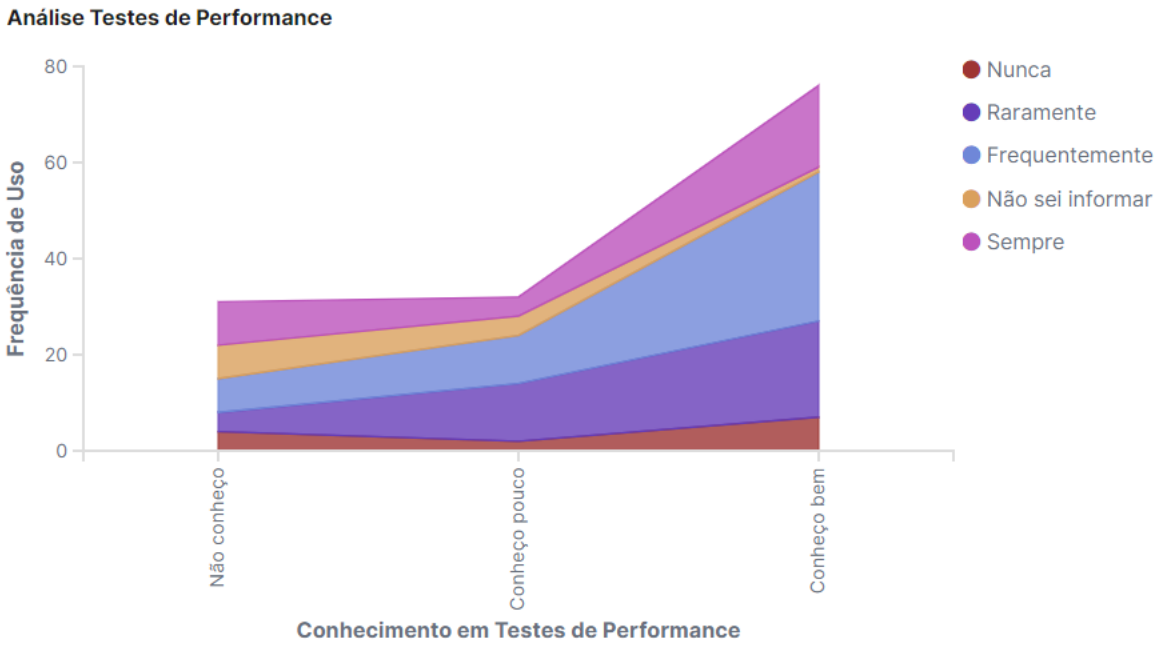
Dos 54.68% que conhecem bem a pirâmide de testes, 13.16% relataram raramente usar esse tipo de testes, enquanto 2.63% afirmaram nunca ter utilizado e 1.32% declarou não saber. Isso indica uma oportunidade de melhoria na compreensão e adoção de práticas de teste de integração, que são importantes para garantir a funcionalidade, desempenho e integridade dos sistemas em um ambiente distribuído.

Os testes de integração são essenciais para garantir que os sistemas funcionem corretamente em diferentes plataformas e condições, bem como para identificar problemas de integração e comunicação entre os componentes. Ao se adotar as melhores práticas de teste de integração, é possível melhorar a qualidade do software, reduzir custos de manutenção e

aumentar a satisfação dos usuários finais. Por isso, é crucial que as empresas incentivem e promovam a adoção de práticas de teste de serviço como parte fundamental do processo de desenvolvimento de software.

A análise do cenário de conhecimento sobre pirâmide de testes e análise sobre a frequência de uso de testes de desempenho e testes de estresse, caracterizados como testes não funcionais pode ser vista por meio do Gráfico 25.

Gráfico 25: Análise de dados – Pirâmide de Testes



Fonte: O Autor

Como pode ser visto no Gráfico 25, a maioria dos entrevistados conhece bem a pirâmide de testes, mas uma proporção significativa não aplica adequadamente as práticas de teste de performance e stress. Dos que conhecem a pirâmide, 26,32% raramente usam testes de performance, 9,21% nunca usaram e 1,32% não sabem o que é. Isso revela uma lacuna na adoção dessas práticas que são essenciais para garantir a qualidade do software.

É crucial que os desenvolvedores entendam e apliquem essas práticas para garantir a capacidade do software de lidar com cargas de trabalho pesadas e atender às expectativas dos usuários. Adotar as melhores práticas pode melhorar a qualidade do software, reduzir custos de manutenção e aumentar a satisfação dos usuários finais. Fica evidente a necessidade de uma promoção de melhoria das práticas de teste software.

De forma resumida, os dados apresentados nos gráficos sobre análise de resultados podem ser encontrados, de forma resumida, por meio da Tabela 3:

Tabela 3: Visão Geral – Pirâmide de testes

Tipo de Teste	Porcentagem que conhece bem a pirâmide de testes	Porcentagem que conhece pouco da pirâmide de testes	Porcentagem que raramente utiliza	Porcentagem que nunca utiliza	Porcentagem que não sabe
Testes Unitários	54.68%	23.02%	11.84%	51.08%	2.88%
Testes Funcionais	54.68%	-	14.47%	5.26%	-
Testes de Serviço	54.68%	-	13.16%	2.63%	1.32%
Testes de Performance	54.68%	-	26.32%	9.21%	1.32%

Fonte: O Autor

De acordo a Tabela 3, a análise dos dados mostra que dos 54.68% respondentes que conhecem bem a pirâmide de testes, 11.84% raramente utilizam os testes unitários. Em relação aos 23.02% dos respondentes que conhecem pouco, o percentual dos que raramente e nunca utilizam testes unitários é de 51,08% e 2.88%, respectivamente, indicando uma grande lacuna e oportunidade de melhoria para adoção e práticas de teste de software.

Quanto aos testes funcionais, 14.47% dos respondentes que conhecem bem a pirâmide de testes raramente os utiliza e 5.26% nunca os utilizam e, em relação aos testes de serviço ou testes de integração, 13.16% dos respondentes que conhecem bem a pirâmide de testes raramente os utilizam, 2.63% nunca os utilizaram e 1.32% não sabem o que são.

Por fim, em relação aos testes de performance, 26.32% dos respondentes que conhecem bem a pirâmide de testes raramente os utiliza, 9.21% nunca os utilizam e 1.32% não sabem do que se trata.

De forma resumida, com base nas análises dos gráficos dos respondentes, percebe-se nitidamente a oportunidade de mercado para um *roadmap* de melhoria de teste de software, capaz de realizar o mapeamento das práticas de teste utilizadas pela organização e, propor, por meio de planos de melhoria, o aumento do nível de maturidade em teste de software e, aumento da frequência de uso de práticas de teste de software, com base na pirâmide de testes.

4. *ROADMAP* ROMTES

O *roadmap* RoMTeS, acrônimo para "*Roadmap* de Melhoria de Teste de Software" é um *roadmap* desenvolvido para responder a três questões. a) Para onde estamos indo? No tocante às práticas de teste de software aplicadas aos projetos de desenvolvimento de software. b) Onde nos encontramos agora? Uma visão clara e direta dos processos e práticas de teste de software aplicadas na organização, antes da aplicação do *roadmap.* c) Como podemos chegar lá? Resposta que será dada com a execução de cada um dos passos do *roadmap* RoMTeS.

Este roadmap foi desenvolvido como um guia orientador do processo de melhoria de teste de software das organizações que atuem com o desenvolvimento de micro serviços.

O roadmap foi construído em etapas, iniciando pela criação da sua versão 1.0. Em seguida, três profissionais especialistas em testes de software analisaram minuciosamente o documento e fizeram sugestões para melhorias e pontos que não eram essenciais (Apêndice 2). Após as entrevistas, o roadmap foi revisado e finalizado, tornando-se um guia orientador para aprimoramento dos processos de teste de software em organizações.

4.1. Entrevistas

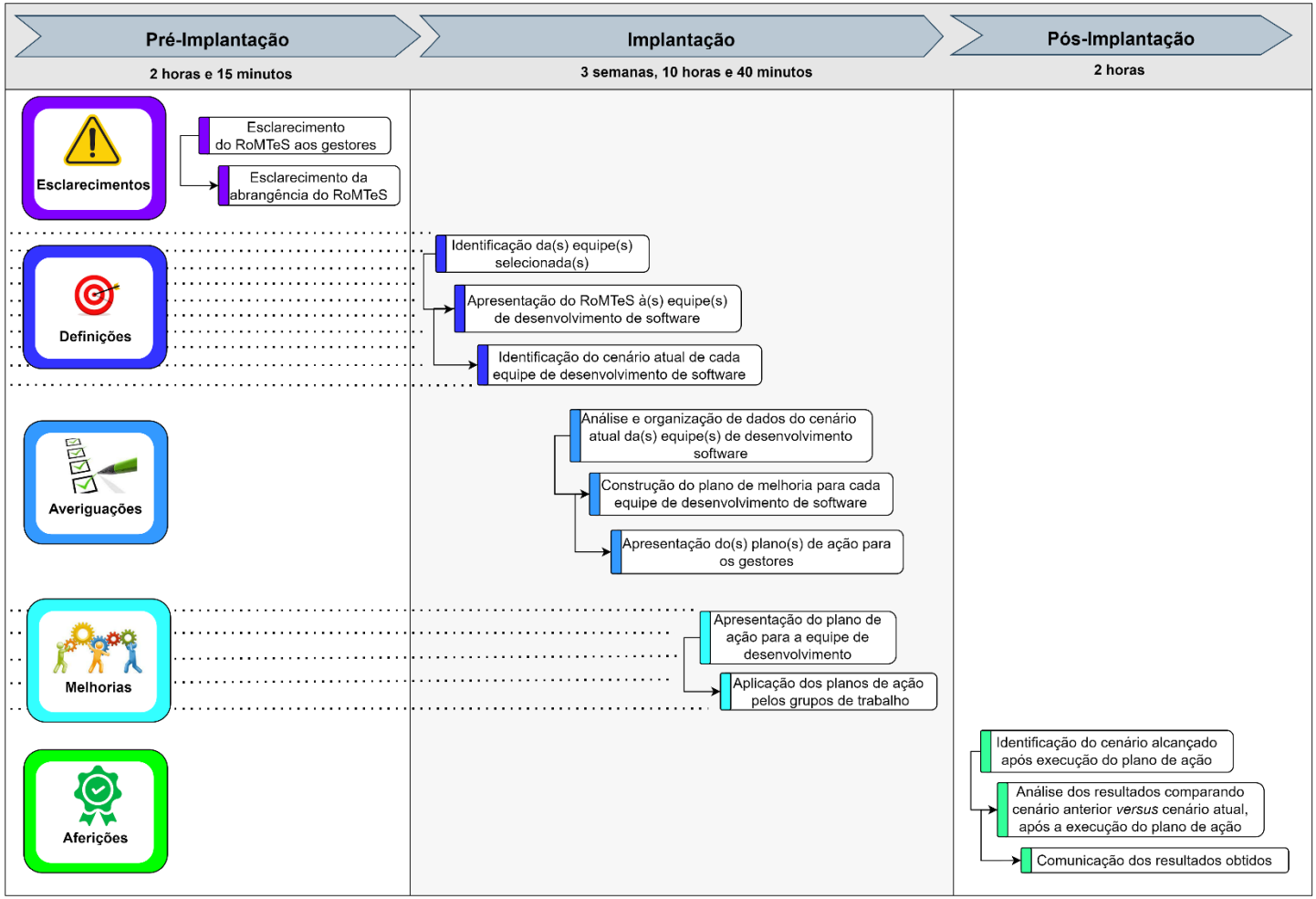
Foram realizadas entrevistas com três especialistas na área de testes de software para análise do *roadmap* de melhoria de teste de software. Estas entrevistas estão no Apêndice 2 deste trabalho.

Após as entrevistas, o RoMTeS teve modificações, a fim de aprimorar as melhorias levantadas durante as entrevistas e, a descrição do *roadmap*, bem como de suas etapas já comtempla as melhorias levantadas nas entrevistas com especialistas.

4.2. Etapas do *roadmap* RoMTeS

Em uma visão geral, pode-se observar o *roadmap* RoMTeS, em suas três etapas, pré-implantação, implantação e pós-implantação por meio da Figura 4.

Figura 4: Visão geral do RoMTeS

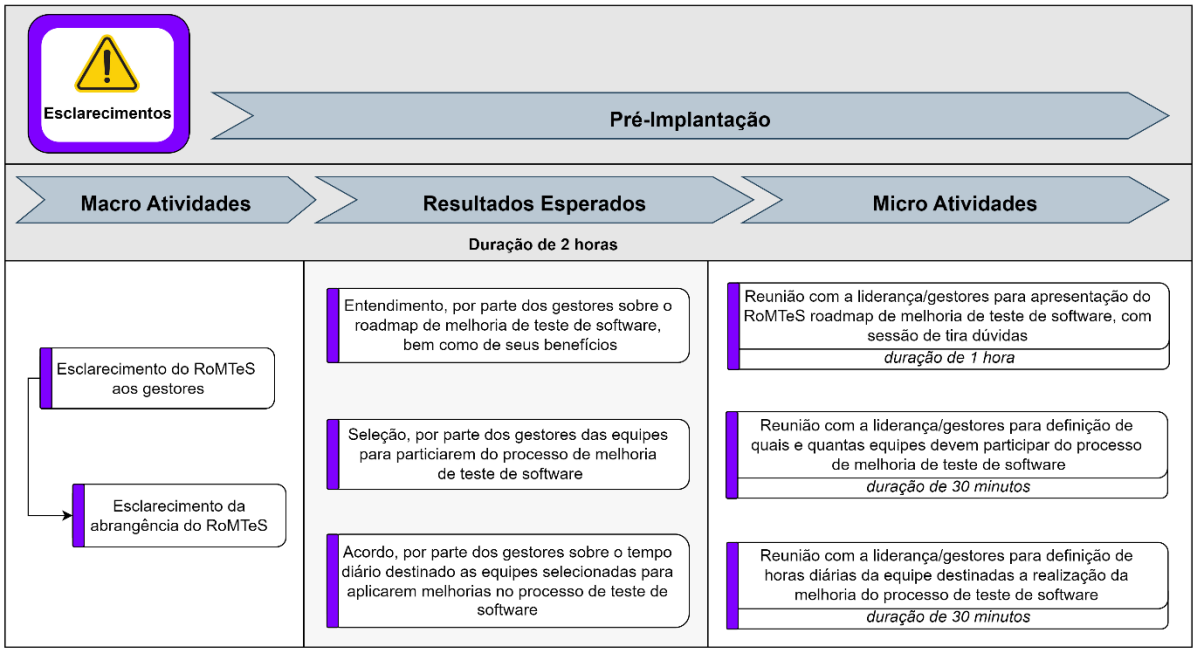


Fonte: O Autor

Conforme pode ser observado na Figura 4, a primeira etapa do *roadmap*, chamada de pré-implantação, consiste em apresentar o *roadmap* à organização e fornecer esclarecimentos iniciais, o que leva aproximadamente 2 horas e 15 minutos. Já a segunda etapa, denominada implantação, é composta pelas fases de definição, investigação e melhoria, na qual o *roadmap* é apresentado às equipes técnicas e são coletados dados para identificar o cenário atual. Nessa etapa, a equipe é preparada e as melhorias em teste de software são aplicadas, levando cerca de 3 semanas, 10 horas e 40 minutos. No entanto, o tempo total pode variar de acordo com as necessidades específicas da organização.

Na etapa de pós-implantação, ocorre a fase de aferições, que consiste na avaliação das práticas de teste de software aplicadas, na análise do cenário atual em comparação com o cenário anterior e na comunicação dos resultados aos gestores, totalizando aproximadamente 2 horas. Cabe ressaltar que os tempos estipulados levam em conta as atividades necessárias em cada etapa, podendo variar de acordo com os objetivos específicos da organização. Na sequência, será apresentada a descrição das atividades executadas na etapa de esclarecimentos, conforme apresentado na Figura 5:

Figura 5: RoMTeS: Etapa de esclarecimentos



Fonte: O Autor

Como visto, a etapa de pré-implantação é composta pela etapa de esclarecimentos, que é responsável por realizar os esclarecimentos iniciais do *roadmap*. Isso inclui a apresentação

do *roadmap* aos gestores e liderança das organizações, bem como a explicação da abrangência do *roadmap*. Durante essa etapa, é importante que os gestores identifiquem as equipes que serão abordadas na implantação do *roadmap* e o tempo semanal que cada equipe poderá dedicar às melhorias de teste de software. Essas atividades geralmente levam cerca de 2 horas para serem concluídas.

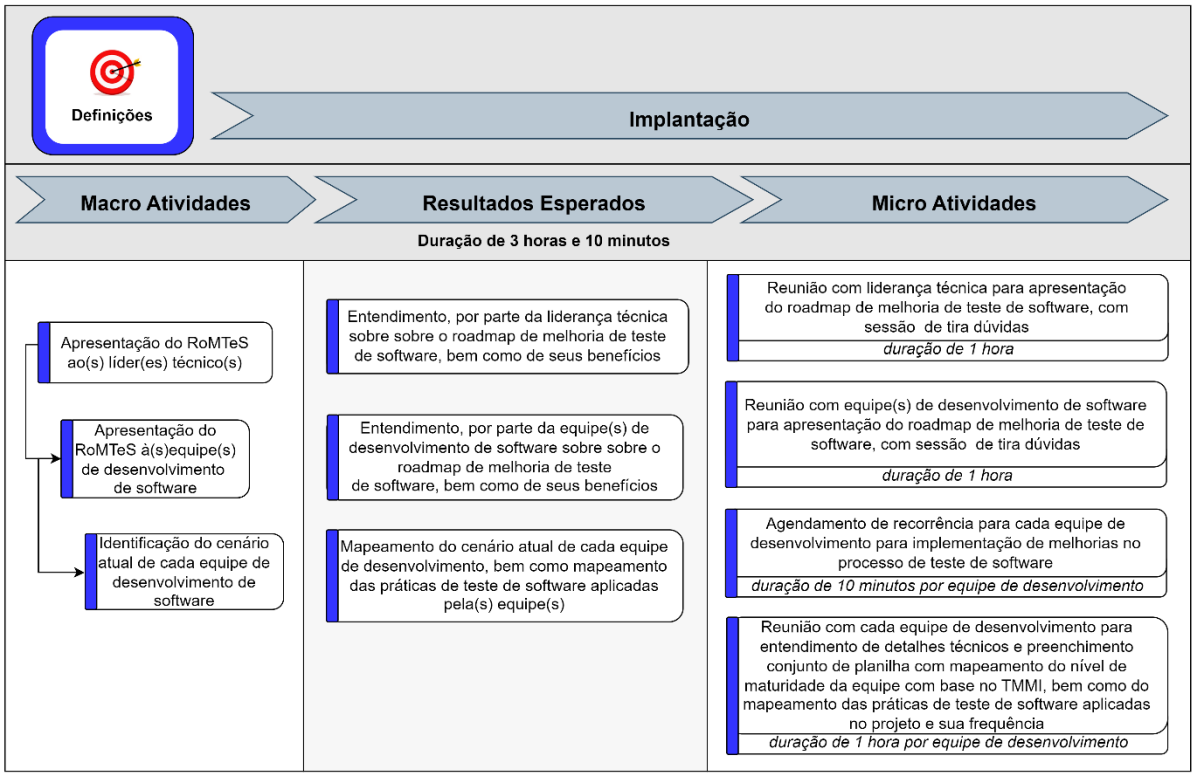
Como atividades centrais desta etapa, a primeira atividade é o esclarecimento do RoMTeS aos gestores. Para isso, será realizada uma reunião com a liderança/gestores da organização para apresentação do *roadmap* de melhoria de teste de software, com tempo dedicado para esclarecimentos de dúvidas. Essa reunião apresentará o *roadmap* e suas etapas, com duração média de uma hora. O resultado esperado dessa atividade é que os gestores compreendam o *roadmap* de melhoria de teste de software, entendam seus benefícios e possam esclarecer suas dúvidas sobre ele.

Outra atividade central é esclarecer a abrangência do RoMTeS, que será realizada por meio de uma reunião com a liderança/gestores da organização para definir quais equipes serão envolvidas no processo de melhoria de teste de software. Estima-se que essa tarefa leve cerca de meia hora. O resultado esperado é a seleção das equipes pelos gestores para participar do processo de melhoria de teste de software.

A atividade de esclarecimento da abrangência do RoMTeS também inclui uma reunião com a liderança/gestores para definir quantas horas diárias cada equipe deve dedicar para participar do processo de melhoria de teste de software. Essa tarefa leva em média meia hora e o resultado esperado é o acordo dos gestores quanto ao tempo destinado para cada equipe na implantação do *roadmap* e do plano de ação. É sugerido um tempo mínimo de 2 reuniões de uma hora e meia com cada equipe de desenvolvimento ao longo de 3 semanas para aplicação dos planos de melhoria.

A seguir, teremos a descrição das atividades executadas na etapa de definições, a qual segue na sequência da aplicação do *roadmap* RoMTeS, por meio da Figura 6.

Figura 6: RoMTeS: Etapa de definições



Fonte: O Autor

Como apresentado na Figura 6, a etapa de implantação segue a etapa de esclarecimentos, iniciando-se com a etapa de definições, a qual é composta pelas atividades de apresentação do RoMTeS aos líderes técnicos, apresentação do RoMTeS às equipes de desenvolvimento de software e identificação do cenário atual de cada equipe de desenvolvimento de software.

A etapa de implantação é composta pela etapa de definições, que inclui como atividades centrais a apresentação do RoMTeS aos líderes técnicos e a identificação do cenário atual de cada equipe de desenvolvimento de software. A primeira atividade consiste em uma reunião com a liderança técnica da organização para apresentação do *roadmap* de melhoria de teste de software, com um tempo dedicado na reunião para esclarecer dúvidas. Durante essa reunião, serão apresentadas todas as etapas do *roadmap*, com uma duração média de uma hora. O objetivo dessa atividade é que os líderes técnicos compreendam o *roadmap* de melhoria de teste de software, seus benefícios e possam esclarecer suas dúvidas relacionadas a ele.

Como segunda atividade, será realizada a apresentação do RoMTeS às equipes técnicas de desenvolvimento de software. Esta atividade será realizada por meio de uma reunião com as equipes envolvidas na melhoria do teste de software para apresentação do *roadmap* e esclarecimentos de dúvidas. A reunião terá uma duração média de uma hora e se concentrará

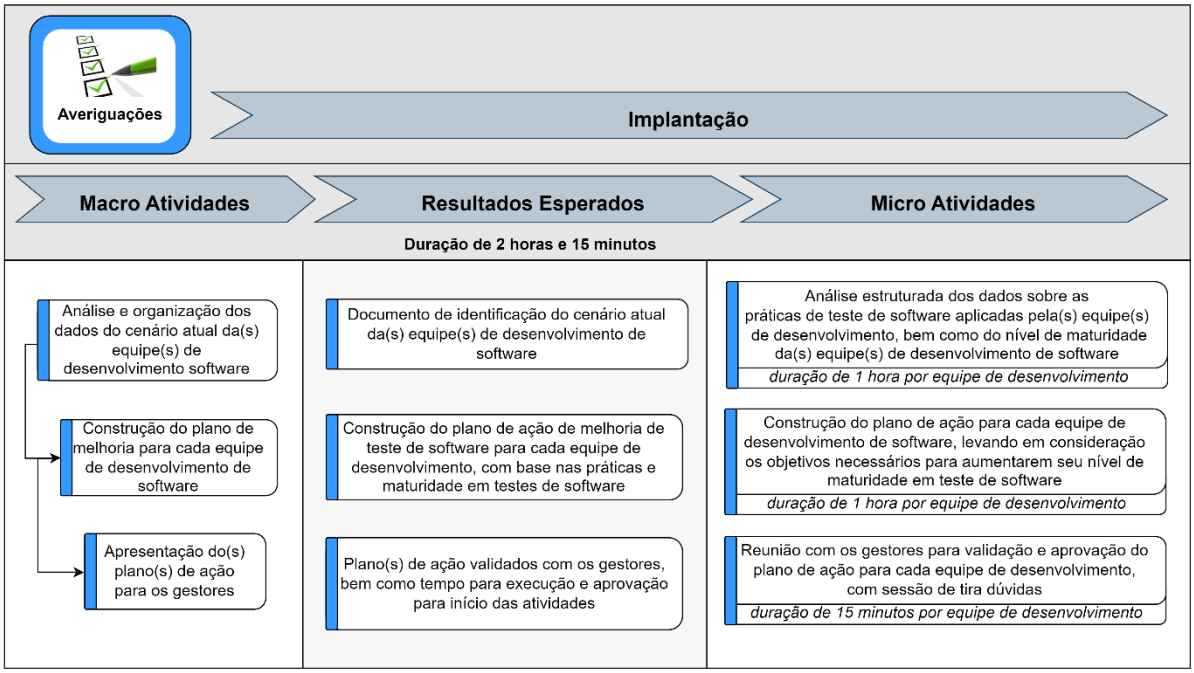
em aspectos mais técnicos do *roadmap*. O resultado esperado é que as equipes de desenvolvimento de software entendam o *roadmap* de melhoria de teste de software, seus benefícios e esclareçam suas dúvidas.

Uma tarefa importante inclusa nesta atividade é agendar reuniões regulares com cada equipe de desenvolvimento para implementação de melhorias no processo de teste de software. O tempo estimado para essa tarefa é de 10 minutos e seu objetivo é definir um horário que seja adequado para todas as equipes de desenvolvimento participarem da aplicação das melhorias no processo de teste de software.

A terceira atividade consiste na identificação do cenário atual de cada equipe de desenvolvimento de software. Essa etapa requer uma reunião com cada equipe de desenvolvimento, com duração média de uma hora, para entender os detalhes técnicos e preencher uma planilha em conjunto, mapeando as práticas de teste de software que a equipe pratica e com que frequência. Isso é necessário para avaliar o nível de maturidade da equipe em testes de software, com base no modelo de maturidade TMMi, (consulte Apêndice 03 para relação entre o RoMTeS e o TMMi) e mapear as práticas de teste de software aplicadas. Essa etapa é fundamental para obter uma visão atual da organização antes da implementação do *roadmap* RoMTeS. O resultado esperado é a identificação do cenário atual das equipes de desenvolvimento, o mapeamento das práticas de teste de software aplicadas e sua frequência, bem como a identificação do nível de maturidade da equipe de desenvolvimento de software com base no TMMi.

A seguir, serão descritas as atividades executadas na etapa de Averiguações, que se segue à aplicação do *roadmap* RoMTeS, conforme ilustrado na Figura 7.

Figura 7: RoMTeS: Etapa de Averiguações



Fonte: O Autor

Como pode ser observado na Figura 7, a etapa de implantação é composta também pela etapa de averiguações, que tem como responsabilidade a análise e organização dos dados coletados no cenário atual das equipes de desenvolvimento de software, além da construção do plano de melhoria personalizado para cada equipe e apresentação dos planos de ação para os gestores. Essa etapa tem uma duração aproximada de 2 horas e 15 minutos.

Como atividades centrais desta etapa, temos como primeira atividade a análise e organização dos dados do cenário atual das equipes de desenvolvimento de software. Essa atividade busca realizar uma análise estruturada dos dados coletados sobre as práticas de teste de software aplicadas pelas equipes de desenvolvimento, bem como o nível de maturidade em teste de software de cada uma delas. Para isso, será realizada uma reunião de uma hora com cada equipe de desenvolvimento. Como resultado esperado, espera-se a geração de um documento que identifique o cenário atual das equipes de desenvolvimento de software, contendo as práticas de teste de software utilizadas e a frequência com que são aplicadas, bem como a documentação do nível de maturidade em teste de software de cada equipe.

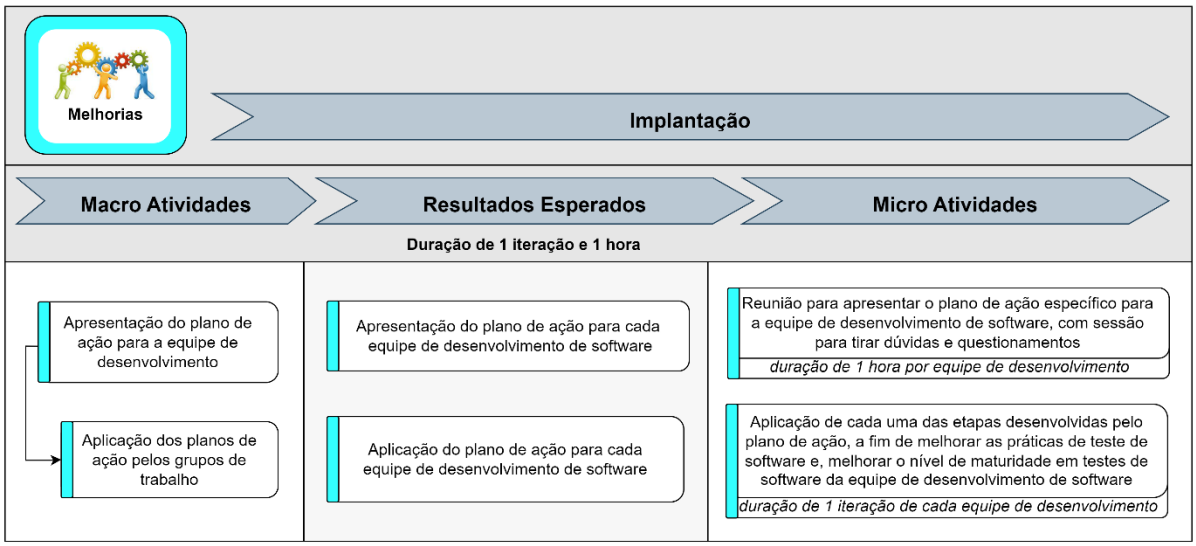
A segunda atividade consiste na elaboração de um plano de melhoria para cada equipe de desenvolvimento de software. Esse plano inclui a definição de ações específicas para aumentar o nível de maturidade em testes da equipe ou para completar as atividades necessárias para atingir o nível de maturidade em teste de software. Sugere-se que seja utilizado o modelo

de maturidade TMMi. Para a elaboração desse plano, será levado em consideração o conhecimento prévio da equipe sobre cada prática de teste, bem como as práticas já aplicadas no cotidiano da equipe. Essa atividade demanda 1 hora para cada equipe de desenvolvimento e resultará na criação de um plano de ação personalizado para cada grupo de trabalho, com o objetivo de melhorar o nível de maturidade em testes de software.

A terceira atividade desta etapa é apresentar os planos de ação aos gestores. Essa tarefa é realizada por meio de reunião com os gestores, com o objetivo de apresentar o plano de ação elaborado para cada equipe de desenvolvimento. Para cada equipe, são dedicados 15 minutos de apresentação. Como resultado desta etapa, temos a apresentação aos gestores dos planos de ação para cada equipe de desenvolvimento, incluindo o tempo previsto para a execução de cada tarefa.

A seguir, teremos a descrição das atividades executadas na etapa de implantação, a atividade de melhorias, a qual segue na sequência da aplicação dos planos de ação do *roadmap* RoMTeS, por meio da Figura 8.

Figura 8: RoMTeS: Etapa de Melhorias



Fonte: O Autor

Na Figura 8, pode-se observar que a etapa de implantação inclui a etapa de melhorias, a qual abrange duas atividades centrais. A primeira é a apresentação do plano de ação para a equipe de desenvolvimento, na qual é realizada uma reunião com a equipe para apresentar o plano de ação elaborado na etapa anterior. Essa atividade tem como objetivo garantir que todos

os membros da equipe estejam cientes das ações que serão realizadas para aprimorar o nível de maturidade em teste de software. O tempo dedicado para essa tarefa é de 30 minutos por equipe de desenvolvimento.

A segunda atividade é a aplicação dos planos de ação em parceria com os grupos de trabalho. Nessa etapa, as ações previstas no plano são executadas em conjunto com a equipe de desenvolvimento, e o progresso é acompanhado para garantir que os objetivos sejam alcançados. O tempo dedicado para essa atividade é de uma iteração, ou seja, o tempo necessário para implementar as ações planejadas, que varia de acordo com o plano de ação de cada equipe.

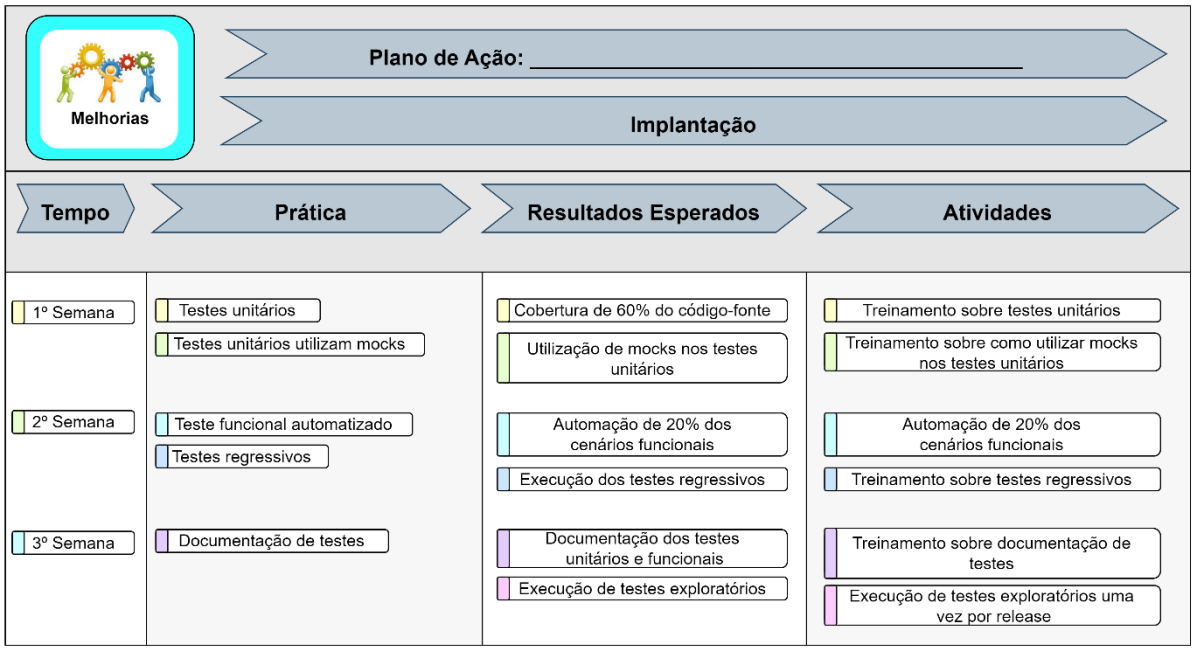
Em resumo, a etapa de melhorias tem duração de aproximadamente uma iteração, aproximadamente 3 semanas e uma hora e, tem como resultado a aplicação das ações previstas no plano de ação, visando aprimorar o nível de maturidade em teste de software da equipe de desenvolvimento.

Nesta etapa, as atividades principais incluem a apresentação do plano de ação para a equipe de desenvolvimento. Isso será feito por meio de uma reunião na qual o plano de ação específico para a equipe de desenvolvimento de software será apresentado, com uma sessão para tirar dúvidas e fazer perguntas. A atividade terá duração aproximada de uma hora e o resultado esperado é que a equipe de desenvolvimento tenha conhecimento do plano de ação, das práticas que serão trabalhadas e dos períodos dos encontros para aplicação das melhorias em teste de software.

A segunda atividade central desta etapa consiste na aplicação dos planos de ação pelos grupos de trabalho, o que levará em média três semanas, conforme acordado com o gestor em relação ao tempo necessário para a execução do plano. Nessa etapa, espera-se que cada etapa do plano de ação seja implementada para aprimorar as práticas de teste de software e aumentar o nível de maturidade em testes de software da equipe de desenvolvimento. É nessa atividade que se concentra o maior esforço do *roadmap*.

A seguir, teremos um modelo do plano de ação, por meio da Figura 9.

Figura 9: RoMTeS: Etapa de Melhorias – Plano de ação

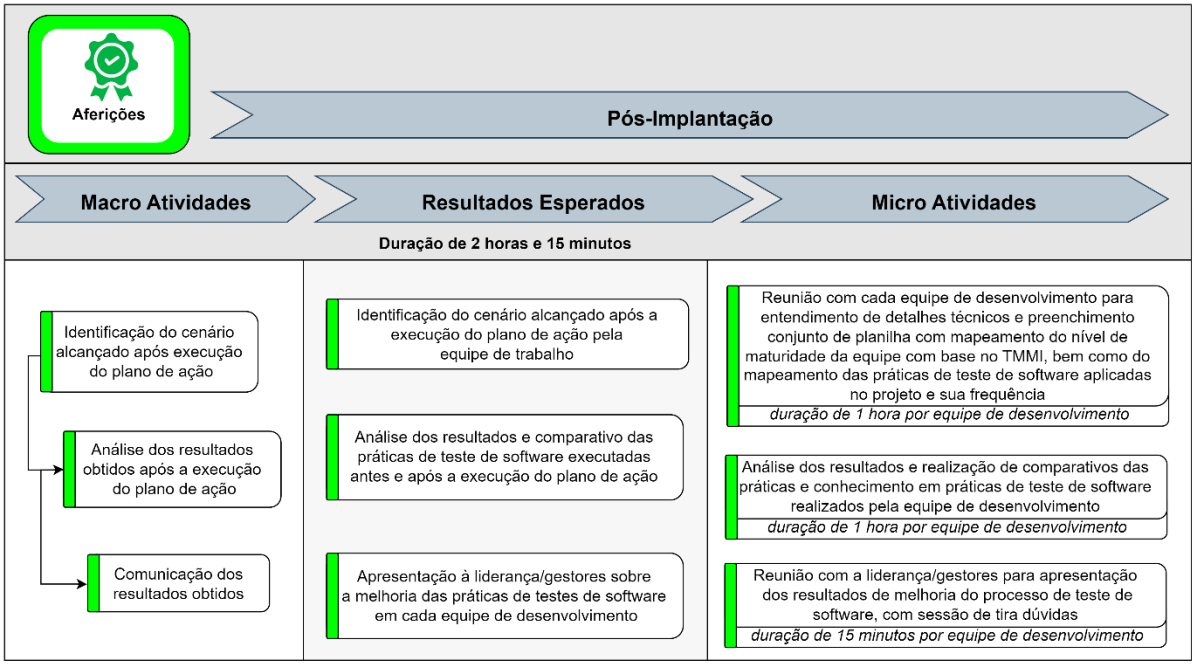


Fonte: O Autor

Conforme a Figura 9, o plano de ação consiste no planejamento das práticas de teste de software a serem abordadas durante as três semanas de implantação. Como exemplo, na primeira semana, são definidos dois temas a serem abordados, juntamente com os resultados esperados para cada um deles. É importante ressaltar que a maioria das atividades consiste em treinamentos com aplicação prática utilizando as tecnologias utilizadas pela equipe de desenvolvimento, visando melhorar tanto os testes de software quanto o nível de maturidade em teste de software das equipes.

A seguir, teremos a descrição das atividades executadas na etapa de pós-implantação, a atividade de aferição, a qual segue na sequência da aplicação do *roadmap* RoMTeS, por meio da Figura 10.

Figura 10: RoMTeS: Etapa de Aferição



Fonte: O Autor

Como ilustrado na Figura 10, a etapa de pós-implantação é composta pela etapa de aferição, que é responsável por avaliar o cenário alcançado após a execução do plano de ação, analisar os resultados obtidos, compará-los ao cenário inicial e comunicar os resultados aos líderes e gestores.

Na primeira atividade, a identificação do cenário após a execução do plano de ação é feita por meio da aplicação de um questionário às equipes de desenvolvimento. O objetivo é coletar dados sobre o cenário atual e o nível de conhecimento em relação às melhores práticas de teste de software. Essa etapa repete o questionário feito no início do processo para identificação do cenário atual, mas desta vez para identificar o cenário e o nível de conhecimento dos membros da equipe de desenvolvimento em relação às práticas de teste de software e usabilidade no ambiente organizacional. Espera-se obter como resultado a identificação do cenário após a execução do plano de ação pela equipe de desenvolvimento. Cada membro da equipe levará em média 15 minutos para preencher o questionário.

A segunda atividade de aferição consiste na análise dos resultados obtidos após a execução do plano de ação. Nessa etapa, serão realizados comparativos das práticas e conhecimentos em testes de software realizados pela equipe de desenvolvimento, levando em consideração a tabulação dos dados e a análise comparativa de cenários. Essa atividade terá duração média de três horas para cada equipe de desenvolvimento. Como resultados esperados,

tem-se a identificação das melhorias alcançadas e dos pontos que precisam ser aprimorados, permitindo que os líderes e gestores avaliem o impacto do *roadmap* nos processos de teste de software.

A última etapa do *roadmap* RoMTeS consiste na comunicação dos resultados obtidos por meio da aplicação do plano de ação. Nessa etapa, será realizada uma reunião com a liderança, gestores e líderes técnicos para apresentação dos resultados da melhoria dos testes de software. A sessão terá como objetivo esclarecer dúvidas sobre o cenário anterior, ações realizadas e resultados atingidos.

Em questões de validação e indicadores de melhoria da utilização do *roadmap* de melhoria de teste de software, utiliza-se s frequência do uso das práticas de teste de software, de forma que um aumento no uso das práticas de teste de software é considerado como avanço em relação ao cenário inicial. Estima-se um tempo de 15 minutos para apresentação sobre cada equipe de desenvolvimento de software.

5. IMPLANTAÇÃO DO ROMTES EM EMPRESA DE PEQUENO PORTE

A implementação do *roadmap* RoMTeS na Organização Alfa, uma empresa de pequeno porte que desenvolve aplicações na área de análise de áudio com modelos de inteligência artificial, foi um processo bastante empolgante. Todos os envolvidos se dedicaram às reuniões e atividades propostas para melhorar o processo de teste de software e alcançar resultados significativos.

Por questões de segurança da informação e proteção da marca da empresa, as informações reveladas neste relatório estão estritamente relacionadas à implementação do *roadmap* RoMTeS e não incluem outros dados sensíveis da organização.

Durante as conversas de implantação do *roadmap* RoMTeS, os líderes e gestores da Organização Alfa fizeram diversas perguntas para entender melhor o processo e organizaram o tempo para implantá-lo sem afetar os outros projetos em andamento.

A empresa em questão utiliza metodologias ágeis para o desenvolvimento de software e adota a arquitetura de micro serviços para a construção de suas aplicações, além de utilizar uma nuvem pública. É importante ressaltar que a equipe de desenvolvimento de software é responsável por criar aplicações de análise de áudio utilizando modelos de inteligência artificial e web, sendo que o desenvolvimento de software é considerado uma atividade meio para a organização.

Nesta organização, a equipe de desenvolvimento de software utiliza sua expertise para criar aplicações de inteligência artificial e web, sendo essa atividade meio para atender às necessidades da empresa como um todo.

A equipe de desenvolvimento de software é composta por sete profissionais, incluindo três desenvolvedores, um líder técnico, um Product Owner, um líder de equipe e um Coordenador técnico. Eles trabalham de forma remota e são responsáveis pelo desenvolvimento e teste das aplicações de inteligência artificial e Web. Quando necessário, a equipe trabalha em conjunto para resolver problemas e garantir a qualidade das entregas.

Em relação a metodologia de trabalho, a equipe utiliza o Scrum como metodologia de trabalho e segue a estrutura para gerenciamento de projetos. Durante o planejamento das iterações, são levados em consideração os requisitos relacionados às atividades de desenvolvimento e testes de software. Além disso, a equipe utiliza uma esteira de integração contínua para garantir a qualidade das aplicações. Para isso, é necessário que a cobertura mínima de testes unitários seja de 50% antes da publicação das aplicações no ambiente produtivo. A organização adota um tempo de iteração de 3 semanas, durante o qual são realizados tanto o desenvolvimento das histórias quanto os testes.

Uma curiosidade sobre a organização é que, durante as conversas sobre a pirâmide de testes e a importância de se considerar as quantidades adequadas para cada nível, todos na equipe de desenvolvimento demonstraram muito interesse e entusiasmo em relação às atividades de melhoria de teste de software. Como métricas da organização, existem a quantidade de testes realizados, utilizando como motivadores centrais da automação dos testes, a facilidade de automação e feedback mais rápido quanto ao resultado dos testes (se estes passaram ou falharam. Como métricas analisadas, esta organização prioriza a cobertura de código por meio da porcentagem de testes unitários existentes por linha de código.

A aplicação do *roadmap* aconteceu no período do mês de janeiro de 2023, ocupando o total de uma iteração de cada uma das equipes de desenvolvimento de software.

As etapas de implantação foram seguidas e, o tempo total de implantação do *roadmap* RoMTeS nesta organização durou um total de 1 iteração, 12 horas, tendo resultados muito promissores na melhoria do teste de software e no aumento da maturidade em teste de software da equipe de desenvolvimento de software.

Com base na estrutura da organização, elaborou-se um cronograma semanal para aplicação do *roadmap*, com margem para que houvesse adequações de horários, se necessário. O cronograma da Semana 01 pode ser visto por meio da Figura 11.

Figura 11: Cronograma Organização Alfa – Semana 01

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00	Esclarescimentos com Gestores	Apresentação do RoMTeS a equipe		Equipe1 - Avaliação do cenário atual	Apresentação dos planos
11:00					
12:00					
13:00					
14:00					
15:00	Esclarecimentos com Liderança			Equipe1 - Plano de ação	
16:00					
17:00					
18:00					

Fonte: O Autor

Com base na Figura 11, pode-se observar a primeira semana de implantação do *roadmap* RoMTeS como uma semana de esclarecimentos, definições e avaliação da equipe de desenvolvimento de software, além de elaboração do plano de ação e apresentação do plano de ação aos gestores e liderança da equipe. O cronograma da Semana 02, Semana03 e Semana 04 são idênticos, como pode ser visto por meio da Figura 12.

Figura 12: Cronograma Organização Alfa – Semana 02, 03 e 04

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00	Equipe1 - Encontro 01		Equipe1 - Encontro 02		Equipe1 - Encontro 03
11:00					
12:00					
13:00					
14:00					
15:00					
16:00					
17:00					
18:00					

Fonte: O Autor

Com base na Figura 12, pode-se observar que a segunda, terceira e quarta semana são iguais em termos de calendário. Isto se dá porque, de acordo o planejamento do *roadmap*, este tempo é destinado para aplicação do plano de plano de ação em parceria com a equipe de desenvolvimento de software. Já o cronograma da última semana de implantação do *roadmap*, a Semana 05 pode ser visto por meio da Figura 13.

Figura 13: Cronograma Organização Alfa – Semana 05

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00	Equipe1 - Reavaliação	Apresentação dos Resultados Gerais			
11:00					
12:00					
13:00					
14:00					
15:00	Organização dos dados				
16:00					
17:00					
18:00					

Fonte: O Autor

Com base na Figura 13, pode-se observar a última semana de implantação do *roadmap*, destinada a reavaliação da equipe de desenvolvimento de software, organização e análise dos dados de cenário alcançado e apresentação dos resultados obtidos aos gestores e liderança. A seguir teremos a descrição de implantação do RoMTeS, a se iniciar pela etapa de esclarecimentos, sua primeira etapa.

5.1. Etapa Esclarecimentos

A etapa de esclarecimentos teve duração de 2 horas em reunião com a liderança da equipe, contendo o coordenador técnico, o *Product Owner*, o líder técnico e o líder da equipe de desenvolvimento de software.

A atividade, chamada de “Esclarecimento do RoMTeS aos gestores” foi realizada por meio de uma reunião com a liderança da equipe, com duração de uma hora e, nesta reunião, foi apresentado a eles a estrutura do *roadmap*, bem como cada etapa e tempo médio definido para cada atividade e, conforme as dúvidas foram surgindo, foram sendo tratadas na reunião, de forma que ao final desta primeira reunião, todos os envolvidos na reunião puderam entender o *roadmap* de melhoria de teste de software.

A segunda atividade presente nessa etapa do *roadmap* é o “Esclarecimento da abrangência do RoMTeS”, esta atividade presentou uma segunda conversa com a liderança da equipe de desenvolvimento para elencar quais as equipes iriam participar do processo de melhoria de teste de software por meio da aplicação do RoMTeS, bem como a definição do tempo semanal investido com cada equipe de desenvolvimento de software. Todas estas definições foram feitas por meio de uma reunião de 1 meia hora, na qual ficou acordado a

aplicação do RoMTeS na equipe central de desenvolvimento de software e, seria aplicado um tempo de 3 horas semanais para encontros e reuniões para melhoria do processo de teste de software

Durante a reunião com a liderança da equipe de desenvolvimento de software, foram coletados os nomes e contatos de cada membro da equipe. Após a conclusão da reunião, foi enviado um e-mail para todos os participantes, contendo um resumo das discussões realizadas, bem como informações sobre o tempo de atuação e os líderes técnicos responsáveis por cada equipe de desenvolvimento selecionada. Adicionalmente, um e-mail foi enviado aos desenvolvedores da equipe central de desenvolvimento, juntamente com seus respectivos gestores e liderança, para informá-los sobre o processo de melhoria de teste de software e agendar uma reunião com eles.

Finalizada a etapa de esclarecimentos, pode-se seguir para a etapa Definições, mais bem descrita a seguir.

5.2. Etapa Definições

A etapa de definições teve duração de 3 horas e 30 minutos, subdivididos em reuniões que envolveram a liderança e a equipe de desenvolvimento de software.

Esta etapa do *roadmap* conta com a atividade de "Apresentação do RoMTeS aos líderes técnicos", atividade realizada por meio de reunião de uma hora com os líderes técnicos da equipe de desenvolvimento de software. Nesta reunião, foram apresentados os objetivos do *roadmap*, bem como algumas dúvidas levantadas durante a reunião foram sanadas.

Esta reunião durou 1 hora e teve grande importância pois, durante esta reunião, pediu-se o engajamento dos líderes técnicos e de seu grupo de trabalho devido aos benefícios obtidos ao final de sua implantação, tanto de melhoria do processo de testes de software, como capacitação das equipes de desenvolvimento e, também, aumento da qualidade das entregas e aplicações desenvolvidas.

Outra atividade presente nesta etapa do *roadmap*, é a "Apresentação do RoMTeS à(s) equipes de desenvolvimento de software". Esta atividade abrangeu todos os membros da equipe de desenvolvimento de software, por meio de uma apresentação dos objetivos do *roadmap* a tirando dúvidas, à medida que estas surgiam na reunião.

Durante esta reunião, de duração de 1 hora e meia, buscou-se por agendar uma reunião com a equipe de desenvolvimento em um horário específico de uma hora para a realização da próxima atividade, a identificação do cenário atual da equipe de desenvolvimento de software.

A atividade de "Identificação do cenário atual de cada equipe de desenvolvimento de software", foi realizada por meio de uma reunião de duração de uma hora, feita com a equipe de desenvolvimento de software e sua liderança técnica, contando com a participação do coordenador da equipe nesta primeira reunião. Esta atividade foi importante para que cada equipe de desenvolvimento, juntamente com o aplicador da *roadmap*, pudessem analisar e montar o cenário atual da equipe por meio do preenchimento mútuo do formulário de práticas de teste de software aplicadas na equipe de desenvolvimento, a qual pode ser observada na Tabela 4.

Tabela 4: Cenário Inicial – Organização Alfa

Prática em Teste de Software	Equipe 01 - ANTES
Desenvolvimento orientado a testes	Nunca
Testes unitários	Nunca
Testes unitários utilizam mocks	Nunca
São aplicados testes de mutação	Nunca
Testes unitários são necessários para conclusão da história	Nunca
Análise estática de código	Sempre
Testes regressivos	Nunca
Há um ambiente específico para execução dos testes	Sempre
O status de retorno das aplicações são validados	Nunca
Os campos no banco de dados são validados	Nunca
Testes funcionais automatizados	Nunca
Testes automatizados estão integrados na esteira CI/CD	Sempre
Erro nos testes acarreta falha na esteira CI/CD	Sempre
Ocorre desativação de testes(bypass) na esteira CI/CD	Nunca
Testes funcionais são necessários para conclusão da história	Nunca
Testes de integração com banco de dados ou outras aplicações	Nunca
Testes de performance	Nunca
Testes de stress	Nunca
Testes não funcionais são necessários para conclusão da história	Nunca
Testes exploratórios	Nunca
São escritos cenários/planos de teste	Sempre
Há padrões na escrita dos cenários/planos de teste	Nunca
Revisão por pares dos cenários/plano de teste	Nunca
São evidenciadas as execuções de teste	Nunca
Ocorre a documentação dos defeitos encontrados	Nunca
Resultados dos testes são utilizados para melhoria do produto	Frequentemente
São estimadas horas para planejamento/execução dos testes	Sempre
Os testes são baseados em requisitos de negócios	Frequentemente
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkins)	Sempre
Execução dos testes são realizados com frequência	Raramente
São planejados quantidade de testes por nível da pirâmide de testes	Nunca
Aonde se encontra a maior quantidade de testes da equipe?	Testes unitários

Fonte: O Autor

Como visto na Tabela 4, foi feito o levantamento em conjunto com a equipe de desenvolvimento de software sobre as práticas de teste aplicadas no dia a dia e, juntamente com as práticas, colheu-se alguns detalhes técnicos das aplicações desenvolvidas e peculiaridades da equipe, para que, com base nessas informações, seja possível o desenvolvimento do plano de ação específico para cada o grupo de trabalho.

Nesta etapa, além da avaliação das práticas de teste de software, foi também avaliado o nível de maturidade em testes de software que a equipe de desenvolvimento apresentou, com base no TMMi, permitindo o avanço do *roadmap* para a etapa de averiguações. Esta análise sobre o nível de maturidade da equipe de desenvolvimento teve como resultado o nível 2 de maturidade, ou seja, nível Gerenciado em maturidade de teste de software, visto que há uma estratégia e planejamento de testes, monitoramento, ambientes de execução e execução de testes.

5.3. Etapa Averiguações

A etapa de averiguações teve duração de 2 horas, subdivididos em análise e estruturação dos dados de cenário atual, construção do plano de ação específico para a equipe de desenvolvimento e, uma reunião com liderança para apresentação e validação dos planos de ação.

A atividade de “Análise e organização dos dados do cenário atual da(s) equipe(s) de desenvolvimento de software” é composta pela análise estruturada dos dados levantados com a equipe de desenvolvimento de software, dados estes coletados em conjunto com a equipe sobre as práticas de teste aplicadas atualmente e o como são aplicadas, além do levantamento do nível de maturidade em teste de software da equipe de desenvolvimento. Esta atividade, por ser realizada após a reunião de levantamentos, teve um tempo médio de 30 minutos, seguido da atividade de “Construção do plano de melhoria para cada equipe de desenvolvimento de software”, etapa esta que durou uma hora e buscou identificar quais as ações são necessárias tanto para melhorar o nível de maturidade da equipe de desenvolvimento como também, de estruturar os testes dentro do nível de maturidade que as equipes se encontravam. Este nível de maturidade foi analisado com base no modelo de maturidade de teste de software TMMi. Com duração média de uma hora, esta atividade contou com a definição das atividades necessárias para melhoria de teste de software da equipe de desenvolvimento, a imagem referente ao plano de ação poderá ser observada mais a seguir, no capítulo sobre Melhorias.

De maneira geral, o plano de ação teve como principal objetivo, melhorar o nível de maturidade em teste de software da equipe de desenvolvimento.

As três semanas da iteração foram compostas por nove reuniões de uma hora cada, para aplicação das melhorias e teve como planejamento, a aplicação de treinamento sobre pirâmide de testes, documento de cenários de teste e de defeitos, testes unitários, testes unitários com a utilização de *Mocks*, testes de integração e testes regressivos. O planejamento contou também com um treinamento de reforço sobre os testes unitários utilizando *Mocks*, testes funcionais automatizados e, como último assunto abordado no plano de ação, houve a revisão por pares.

Outra atividade presente nesta etapa é a de "Apresentação do plano de ação para os gestores", etapa esta realizada por meio de uma reunião de meia hora, na qual foram apresentados aos líderes técnicos todo o planejamento para melhoria dos testes de software e, alinhamento dos resultados esperados com a execução de cada atividade.

5.4. Etapa Melhorias

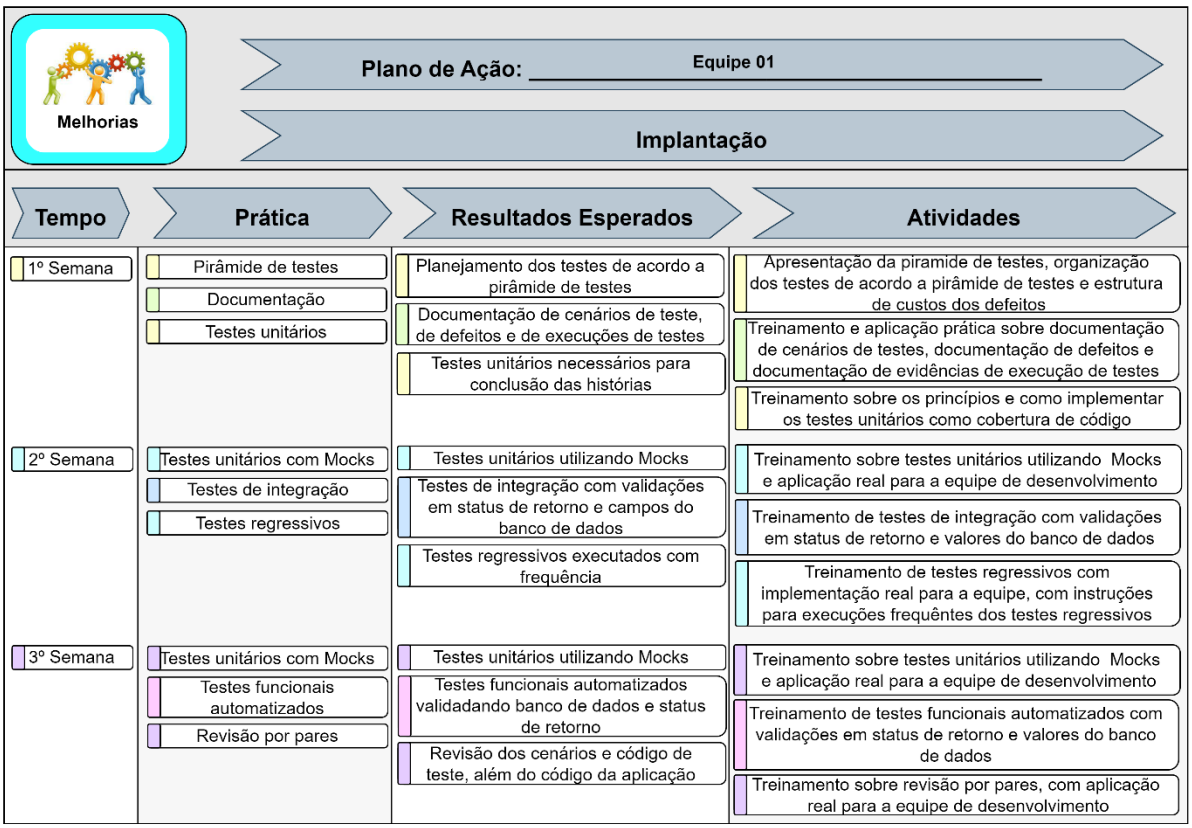
A etapa de melhorias teve duração de 1 hora e uma iteração da equipe de desenvolvimento de software, subdivididos em apresentação do plano de ação para a equipe de desenvolvimento de software e sua execução. A primeira atividade desta etapa é a "apresentação do plano de cação para cada equipe de desenvolvimento de software", atividade esta que foi realizada por meio de uma reunião objetiva com cada equipe de desenvolvimento de software para apresentar os caminhos traçados para melhoria do teste de software. De modo geral, esta reunião deve duração de 1 hora e, de maneira geral, percebeu-se empolgação muito forte da equipe de desenvolvimento como um todo. Para se ter um maior engajamento das equipes de desenvolvimento de software, foi solicitado que, em todas as reuniões relacionadas a esta etapa de Melhorias, todos os integrantes utilizam suas câmeras abertas.

Como alinhado previamente com os gestores, foi definido um tempo de 3 horas semanais para atuação nos planos de ações de melhoria dos testes de software com a equipe de desenvolvimento, fator que foi levado em consideração da criação do plano de ação para a equipe, de forma que, ao final das três semanas, tempo de duração da iteração, pudesse haver melhorias efetivas do processo de teste de software de cada equipe de desenvolvimento.

A segunda atividade desta etapa, a "aplicação dos planos de ação pelos grupos de trabalho" durou o tempo de uma iteração, com três reuniões semanais de uma hora de duração.

O plano de ação para a equipe central de desenvolvimento de software pode ser observado por meio da Figura 14.

Figura 14: Plano de ação – Equipe 01 – Organização Alfa



Fonte: O Autor

Como observado na Figura 14, o plano de ação é composto por atividades realizadas ao longo de três semanas de trabalho com forte participação da equipe de desenvolvimento de software.

Durante a primeira semana, as atividades se concentraram nas práticas de pirâmide de testes, documentação e testes unitários. Na primeira reunião da semana, a equipe recebeu um treinamento sobre como organizar e planejar os testes de acordo com a pirâmide de testes, que consiste em automatizar a maioria dos testes na base e manter uma quantidade menor de testes no topo da pirâmide. O objetivo dessa atividade era que a equipe de desenvolvimento pudesse realizar o planejamento dos testes de acordo com a pirâmide, a fim de garantir uma cobertura de testes eficiente.

Na segunda reunião da semana, foi aplicado um treinamento sobre padronização de documentação de cenários de teste e de relato de defeitos. O objetivo desta atividade foi

capacitar a equipe de desenvolvimento para documentar de forma padronizada seus cenários de teste e relatar os defeitos encontrados nas aplicações desenvolvidas. Espera-se que, como resultado desta atividade, a equipe consiga registrar de forma clara e organizada os cenários de teste e defeitos, facilitando a comunicação entre os membros da equipe e agilizando a resolução de problemas.

Durante a terceira reunião da semana, foi realizado um treinamento sobre testes unitários, abordando os princípios fundamentais e a importância da cobertura de código-fonte. O objetivo desta atividade foi capacitar a equipe de desenvolvimento a implementar testes unitários de forma eficiente, considerando-os como uma etapa fundamental para a conclusão de uma história. O resultado esperado é que a equipe consiga criar testes unitários que cubram adequadamente o código das aplicações desenvolvidas.

Durante a primeira reunião da segunda semana, foi realizado um treinamento que abordou a importância dos testes unitários e a utilização de *Mocks* para isolar o código e realizar testes nas menores unidades do sistema. O objetivo dessa atividade era que a equipe de desenvolvimento pudesse compreender a importância dos testes unitários e ser capaz de implementá-los eficientemente, usando *Mocks* quando necessário para isolar o código e facilitar a realização de testes nas menores unidades das aplicações. Além disso, o treinamento também destacou a relevância da cobertura de código-fonte pelos testes unitários, fornecendo à equipe uma visão clara sobre como garantir a qualidade do código e a estabilidade das aplicações desenvolvidas.

Na segunda reunião da semana, foi ministrado um treinamento sobre testes de integração e validações, tanto no retorno da aplicação quanto nos valores alterados nas bases de dados. O objetivo dessa atividade foi capacitar a equipe de desenvolvimento a criar testes de integração capazes de validar o retorno das aplicações, bem como as integrações com outras aplicações e com o banco de dados. Espera-se que, após o treinamento, a equipe consiga executar esses testes na esteira de integração/entrega contínua.

Na terceira reunião da semana, foi abordado o tema de testes regressivos e a sua importância na garantia da qualidade do software. A equipe de desenvolvimento recebeu um treinamento sobre como executar os testes regressivos de forma frequente e integrada à esteira de integração contínua, garantindo que os testes sejam realizados a cada nova alteração no código-fonte. O objetivo da atividade foi assegurar que a equipe esteja ciente da importância dos testes regressivos e capaz de executá-los de forma eficiente para detectar eventuais regressões no sistema.

Na primeira reunião da terceira e última semana de implementação do plano de ações, foi realizado um treinamento sobre os testes unitários utilizando *Mocks*. O objetivo foi reforçar o conhecimento sobre a importância dos testes unitários e como utilizar os *Mocks* para isolar o código e testar as menores unidades do sistema. Além disso, a equipe recebeu suporte na criação de testes unitários para cenários mais complexos. O resultado esperado desta atividade é que a equipe de desenvolvimento consiga implementar os testes unitários com sucesso, utilizando *Mocks* sempre que necessário, e garantindo a qualidade do código em todas as etapas do desenvolvimento.

Na segunda reunião da semana, foi realizado um treinamento sobre testes funcionais automatizados. O objetivo deste treinamento foi capacitar a equipe de desenvolvimento para a criação de testes funcionais automatizados, que possuam múltiplos pontos de validação e possam ser executados de forma integrada na esteira de integração e entrega contínua. O resultado esperado é que a equipe de desenvolvimento seja capaz de criar testes funcionais robustos, que garantam a qualidade das aplicações desenvolvidas e possam ser executados com frequência de forma automatizada.

Na terceira e última reunião da semana, foi ministrado um treinamento sobre a importância e como realizar a revisão por pares, tanto do código-fonte da aplicação quanto do código de teste e dos cenários de teste. O objetivo desta atividade foi conscientizar a equipe sobre a relevância da revisão por pares para melhorar a qualidade do código e dos testes, além de fornecer orientações sobre como realizar essas revisões de forma eficiente. Espera-se que a equipe de desenvolvimento possa realizar as revisões por pares com frequência e de forma sistemática, contribuindo para a identificação e correção de erros antes que eles se tornem problemas mais graves.

5.5. Etapa Aferições

A etapa de aferições teve duração de 3 horas e 30 minutos, subdivididos em reuniões com a equipe de desenvolvimento para, em conjunto, realizar o mapeamento do cenário alcançado, utilizado para comparativo de antes e depois e análise do cenário de teste de software. Esta etapa contou também com a análise dos dados e resultados alcançados e, com uma etapa muito importante de comunicação dos resultados, que contou com reuniões entre a equipe de desenvolvimento e o gestor responsável para divulgação de resultados de melhoria do processo de teste de software.

A primeira etapa desta tarefa consistiu na "Identificação do cenário alcançado após a execução do plano de ação". Foi realizada uma reunião com a equipe de desenvolvimento de software, com duração de uma hora, para analisar em conjunto as práticas de teste de software e sua frequência de utilização. Durante a reunião, a participação dos membros da equipe de desenvolvimento foi bastante expressiva, e houve um feedback muito positivo sobre a melhoria dos testes de software. De forma geral, os participantes destacaram o quanto aprenderam e aplicaram as melhorias propostas durante o plano de ação.

A segunda atividade desta etapa consistiu na análise dos resultados obtidos após a execução do plano de ação. Foi realizado um levantamento dos dados antes e depois da implementação das melhorias nos testes de software, permitindo uma comparação dos cenários. Além disso, foi elaborado um material de apresentação para os gestores e líderes, a fim de demonstrar a evolução da equipe de desenvolvimento de software em relação à maturidade e à melhoria dos testes de software.

Como resultado da análise do cenário alcançado versus o cenário anterior, pode-se observar a Tabela 5.

Tabela 5: Cenário Alcançado – Organização Alfa

Prática em Teste de Software	Equipe 01 - ANTES	Equipe 01 - Depois
Desenvolvimento orientado a testes	Nunca	Nunca
Testes unitários	Nunca	Frequentemente
Testes unitários utilizam mocks	Nunca	Frequentemente
São aplicados testes de mutação	Nunca	Nunca
Testes unitários são necessários para conclusão da história	Nunca	Sempre
Análise estática de código	Sempre	Sempre
Testes regressivos	Nunca	Frequentemente
Há um ambiente específico para execução dos testes	Sempre	Sempre
O status de retorno das aplicações são validados	Nunca	Sempre
Os campos no banco de dados são validados	Nunca	Sempre
Testes funcionais automatizados	Nunca	Frequentemente
Testes automatizados estão integrados na esteira CI/CD	Sempre	Sempre
Erro nos testes acarreta falha na esteira CI/CD	Sempre	Sempre
Ocorre desativação de testes(bypass) na esteira CI/CD	Nunca	Nunca
Testes funcionais são necessários para conclusão da história	Nunca	Frequentemente
Testes de integração com banco de dados ou outras aplicações	Nunca	Frequentemente
Testes de performance	Nunca	Nunca
Testes de stress	Nunca	Nunca
Testes não funcionais são necessários para conclusão da história	Nunca	Nunca
Testes exploratórios	Nunca	Raramente
São escritos cenários/planos de teste	Sempre	Sempre
Há padrões na escrita dos cenários/planos de teste	Nunca	Sempre
Revisão por pares dos cenários/plano de teste	Nunca	Sempre
São evidenciadas as execuções de teste	Nunca	Sempre
Ocorre a documentação dos defeitos encontrados	Nunca	Sempre
Resultados dos testes são utilizados para melhoria do produto	Frequentemente	Frequentemente
São estimadas horas para planejamento/execução dos testes	Sempre	Sempre
Os testes são baseados em requisitos de negócios	Frequentemente	Sempre
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkins)	Sempre	Sempre
Execução dos testes são realizados com frequência	Raramente	Frequentemente
São planejados quantidade de testes por nível da pirâmide de testes	Nunca	Frequentemente
Aonde se encontra a maior quantidade de testes da equipe?	Testes unitários	Testes unitários

Fonte: O Autor

Observando a Tabela 5, podemos notar melhorias significativas na frequência e na maturidade das práticas de teste de software da equipe de desenvolvimento. Foi possível observar uma melhora na utilização de testes unitários, com a utilização de *Mocks* para isolar o código e torná-lo testável, além da inclusão desses testes como obrigatórios para a conclusão de histórias. Também houve uma melhoria nos testes regressivos e nas validações dos testes de integração, tanto no status de retorno das aplicações quanto nos campos das bases de dados. A inclusão de testes funcionais automatizados e sua execução frequente, juntamente com testes de integração com o banco de dados e outras aplicações, foi outra prática adotada. A equipe agora também é capaz de realizar testes exploratórios e utiliza os resultados dos testes como uma forma de melhorar seus produtos. Em resumo, a equipe de desenvolvimento evoluiu para executar testes com mais frequência e planejar testes baseados na pirâmide de testes, com mais testes unitários na base e menos testes em níveis superiores da pirâmide.

A equipe também passou a incluir testes funcionais automatizados como parte do processo de conclusão das histórias e a realizar com mais frequência testes de integração com o banco de dados e outras aplicações. Além disso, a equipe agora consegue realizar testes exploratórios e utilizar os resultados dos testes para melhorar os produtos.

De forma geral, a equipe evoluiu no sentido de realizar testes com mais frequência e planejar a quantidade de testes com base na pirâmide de testes, com uma maior quantidade de testes unitários na base e quantidades menores de testes à medida que se sobe na pirâmide.

Em relação ao nível de maturidade em teste de software com base no TMMi, a equipe conseguiu avançar em seus processos para melhorar algumas etapas do nível 2-Gerenciado e iniciar atividades relativas ao nível 3-Definido. Entre essas atividades, incluem-se a utilização de *mocks* nos testes unitários, um maior foco na criação de testes de integração e testes funcionais automatizados, além da prática de revisão por pares.

Para a análise desta equipe de desenvolvimento, foi dedicado um total de uma hora e meia para preparação do material de análise comparativo, documento este, usado como base para apresentação dos resultados aos gestores e liderança.

A última atividade da etapa, a "Comunicação dos resultados obtidos" se deu por meio de uma reunião de duração de uma hora para apresentação dos resultados obtidos pela equipe de desenvolvimento em maturidade e melhoria em teste de software.

5.6. Entrevista

A entrevista foi feita após a implementação do RoMTeS e teve como principal objetivo analisar a opinião dos gestores e liderança envolvida na aplicação do *roadmap* de melhoria de teste de software. Esta entrevista foi relativamente curta, realizada por meio de reunião com a liderança, contanto com sete questões chave para a avaliação do produto.

A entrevista foi feita juntamente com o líder técnico da equipe de desenvolvimento, o Product Owner e, com o coordenador técnico.

A primeira pergunta realizada foi sobre a percepção da liderança em relação ao aumento do nível de maturidade em teste de software após a implementação do RoMTeS. A resposta da liderança foi positiva, indicando que já percebeu melhorias significativas e que espera que essa percepção aumente com o tempo, à medida que as práticas aprendidas sejam estendidas aos

demais micro serviços da equipe. De forma geral, a liderança percebeu valor na melhoria das práticas de teste de software.

A segunda pergunta realizada foi sobre a percepção da liderança em relação à usabilidade do RoMTeS. Ou seja, se o *roadmap* é fácil ou difícil de ser aplicado. A resposta da liderança foi que a aplicação do RoMTeS é considerada fácil, mas é importante destacar que depende muito de ter uma pessoa capacitada em práticas de teste de software para liderar os treinamentos e a aplicação dos planos de ação na equipe de desenvolvimento. Ter um líder técnico experiente e dedicado ao sucesso da implementação do *roadmap* é essencial para garantir que as práticas de teste de software sejam aplicadas adequadamente. Além disso, a liderança destacou que a documentação e os materiais disponíveis no RoMTeS foram muito úteis na compreensão do processo e na aplicação das práticas recomendadas.

A terceira pergunta feita durante a entrevista foi sobre a percepção da liderança em relação à viabilidade do RoMTeS. A liderança respondeu que, de fato, considera o *roadmap* viável e que os resultados obtidos pela equipe de desenvolvimento em relação à qualidade das entregas são prova disso. O líder técnico da equipe enfatizou a importância da aplicação do *roadmap* como uma forma de melhorar continuamente os processos de teste de software, destacando que o RoMTeS é uma ferramenta útil nesse sentido.

A quarta pergunta abordou a percepção da liderança sobre a utilidade do RoMTeS em organizações que trabalham com desenvolvimento ágil de software, incluindo aquelas que utilizam a arquitetura de micro serviços. De forma geral, a liderança afirmou que percebeu que o *roadmap* pode ser aplicado não apenas em organizações com arquiteturas de micro serviços, mas também em outras arquiteturas e até mesmo em equipes que trabalham com o desenvolvimento de aplicativos móveis ou outras tecnologias. Isso demonstra que o RoMTeS é uma ferramenta versátil e pode ser adaptado a diferentes contextos de desenvolvimento de software.

A quinta pergunta realizada foi sobre a possibilidade de comercialização do RoMTeS, e toda a liderança concordou que estaria aberta a discutir valores e serviços de consultoria, desde que o *roadmap* de melhoria de teste de software fosse acompanhado por um serviço de consultoria para sua aplicação, especialmente em relação às demonstrações presentes no plano de ação.

Na sexta pergunta, a liderança foi questionada sobre sua experiência com a implantação do RoMTeS. A resposta foi muito positiva, com a liderança relatando que a experiência foi muito boa e que a equipe de desenvolvimento também percebeu o valor da implantação do *roadmap* de melhoria de teste de software, fornecendo feedbacks positivos.

Como última pergunta, questionou-se à liderança se ela recomendaria a aplicação do *Roadmap* de Melhoria de Teste de Software para outras organizações e equipes que atuam no ambiente de desenvolvimento de software. A resposta unânime da liderança foi que sim, recomendariam a aplicação do RoMTeS devido aos benefícios obtidos na qualidade das entregas.

Com base na entrevista realizada com a liderança envolvida na aplicação do *Roadmap* de Melhoria de Teste de Software, pode-se concluir que a implantação do RoMTeS foi percebida como algo extremamente valioso e positivo para a equipe de desenvolvimento, já que contribuiu significativamente para o aumento da qualidade das entregas realizadas. Além disso, a liderança concordou que o RoMTeS é um produto viável e útil para outras organizações e equipes que atuam com o desenvolvimento de software. A comercialização do *roadmap* também foi discutida e a liderança se mostrou aberta à conversas sobre consultoria para sua aplicação. Em resumo, a aplicação do RoMTeS foi considerada bem-sucedida e recomendada pela liderança entrevistada.

5.7. Considerações parciais

A implementação bem-sucedida do *roadmap* RoMTeS nesta organização de pequeno porte mostra que o investimento em práticas de melhoria de testes de software pode trazer resultados positivos em termos de qualidade das entregas e satisfação da equipe. A equipe de desenvolvimento elogiou a rapidez em encontrar defeitos e a aplicação prática das práticas de teste de software, e considerou o tempo dedicado à implementação das melhorias adequado. Isso sugere que outras organizações de desenvolvimento de software podem se beneficiar da aplicação do RoMTeS e de outras práticas de melhoria de teste de software. É importante lembrar que a implementação dessas práticas pode exigir o apoio da liderança e o envolvimento da equipe de desenvolvimento para obter os melhores resultados.

A liderança da equipe, composta pelo líder técnico, Product Owner, líder de equipe e coordenador técnico, ficou satisfeita com os resultados alcançados após a implementação do *roadmap* RoMTeS. Eles puderam constatar as melhorias significativas na qualidade das entregas de desenvolvimento de software, graças ao aumento do nível de maturidade da equipe de desenvolvimento para o nível 3 - Definido, com base no modelo de maturidade em teste de software TMMi.

Todos os membros da liderança puderam perceber o valor em dedicar um tempo adequado às melhorias propostas pelo *roadmap*, assim como a importância de treinamentos e métodos de aplicação prática das práticas de teste de software. Além disso, a liderança destacou o feedback positivo da equipe de desenvolvimento sobre a rapidez em encontrar defeitos antes de chegarem ao ambiente produtivo.

Com base nos resultados obtidos, a implementação do RoMTeS se mostrou empolgante e desafiadora, mas os benefícios alcançados em relação à qualidade das entregas foram evidentes.

Com base na entrevista realizada com a liderança da equipe de desenvolvimento de software após a implementação do *Roadmap* de Melhoria de Teste de Software (RoMTeS), foi possível concluir que a aplicação do *roadmap* foi percebida como um produto viável e útil pela liderança. Eles destacaram que a aplicação do RoMTeS é fácil, desde que uma pessoa com conhecimento em práticas de teste de software lidere os treinamentos e aplicações dos planos de ação para capacitação da equipe de desenvolvimento.

Além disso, a liderança destacou que o RoMTeS pode ser aplicado em outras organizações e equipes que atuam com desenvolvimento de software, independentemente da arquitetura utilizada, pois acreditam que a aplicação do roadmap pode trazer benefícios em relação à qualidade das entregas realizadas.

Em resumo, a entrevista evidenciou a satisfação da liderança com a implantação do RoMTeS e a possibilidade de sua aplicação em outras organizações. A utilização de ferramentas como essa pode contribuir significativamente para o desenvolvimento de softwares de qualidade e atender às necessidades dos clientes de forma mais eficiente.

6. IMPLANTAÇÃO DO ROMTES EM EMPRESA DE MÉDIO PORTE

A implementação do *roadmap* RoMTeS na Organização Beta mostrou-se como um grande desafio, principalmente devido à baixa maturidade em relação aos testes de software na organização e sua estrutura. Como empresa de médio porte que atua no setor de outsourcing de impressão, a organização conta com sua própria equipe de desenvolvimento de software e uma equipe dedicada à qualidade e testes das aplicações desenvolvidas. Apesar das dificuldades encontradas, a equipe responsável pela implementação do RoMTeS permaneceu motivada em busca da melhoria contínua da qualidade do software entregue pela organização.

Por questões de confidencialidade e privacidade de informações, neste relato serão apresentadas somente informações estritamente relacionadas ao *roadmap* RoMTeS implantado na Organização Beta, uma empresa de médio porte que atua no setor de outsourcing de impressão. As conversas iniciais sobre a implantação do *roadmap* foram realizadas com gestores e coordenadores da organização, os quais demonstraram grande entusiasmo em relação à implementação do RoMTeS.

Antes de descrever a implantação do *roadmap* de melhoria de teste de software na Organização Beta, é importante mencionar algumas características relevantes. A primeira delas é que, embora a empresa adote o modelo ágil de desenvolvimento de software, conta com uma equipe específica para atuar em testes e qualidade de software, a qual foi selecionada para a implementação do RoMTeS.

A Organização Beta é uma empresa que atua no setor de outsourcing de impressão e desenvolve suas próprias aplicações Web, utilizando arquitetura de micro serviços. Apesar de trabalhar no modelo ágil, a organização possui uma equipe específica para testes e qualidade de software, composta por 5 profissionais, liderados por um gestor e um coordenador. A equipe trabalha presencialmente, o que se mostrou um desafio para a implementação do RoMTeS, o *roadmap* de melhoria de teste de software.

Para gerenciar os projetos, a equipe utiliza o Kanban e estabeleceu um período de três semanas para aplicação das melhorias em teste de software. A Organização Beta não adota a esteira de integração contínua e entrega contínua e, como trabalha com Kanban, não realiza planejamento de sprints, mas sim de backlogs.

Um fator interessante sobre essa organização é que, apesar de ter uma equipe responsável pelos testes das aplicações, os membros desta equipe realizam apenas testes funcionais e não atuam com automação de processos/testes. Como métricas analisadas, esta organização prioriza a quantidade de testes manuais realizados por entrega da equipe de desenvolvimento de software, sem métricas para testes unitários, testes funcionais automatizados e testes de desempenho (*performance*).

A aplicação do *roadmap* aconteceu durante o mês de janeiro de 2023, ocupando um total de três semanas para aplicação das melhorias em parceria com a equipe de qualidade e testes de software.

As etapas de implantação foram seguidas e, o tempo total de implantação do *roadmap* RoMTeS nesta organização durou um total de 12 horas e três semanas. Os resultados obtidos com a implementação do *roadmap* RoMTeS nesta organização estruturou os processos e, incluiu novas métricas para que a organização análise em quesitos de testes automatizados.

Com base na estrutura da organização, elaborou-se um cronograma semanal para aplicação do *roadmap*, com margem para que houvesse adequações de horários, se necessário. O cronograma da Semana 1 pode ser visto por meio da Figura 15.

Figura 15: Cronograma Empresa Beta – Semana 01

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00					
11:00	Esclarescimentos	Apresentação do	Equipe1 - Avaliação		
12:00	com os Gestores	RoMTeS a equipe de	do cenário atual		
13:00					
14:00					
15:00	Esclarecimentos		Equipe1 -		
16:00	com Liderança		Elaboração do plano		
17:00					
18:00					

Fonte: O Autor

Com base na Figura 15, pode-se observar a primeira semana de implantação do *roadmap* RoMTeS como uma semana de esclarecimentos, definições e avaliação da equipe de testes de software, além de elaboração do plano de ação e apresentação do plano de ação aos gestores e liderança da equipe. O cronograma da Semana 02, Semana03 e Semana 04 são idênticos, como pode ser visto por meio da Figura 16.

Figura 16: Cronograma Empresa Beta – Semana 02, 03 e 04

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00					
11:00	Equipe1 -		Equipe1 -		Equipe1 -
12:00	Encontro 01		Encontro 02		Encontro 03
13:00					
14:00					
15:00					
16:00					
17:00					
18:00					

Fonte: O Autor

Com base na Figura 16, pode-se observar que a segunda, terceira e quarta semana são iguais em termos de calendário. Isto se dá porque, de acordo o planejamento do *roadmap*, este tempo é destinado para aplicação do plano de plano de ação em parceria com a equipe de testes

de software. Já o cronograma da última semana de implantação do *roadmap*, a Semana 05 pode ser visto por meio da Figura 17.

Figura 17: Cronograma Empresa Beta – Semana 05

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00					
11:00	Equipe1 - Reavaliação	Apresentação dos resultados gerais			
12:00					
13:00					
14:00					
15:00	Organização dos dados				
16:00					
17:00					
18:00					

Fonte: O Autor

Com base na Figura 17, pode-se observar a última semana de implantação do *roadmap*, destinada a reavaliação da equipe de desenvolvimento de software, organização e análise dos dados de cenário alcançado e apresentação dos resultados obtidos aos gestores e liderança. A seguir teremos a descrição de implantação do RoMTeS, a se iniciar pela etapa de esclarecimentos, sua primeira etapa.

6.1. Etapa Esclarecimentos

A etapa de esclarecimentos teve a duração de duas horas em uma reunião com a liderança da equipe, incluindo o coordenador, gestor e líderes técnicos da organização, além dos líderes das equipes de desenvolvimento de software e de testes de software

A atividade, denominada "Esclarecimento do RoMTeS aos gestores", foi conduzida por meio de uma reunião de uma hora de duração, na qual foi apresentada a estrutura do *roadmap*, incluindo cada etapa e o tempo médio definido para cada atividade. À medida que surgiram dúvidas, elas foram tratadas na reunião, garantindo que, ao final, todos os participantes tivessem entendido completamente o *roadmap* de melhoria de teste de software.

A segunda atividade da etapa de esclarecimentos foi o "Esclarecimento da Abrangência do RoMTeS", que consistiu em uma reunião adicional com os gestores e coordenador para definir quais equipes seriam abrangidas pela implementação do *roadmap* de melhoria de teste de software, bem como o tempo semanal que seria investido pela equipe na aplicação do

RoMTeS. Nessa reunião de uma hora e meia, ficou acordado que a equipe central de testes de software seria a escolhida para aplicar o RoMTeS, com um investimento de três horas semanais para reuniões e encontros de melhoria de processos. Durante o período de três semanas, a equipe se concentraria nos tópicos abordados nas reuniões para aprimorar o processo de teste da organização. Além disso, o gestor solicitou que a ferramenta AzureDevOps, já adquirida pela organização, fosse utilizada para incluir uma ferramenta de integração contínua/entrega contínua.

A seleção da equipe de testes de software foi acompanhada pela coleta dos nomes e contatos de cada membro, levantados durante a reunião com a liderança. Após a reunião, foi enviado um e-mail para todos os participantes com um resumo dos tópicos discutidos e a definição do tempo de atuação e líderes técnicos responsáveis pela equipe selecionada.

Também foi enviado um e-mail aos membros da equipe de testes selecionados, com cópia para seus respectivos gestores e liderança, informando sobre o processo de melhoria de teste de software e agendando uma reunião com eles. Além disso, foi acordado com os gestores que, em determinadas reuniões técnicas, membros das equipes de desenvolvimento de software também estariam presentes para capacitação em novas técnicas e termos da organização.

Finalizada a etapa de esclarecimentos, pode-se seguir para a etapa Definições, mais bem descrita a seguir.

6.2. Etapa Definições

A etapa de definições teve duração de 3 horas e 30 minutos, subdivididos em reuniões que envolveram a liderança e a equipe de testes de software.

Nesta etapa do *roadmap*, foi realizada a atividade de “Apresentação do RoMTeS aos líderes técnicos”, por meio de uma reunião de uma hora com os líderes técnicos da equipe de testes de software. Durante a reunião, foram apresentados detalhes técnicos sobre todo o processo de implementação do *roadmap* de melhoria de teste de software, incluindo os objetivos definidos para o projeto. Além disso, foram abordadas as dúvidas levantadas durante a reunião, com o intuito de esclarecê-las e garantir a compreensão completa do projeto pelos líderes técnicos.

Durante esta reunião de uma hora, enfatizou-se a importância do engajamento dos líderes técnicos e seus respectivos grupos de trabalho para a implantação do *roadmap* de melhoria de teste de software. Foi apresentado detalhadamente os objetivos do *roadmap* e foram

sanadas algumas dúvidas levantadas pelos líderes técnicos. Foi destacado que a implementação do *roadmap* trará benefícios significativos para a organização, como a melhoria do processo de teste de software, capacitação das equipes de testes de software e aumento da qualidade das entregas e aplicações desenvolvidas.

Outra atividade importante desta etapa do *roadmap* é a "Apresentação do RoMTeS às equipes de desenvolvimento de software". Nesta atividade, todos os membros da equipe de testes de software foram envolvidos, por meio de uma apresentação dos objetivos do *roadmap* e esclarecimento de dúvidas que surgiram durante a reunião.

Durante esta reunião, com duração de 1 hora e meia, foi agendada uma próxima reunião de uma hora com a equipe de testes para realizar a atividade de identificação do cenário atual da equipe de desenvolvimento de software. A importância desta atividade é entender como a equipe de desenvolvimento está operando atualmente e quais são seus principais desafios e problemas, a fim de propor soluções mais adequadas e eficazes para a melhoria do processo de teste de software.

A atividade de “Identificação do cenário atual de cada equipe de desenvolvimento de software” foi realizada por meio de uma reunião com a equipe de testes de software e sua liderança técnica, contando com a participação do coordenador da equipe. A reunião teve duração de uma hora e teve como objetivo analisar e montar o cenário atual da equipe por meio do preenchimento mútuo do formulário de práticas de teste de software aplicadas na equipe. Os resultados desta atividade podem ser observados na Tabela 6.

Tabela 6: Cenário Inicial – Organização Beta

Prática em Teste de Software	Equipe 01 - ANTES
Desenvolvimento orientado a testes	Nunca
Testes unitários	Nunca
Testes unitários utilizam mocks	Nunca
São aplicados testes de mutação	Nunca
Testes unitários são necessários para conclusão da história	Nunca
Análise estática de código	Nunca
Testes regressivos	Nunca
Há um ambiente específico para execução dos testes	Sempre
O status de retorno das aplicações são validados	Frequentemente
Os campos no banco de dados são validados	Frequentemente
Testes funcionais automatizados	Nunca
Testes automatizados estão integrados na esteira CI/CD	Nunca
Erro nos testes acarreta falha na esteira CI/CD	Nunca
Ocorre desativação de testes(bypass) na esteira CI/CD	Não sei informar
Testes funcionais são necessários para conclusão da história	Sempre
Testes de integração com banco de dados ou outras aplicações	Nunca
Testes de performance	Nunca
Testes de stress	Nunca
Testes não funcionais são necessários para conclusão da histór	Nunca
Testes exploratórios	Nunca
São escritos cenários/planos de teste	Sempre
Há padrões na escrita dos cenários/planos de teste	Nunca
Revisão por pares dos cenários/plano de teste	Nunca
São evidenciadas as execuções de teste	Sempre
Ocorre a documentação dos defeitos encontrados	Nunca
Resultados dos testes são utilizados para melhoria do produto	Frequentemente
São estimadas horas para planejamento/execução dos testes	Sempre
Os testes são baseados em requisitos de negócios	Frequentemente
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkins)	Nunca
Execução dos testes são realizados com frequência	Raramente
São planejados quantidade de testes por nível da pirâmide de testes	Nunca
aonde se encontra a maior quantidade de testes da equipe?	Testes manuais

Fonte: O Autor

A Tabela 6 fornece um resumo das práticas de teste de software aplicadas pela equipe no dia a dia. Durante a reunião, em conjunto com a equipe de testes de software, foram coletados detalhes técnicos das aplicações desenvolvidas e peculiaridades da equipe e da organização. Essas informações são fundamentais para o desenvolvimento do plano de ação específico que visa a melhoria do processo de teste de software da organização.

Nesta etapa, além da avaliação das práticas de teste de software, também foi realizada uma avaliação do nível de maturidade da equipe em testes de software, com base no TMMi. Essa avaliação permitiu avançar para a próxima etapa do *roadmap*, que é a de averiguações. O resultado dessa análise foi o nível 1 de maturidade, o que indica que a equipe se encontra em um estágio inicial de maturidade em testes de software, uma vez que ainda não executa todas as características do nível 2, possuindo apenas ambientes de execução e planejamento de testes.

6.3. Etapa Averiguações

A etapa de averiguações teve duração de 2 horas, subdivididos em análise e estruturação dos dados de cenário atual, construção do plano de ação específico para a equipe de testes de software e, uma reunião com liderança para apresentação e validação dos planos de ação.

A atividade de "Análise e organização dos dados do cenário atual da(s) equipe(s) de desenvolvimento de software" é essencial para a construção do plano de melhoria do processo de testes de software. Esta atividade envolve a análise dos dados coletados em conjunto com a equipe de testes de software sobre as práticas de teste aplicadas atualmente e o nível de maturidade em testes de software da equipe. Em média, esta atividade leva cerca de 30 minutos.

Já a atividade de "Construção do plano de melhoria para cada equipe de desenvolvimento de software" tem como objetivo identificar as ações necessárias para melhorar o nível de maturidade da equipe de testes com base no modelo TMMi. Com duração média de uma hora, essa atividade conta com a definição das atividades necessárias para melhorar o processo de testes de software da equipe de testes. A imagem referente ao plano de ação pode ser observada no capítulo sobre Melhorias.

A implantação do *roadmap* teve como principal objetivo melhorar o nível de maturidade em teste de software da equipe, buscando estruturar os testes na organização. Para isso, foram realizadas nove reuniões de uma hora cada durante as três semanas da implantação, nas quais foram aplicadas melhorias, como treinamento sobre pirâmide de testes, testes unitários com a utilização de *Mocks*, documentação de cenários de testes e defeitos encontrados, esteira de integração contínua e entrega contínua, testes funcionais automatizados, testes regressivos e treinamento sobre revisão por pares de cenários de teste, testes de código e código-fonte das aplicações. Essas melhorias foram essenciais para aprimorar a qualidade das entregas e aplicações desenvolvidas pela equipe.

A etapa seguinte consistiu na apresentação do plano de ação aos gestores, a fim de alinhar os resultados esperados com a execução de cada atividade. Esta atividade foi conduzida por meio de uma reunião de meia hora com os líderes técnicos, na qual foram apresentados os objetivos, as estratégias e as ações para a melhoria dos testes de software. O propósito dessa atividade foi obter o comprometimento dos gestores em relação às melhorias propostas e garantir a sua participação ativa no processo de mudança.

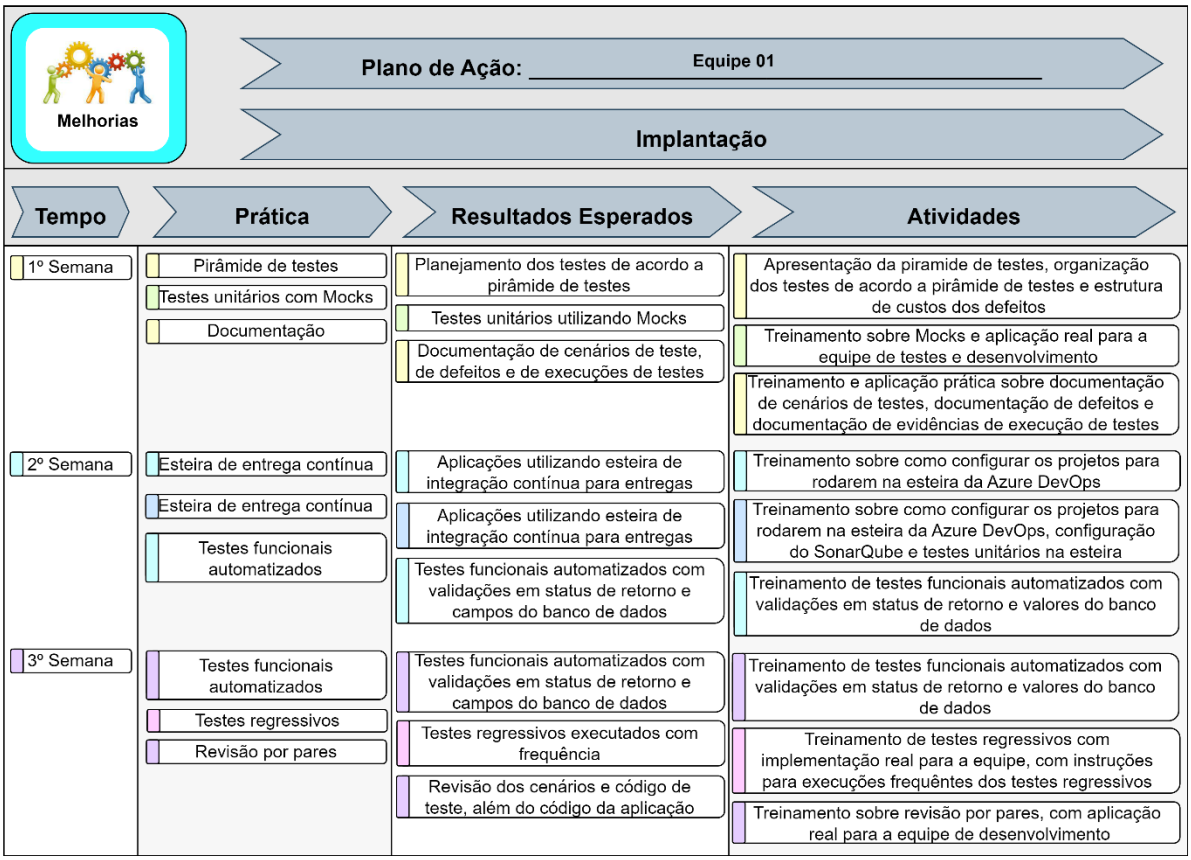
6.4. Etapa Melhorias

A etapa de melhorias teve duração de três semanas, com um total de nove reuniões de uma hora cada, realizadas pela equipe de testes de software. A primeira atividade foi a apresentação do plano de ação para cada equipe de desenvolvimento de software, realizada em uma reunião objetiva com a equipe de testes. Durante a apresentação, notou-se que a equipe de desenvolvimento ficou empolgada com as melhorias propostas, especialmente com a parte de automação do processo de entrega contínua (*deploy*), que até então era realizado manualmente pela equipe de testes de software. Apesar do engajamento da equipe de testes, foi observado que, durante as três reuniões em que os desenvolvedores também participaram, a pedido da gerência, eles não reconheciam a importância dos testes e não demonstravam tanto interesse no processo de melhoria.

Conforme acordado com os gestores, foi estabelecido um período de 3 horas por semana para a equipe de testes trabalhar nos planos de ação para a melhoria dos testes de software. Esse tempo foi considerado na criação do plano de ação, de modo que, ao final das três semanas, tempo previsto para a implementação do plano, houvesse melhorias significativas no processo de teste de software da equipe.

A segunda atividade desta etapa, a “aplicação dos planos de ação pelos grupos de trabalho” durou o tempo das três semanas, com três reuniões semanais de uma hora de duração. O plano de ação para a equipe de testes de software pode ser observado por meio da Figura 18.

Figura 18: Plano de ação – Equipe 01 – Organização Beta



Fonte: O Autor

Como ilustrado na Figura 18, o plano de ação foi dividido em atividades a serem realizadas ao longo de três semanas de trabalho com a participação ativa da equipe de testes de software. Em três dessas reuniões, a gerência solicitou a presença das equipes de desenvolvimento de software.

A primeira semana teve como foco aprimorar as práticas relacionadas à pirâmide de testes, testes unitários com *mocks* e documentação de defeitos e cenários de teste. Na primeira reunião, foi ministrado um treinamento sobre como organizar e planejar os testes de forma que a maioria dos testes sejam automatizados e fiquem na base da pirâmide, e apenas uma pequena quantidade esteja no topo. Essa reunião foi estendida aos desenvolvedores, e para nossa surpresa, metade deles se mostrou interessada e compareceu. O objetivo dessa atividade é garantir que a equipe possa planejar os testes de acordo com a pirâmide, priorizando a automação e otimizando o processo de testes.

Na segunda reunião da semana, a equipe de testes recebeu um treinamento sobre testes unitários utilizando *mocks*. Durante a reunião, foram apresentados os princípios dos testes unitários e como implementá-los de forma a garantir a cobertura do código-fonte e utilizar

mocks quando necessário, para isolar o código e testar cada unidade do código-fonte. O objetivo desta atividade é que a equipe de desenvolvimento seja capaz de criar testes unitários de forma eficiente e considere esta etapa fundamental para a conclusão de uma história. A reunião também foi aberta aos desenvolvedores interessados, e novamente metade dos desenvolvedores participaram da reunião.

Na terceira reunião da semana, foi realizado um treinamento sobre a padronização da documentação de cenários de testes e de defeitos. O objetivo desta atividade é que a equipe de testes seja capaz de documentar seus cenários de teste de maneira clara e concisa, bem como documentar os defeitos encontrados nas aplicações desenvolvidas. Durante a reunião, a equipe de testes demonstrou entusiasmo e participação ativa, esclarecendo dúvidas e contribuindo com sugestões para a melhoria do processo de documentação.

A segunda semana de atividades abrangeu a esteira de integração e entrega contínua, bem como os testes funcionais automatizados. Na primeira reunião, foi aplicado um treinamento sobre como configurar a esteira de CI/CD utilizando a cloud pública Azure DevOps, que já estava sendo analisada pela organização. Na segunda reunião, optou-se por continuar com o tema de integração contínua e entrega contínua para auxiliar com dúvidas e projetos que ainda não haviam sido incluídos na esteira de integração, além de incluir o auxílio para configuração de ferramentas de análise estática de código na esteira, como o SonarQube, e integração dos testes automatizados na esteira. O resultado esperado para esta atividade é que a equipe consiga utilizar a ferramenta Azure DevOps como ferramenta de integração e entrega contínua, com base nos exemplos realizados durante a reunião.

A terceira reunião da semana foi focada em testes funcionais automatizados. Foi aplicado um treinamento prático utilizando aplicações da organização para a automação de fluxos funcionais. Além disso, os testes funcionais automatizados foram incluídos na esteira de integração e entrega contínua. O objetivo desta atividade é que a equipe possa realizar testes funcionais automatizados nas aplicações de forma eficiente e integrá-los com sucesso na esteira de CI/CD.

A terceira e última semana para a implementação do plano de ações teve atividades relacionadas aos testes automatizados funcionais, incluindo validações de resultados de retorno das aplicações, testes regressivos e revisão por pares. Na primeira reunião da semana, a equipe recebeu um treinamento prático sobre testes funcionais automatizados, incluindo a utilização de pontos de validação no banco de dados e no resultado de retorno das aplicações. A reunião também foi dedicada a esclarecer dúvidas sobre cenários que a equipe não tenha conseguido automatizar. Para esta atividade, o resultado esperado é que a equipe consiga realizar testes

funcionais automatizados das aplicações, utilizando pontos de validação como o status de retorno das aplicações e valores do banco de dados.

Na segunda reunião da semana, foi realizado um treinamento sobre testes regressivos e a importância de se realizar testes, especialmente automatizados, com frequência. O objetivo foi enfatizar a importância de garantir que a aplicação continue funcionando mesmo com mudanças no código-fonte. Também foi discutida a inclusão dos testes automatizados na esteira de integração e entrega contínua, para que sejam executados com frequência. O resultado esperado desta atividade é que a equipe execute regularmente os testes automatizados para garantir a estabilidade da aplicação.

No terceiro encontro da semana, foi realizado um treinamento sobre a importância e como realizar revisão por pares, tanto de código-fonte quanto de código de teste, além da revisão de cenários de teste. Nessa atividade, a equipe de desenvolvimento e a equipe de testes trabalharam juntas para aprender a executar revisões de pares de forma eficaz. O objetivo dessa atividade é garantir que a equipe de desenvolvimento possa realizar revisões de pares com frequência e que a equipe de testes seja capaz de realizar revisões de cenários de teste de software.

6.5. Etapa Aferições

A etapa de aferições teve duração de 3 horas e 30 minutos, subdivididos em reuniões com a equipe de testes para, em conjunto, realizar o mapeamento do cenário alcançado, utilizado para comparativo de antes e depois e análise do cenário de teste de software. Esta etapa contou também com a análise dos dados e resultados alcançados e, com uma etapa muito importante de comunicação dos resultados, que contou com reuniões entre a equipe de desenvolvimento e o gestor responsável para divulgação de resultados de melhoria do processo de teste de software.

A primeira atividade desta tarefa foi a “Identificação do cenário alcançado após a execução do plano de ação”. Essa etapa consistiu em uma reunião de uma hora com a equipe de testes de software, na qual foram analisadas em conjunto as práticas de teste de software e a frequência de utilização dessas práticas. A participação dos membros das equipes foi bastante relevante, e ao longo da reunião, foram fornecidos feedbacks muito positivos sobre a melhoria da estruturação dos testes, a inclusão de testes automatizados e, principalmente, a automação do *deploy* por meio de ferramentas de integração e entrega contínua.

A segunda etapa deste processo consistiu na análise dos resultados obtidos após a execução do plano de ação. Foram coletados dados e comparados o cenário anterior à implementação das melhorias no teste de software com o cenário após a implantação dessas melhorias. Além disso, foi elaborado um material de apresentação para os gestores e líderes, destacando a evolução da equipe de teste de software em relação à maturidade e melhoria do processo de teste.

Como resultado da análise do cenário alcançado versus o cenário anterior, pode-se observar a Tabela 7.

Tabela 7: Cenário Alcançado – Organização Beta

Prática em Teste de Software	Equipe 01 - ANTES	Equipe 01 - DEPOIS
Desenvolvimento orientado a testes	Nunca	Nunca
Testes unitários	Nunca	Frequentemente
Testes unitários utilizam mocks	Nunca	Frequentemente
São aplicados testes de mutação	Nunca	Nunca
Testes unitários são necessários para conclusão da história	Nunca	Sempre
Análise estática de código	Nunca	Sempre
Testes regressivos	Nunca	Raramente
Há um ambiente específico para execução dos testes	Sempre	Sempre
O status de retorno das aplicações são validados	Frequentemente	Sempre
Os campos no banco de dados são validados	Frequentemente	Sempre
Testes funcionais automatizados	Nunca	Frequentemente
Testes automatizados estão integrados na esteira CI/CD	Nunca	Sempre
Erro nos testes acarreta falha na esteira CI/CD	Nunca	Sempre
Ocorre desativação de testes(bypass) na esteira CI/CD	Não sei informar	Nunca
Testes funcionais são necessários para conclusão da história	Sempre	Sempre
Testes de integração com banco de dados ou outras aplicações	Nunca	Nunca
Testes de performance	Nunca	Nunca
Testes de stress	Nunca	Nunca
Testes não funcionais são necessários para conclusão da histór	Nunca	Nunca
Testes exploratórios	Nunca	Nunca
São escritos cenários/planos de teste	Sempre	Sempre
Há padrões na escrita dos cenários/planos de teste	Nunca	Sempre
Revisão por pares dos cenários/plano de teste	Nunca	Frequentemente
São evidenciadas as execuções de teste	Sempre	Sempre
Ocorre a documentação dos defeitos encontrados	Nunca	Sempre
Resultados dos testes são utilizados para melhoria do produto	Frequentemente	Frequentemente
São estimadas horas para planejamento/execução dos testes	Sempre	Sempre
Os testes são baseados em requisitos de negócios	Frequentemente	Sempre
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkins)	Nunca	Sempre
Execução dos testes são realizados com frequência	Raramente	Frequentemente
São planejados quantidade de testes por nível da pirâmide de testes	Nunca	Frequentemente
aonde se encontra a maior quantidade de testes da equipe?	Testes manuais	Testes unitários

Fonte: O Autor

Observando a Tabela 7, verificou-se melhorias significativas na frequência e no nível de maturidade do teste de software realizados pela equipe. Dentre as práticas de teste, destaca-se o aumento do uso de testes unitários com *Mocks*, que foram incluídos como obrigatórios para

conclusão das histórias. Além disso, houve uma melhoria na análise estática de código e na execução de testes regressivos, que passaram a ser executados na esteira de integração e entrega contínua.

A equipe de teste de software anteriormente a aplicação do plano de ação apresentava um baixo nível de maturidade, uma vez que muitas práticas importantes estavam ausentes, como a realização de testes unitários, análise estática de código, testes regressivos e a própria esteira de integração e entrega contínua não existia. Além disso, a revisão por pares dos cenários/planos de teste e a documentação dos defeitos encontrados também eram pouco frequentes. Após a implementação do plano de ação, foi possível observar um avanço significativo em relação ao nível de maturidade da equipe de testes, uma vez que as práticas de teste de software foram implementadas como a implementação de testes unitários, análise estática de código, documentação dos defeitos encontrados, implantação de esteira de integração e entrega contínua e, sua integração com os testes automatizados. Além disso, a equipe também passou a realizar a revisão por pares dos cenários/planos de teste e a planejar a quantidade de testes com base na pirâmide de testes.

Com as melhorias, a equipe conseguiu avançar para o nível 2 de maturidade, o nível Gerenciado, de acordo com as descrições do TMMI. Foram implementadas diversas práticas, como a utilização de *mocks* nos testes unitários, o foco na criação de testes unitários e de integração, a implementação de testes funcionais automatizados, a revisão por pares, a documentação dos defeitos encontrados, a estimativa de horas para planejamento e execução dos testes.

Um ponto positivo a ser destacado é que a equipe mantém um ambiente específico para execução dos testes e possui uma gestão de ciclo de vida de aplicação. Além disso, os resultados dos testes são utilizados para a melhoria contínua do produto.

Outro aspecto que apresentou grande melhoria foi a utilização de validações por meio de status de retorno das aplicações e de valores no banco de dados, o que demonstra uma preocupação maior em validar o funcionamento das aplicações.

Com base nos resultados, foi elaborado um material de apresentação para os gestores e líderes, destacando a evolução da equipe de testes de software em relação à maturidade e melhoria do processo de teste.

Um ponto de grande melhoria que a organização alcançou foi a adoção de um sistema de integração e entrega contínua, o que permitiu incluir testes unitários, testes funcionais automatizados e testes regressivos a serem executados sempre que há uma atualização no código-fonte da aplicação.

A revisão por pares dos cenários de teste também trouxe benefícios à equipe de testes, resultando em uma maior frequência na análise dos requisitos de negócio e dos resultados dos testes. Os testes unitários também passaram a ser a maioria dos testes realizados nas aplicações e a pirâmide de testes é agora sempre considerada na inclusão de novos testes.

A equipe de testes de software obteve um avanço significativo em relação ao seu nível de maturidade em teste de software, de acordo com o TMMI. Apesar de já possuir um planejamento de testes e ambientes de execução de testes, a equipe conseguiu incrementar seus processos por meio do monitoramento e controle, utilizando os resultados da esteira de integração e entrega contínua. Além disso, houve uma organização mais eficiente dos testes e implementação da revisão por pares. Com isso, a equipe alcançou o nível 2 - Gerenciado, incluindo algumas etapas do nível 3 - Definido.

Para a análise desta equipe de testes de software, foi dedicado um total de uma hora e meia para preparação do material de análise comparativo, documento este, usado como base para apresentação dos resultados aos gestores e liderança.

A última atividade da etapa, a "Comunicação dos resultados obtidos" se deu por meio de uma reunião de duração de uma hora para apresentação dos resultados obtidos pela equipe de testes em maturidade e melhoria em teste de software.

6.6. Entrevista

Após a implementação do RoMTeS, uma entrevista foi conduzida com os gestores e líderes envolvidos na aplicação do *roadmap*, com o objetivo principal de analisar a percepção deles sobre as melhorias alcançadas no teste de software. A entrevista foi realizada por meio de uma reunião relativamente curta, contando com a participação do líder técnico da equipe de testes de software, do gestor e do coordenador da equipe. Foram formuladas sete questões-chave para avaliar o produto e coletar feedback. A entrevista foi feita juntamente com o líder técnico da equipe de testes de software, o gestor e o coordenador da equipe.

A primeira pergunta realizada na entrevista visou obter a percepção da liderança em relação ao aumento do nível de maturidade em teste de software após a implementação do RoMTeS. A resposta obtida foi positiva, indicando que a liderança percebeu claramente os benefícios proporcionados pela implantação do *roadmap* de melhoria de teste de software. Os aspectos que mais agregaram valor à organização foram a inclusão e configuração da esteira de integração e *deploy* contínuo, o que permitiu maior autonomia e liberdade para as publicações

das aplicações, antes feitas de forma manual. Além disso, a padronização de escrita de cenários de teste, a inclusão de testes automatizados e a aplicação dos testes na esteira de integração contínua contribuíram para a estruturação dos testes na organização. De modo geral, a liderança teve uma percepção muito positiva do RoMTeS e dos benefícios proporcionados por sua aplicação na equipe de testes de software.

A segunda pergunta realizada foi sobre a facilidade de uso do RoMTeS, ou seja, se é um *roadmap* de difícil ou fácil aplicação. Em resposta, a liderança comentou que a aplicação do *roadmap* é relativamente fácil, mas é necessário que uma pessoa com conhecimento das práticas de teste de software oriente nas ações a serem tomadas e forneça suporte à equipe de testes para sanar possíveis dúvidas que possam surgir.

A terceira pergunta realizada foi sobre a percepção da liderança quanto à viabilidade do RoMTeS, ou seja, se é um *roadmap* realizável. A liderança concordou que sim, o *roadmap* é um produto viável e traz resultados muito positivos em relação à automação de processos de *deploy*, padronização de processos e disseminação de conhecimento entre a equipe de testes e as equipes de desenvolvimento de software.

A quarta pergunta realizada foi sobre a percepção da liderança em relação à utilidade do RoMTeS, ou seja, se o *roadmap* pode ser aplicado em organizações que atuam com o desenvolvimento ágil de software utilizando a arquitetura de micro serviços. De maneira geral, a liderança respondeu que acreditam que o *roadmap* pode ser utilizado não somente em organizações que atuam com a arquitetura de micro serviços, mas também em outras arquiteturas e em organizações que não tenham um processo de teste estruturado, como era o caso antes da aplicação do RoMTeS. Além disso, a liderança destacou a possibilidade de aplicação do *roadmap* em equipes que sejam focadas em teste e qualidade, como a equipe de testes de software da organização em questão.

A quinta pergunta feita na entrevista foi sobre a possibilidade de comercialização do RoMTeS. A liderança concordou que estaria disposta a não apenas realizar parcerias para a aplicação do *roadmap* na equipe de testes de software, mas também para sua aplicação em equipes de desenvolvimento de software, a fim de disseminar e ampliar conhecimentos, além de melhorar práticas em relação ao teste de software.

Na sexta pergunta, questionou-se a liderança sobre sua experiência com a implantação do RoMTeS. Em resposta, a liderança comentou que a experiência foi muito positiva, uma vez que a automação de processos, estruturação dos testes e padronização de processos contribuíram para a melhoria da qualidade das aplicações da organização. Além disso, a liderança mencionou que durante o processo de implantação do *roadmap*, receberam feedbacks

muito positivos da equipe de testes de software e de membros de equipes de desenvolvimento que começaram a enxergar a estruturação da equipe de processos de teste de software como um valor para a organização como um todo.

Como última pergunta, foi questionado à liderança se eles recomendariam a aplicação do *Roadmap* de Melhoria de Teste de Software para outras organizações e equipes que atuam no ambiente de desenvolvimento de software. A resposta unânime da liderança foi que sim, eles recomendariam a aplicação do RoMTeS não só para outras equipes de desenvolvimento de software, mas também para outras equipes que atuam na área de testes e qualidade das aplicações. Eles ressaltaram que a aplicação do *roadmap* trouxe resultados muito positivos para a organização, como a automação de processos, a estruturação dos testes e a padronização dos processos, o que levou a uma melhora significativa na qualidade das aplicações desenvolvidas. Além disso, durante o processo de implantação do *roadmap*, a equipe recebeu feedbacks muito positivos não só da equipe de testes de software, mas também de membros das equipes de desenvolvimento que começaram a ver a estruturação da equipe de testes de software e seus processos.

Ao final da entrevista, a liderança expressou que a aplicação do RoMTeS trouxe muitos benefícios para a organização, como a melhoria na qualidade das aplicações, a automação de processos, a padronização de práticas e a disseminação de conhecimentos entre as equipes de teste e desenvolvimento de software. Além disso, ressaltaram que a aplicação do *roadmap* pode ser realizada em diferentes arquiteturas de software e em organizações que ainda não tenham um processo de teste estruturado. A liderança também se mostrou disposta a recomendar a aplicação do RoMTeS para outras organizações e equipes que atuem no ambiente de desenvolvimento de software. Em resumo, a aplicação do *roadmap* de melhoria de teste de software foi considerada um sucesso pela liderança, trazendo benefícios tangíveis para a organização.

6.7. Considerações parciais

A implementação do *roadmap* RoMTeS em uma organização de porte médio com estrutura e maturidade de testes classificada como inicial pelo TMMI foi um grande desafio. Durante as três semanas de implantação, foram percebidas melhorias significativas para a organização, que passou a utilizar como métricas a quantidade de testes unitários realizados.

O tempo dedicado pela equipe de testes para aplicar as melhorias discutidas em reuniões semanais foi essencial para os resultados obtidos. Além disso, esse período foi uma excelente oportunidade de aprendizado para os membros da equipe de testes, que deram feedbacks muito positivos sobre a experiência.

Em geral, a liderança avaliou a implementação do RoMTeS como uma experiência excepcional e recomendou sua aplicação não apenas às equipes de desenvolvimento de software, mas também às organizações voltadas para testes e qualidade das aplicações. A implantação do RoMTeS gerou resultados positivos, como automação de processos, padronização dos procedimentos e disseminação de conhecimentos entre as equipes.

A aplicação do *roadmap* nesta organização apresentou um desafio significativo devido à característica de todas as reuniões serem presenciais. No entanto, a equipe de testes ficou muito satisfeita com as práticas de teste de software apresentadas no plano de ação, especialmente com a integração contínua por meio da automação de processos de *deploy* e a inclusão de testes na esteira de integração contínua. A equipe demonstrou grande disposição para aprender sobre as práticas de teste de software e aplicá-las em projetos da organização, o que contribuiu significativamente para a estruturação dos testes na organização e elevou o nível de maturidade da equipe em testes de software. Além disso, o tempo dedicado pela equipe de testes para aplicar as melhorias discutidas em reuniões semanais foi essencial para os resultados obtidos e proporcionou um período de muito aprendizado e feedbacks positivos.

No que diz respeito à liderança da equipe, observou-se que todos se mostraram satisfeitos com os resultados obtidos e sugeriram que, após um período de 6 meses, o plano estratégico de aprimoramento dos testes de software fosse aplicado novamente, com o objetivo de elevar ainda mais o nível de maturidade da equipe em relação aos testes de software.

Um fator interessante sobre esta equipe foi a inclusão de uma métrica para análise da qualidade das aplicações: a cobertura de código por testes unitários. A equipe reconheceu a importância dessa métrica e passou a monitorá-la de perto. Além disso, destacou-se a habilidade da equipe em aprender rapidamente sobre as práticas de teste de software e aplicá-las nos projetos da organização, o que contribuiu para a melhoria do nível de maturidade em teste de software da equipe. Apesar das reuniões presenciais terem sido um desafio, a equipe de testes elogiou todas as práticas abordadas no plano de ação, em especial a integração contínua por meio da automação de processos de *deploy* e inclusão de testes na esteira de integração contínua. O tempo dedicado pela equipe para aplicar as melhorias discutidas em reuniões semanais foi essencial para os resultados obtidos e proporcionou um período de muito aprendizado, conforme destacado pelos feedbacks muito positivos recebidos.

7. IMPLANTAÇÃO DO ROMTES EM EMPRESA DE GRANDE PORTE

A implementação do *roadmap* RoMTeS na Organização Gama, uma empresa de grande porte do setor financeiro com aproximadamente 105 mil funcionários no país, foi um desafiador devido à grande quantidade de equipes de desenvolvimento envolvidas no processo de melhoria de teste de software. No entanto, de forma geral, foi um processo de grande crescimento para todos os envolvidos.

Por questões de segurança da informação e proteção de marca, os resultados divulgados nesta pesquisa serão limitados a informações estritamente relacionadas ao *roadmap* RoMTeS.

As conversas sobre a implantação do *roadmap* RoMTeS foram realizadas com gestores e superintendentes da Organização Gama. Desde o primeiro encontro, ficou evidente que estavam empolgados com os resultados que poderiam ser atingidos. A organização apresentou nas conversas, uma preocupação com a qualidade e testes de software e, por isso, apoiou fortemente a implementação do *roadmap* em 13 equipes de desenvolvimento de software.

Em todos os grupos, a empresa adota métodos ágeis e utiliza arquitetura de micro serviços na construção das aplicações, características que proporcionaram uma certa uniformidade nas práticas de desenvolvimento e testes de software.

Neste conjunto de grupos de desenvolvimento de software, todos utilizam o desenvolvimento de software como uma atividade meio para a organização, produzindo softwares de categoria de aplicações Web. As equipes de desenvolvimento são compostas por 6 a 9 profissionais, que atuam de forma remota e são responsáveis tanto pelo desenvolvimento quanto pelos testes das aplicações.

As 13 equipes de desenvolvimento de software utilizam o Scrum para gerenciamento de projetos e, no planejamento das iterações, levam em consideração apenas as atividades de desenvolvimento, enquanto os testes são avaliados apenas como métricas de qualidade, sendo exigido um mínimo de 50% de cobertura de código por meio de testes unitários para que a aplicação possa ser implantada em ambiente produtivo. As equipes são compostas por 6 a 9 profissionais, sendo padronizadas com 2 profissionais sênior, 3 plenos e 4 juniores, todos responsáveis pela qualidade das aplicações desenvolvidas.

A organização adota uma iteração de 3 semanas para desenvolvimento e testes, sendo que as equipes consideram utilizar a pirâmide de testes para orientar a quantidade de testes criados. No entanto, na análise dos dados da implementação do *roadmap* RoMTeS, avalia a real quantidade de testes realizados, com foco na automação dos testes para obter feedback mais

rápido e facilidade de automação. A cobertura de código por meio da porcentagem de testes unitários por linha de código é uma das métricas prioritárias de qualidade para a organização.

Outra característica que se aplica a todas as equipes de desenvolvimento nesta organização é que cada equipe é liderada por um profissional que assume a função de líder técnico. Este desenvolvedor é responsável pela liderança técnica do grupo de desenvolvimento de software. Além disso, todos os membros da equipe trabalham de forma remota, o que permitiu a implementação do *roadmap* de forma totalmente remota.

A aplicação do *roadmap* RoMTeS ocorreu durante o período de janeiro a fevereiro de 2023, ocupando uma iteração completa de cada uma das equipes de desenvolvimento de software. Os resultados obtidos com a implementação do *roadmap* de melhoria de teste de software RoMTeS foram muito positivos, resultando na melhoria da qualidade dos testes de software. Além disso, houve um aumento significativo no nível de maturidade em testes de software das equipes envolvidas no processo.

Com base na estrutura da organização, elaborou-se um cronograma semanal para aplicação do *roadmap*, com margem para que houvesse adequações de horários, se necessário. O cronograma da Semana 01 pode ser visto por meio da Figura 19.

Figura 19: Cronograma Empresa Gama – Semana 01

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00				Equipe7 - Avaliação	
10:00	Esclarescimentos com Gestores	Apresentação do RoMTeS as equipes	Equipe4 - Avaliação	Equipe7 - Plano de ação	Equipe11 - Avaliação
11:00			Equipe4 - Plano de ação	Equipe8 - Avaliação	Equipe11 - Plano de ação
12:00				Equipe8 - Plano de ação	Equipe12 - Avaliação
13:00		Equipe1 - Avaliação			Equipe12 - Plano de ação
14:00		Equipe1 - Plano de ação	Equipe5 - Avaliação		
15:00	Esclarecimentos com Líderança	Equipe2 - Avaliação	Equipe5 - Plano de ação	Equipe9 - Avaliação	Equipe13 - Avaliação
16:00		Equipe2 - Plano de ação	Equipe6 - Avaliação	Equipe9 - Plano de ação	Equipe13 - Plano de ação
17:00		Equipe3 - Avaliação	Equipe6 - Plano de ação	Equipe10 - Avaliação	
18:00		Equipe3 - Plano de ação		Equipe10 - Plano de ação	Apresentação dos planos de ação aos Gestores

Fonte: O Autor

Com base na Figura 19, pode-se observar a primeira semana de implantação do *roadmap* RoMTeS como uma semana de esclarecimentos, definições e avaliação de cada uma das 13

equipes de desenvolvimento de software, além de elaboração do plano de ação para cada equipe e apresentação dos planos de ação aos gestores e liderança da equipe. O cronograma da Semana 02, Semana03 e Semana 04 são idênticos, como pode ser visto por meio da Figura 20.

Figura 20: Cronograma Empresa Gama – Semana 02, 03 e 04

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
9:30		Equipe6 - Encontro1			
10:00	Equipe1 - Encontro1		Equipe12 - Encontro1	Equipe4 - Encontro2	Equipe9 - Encontro2
10:30					
11:00		Equipe7 - Encontro1			
11:30	Equipe2 - Encontro1		Equipe12 - Encontro1	Equipe5 - Encontro2	Equipe10 - Encontro2
12:00					
12:30					
13:00					
13:30		Equipe8 - Encontro1			
14:00					
14:30	Equipe3 - Encontro1		Equipe1 - Encontro2	Equipe6 - Encontro2	Equipe11 - Encontro2
15:00		Equipe9 - Encontro1			
15:30					
16:00	Equipe4 - Encontro1		Equipe2 - Encontro2	Equipe7 - Encontro2	Equipe12 - Encontro2
16:30		Equipe10 - Encontro1			
17:00					
17:30	Equipe5 - Encontro1		Equipe3 - Encontro2	Equipe8 - Encontro2	Equipe13 - Encontro2
18:00		Equipe11 - Encontro1			
18:30					

Fonte: O Autor

Com base na Figura 20, pode-se observar que a segunda, terceira e quarta semana são iguais em termos de calendário. Isto se dá porque, de acordo o planejamento do *roadmap*, este tempo é destinado para aplicação do plano de plano de ação específico para cada equipe de desenvolvimento. Já o cronograma da última semana de implantação do *roadmap*, a Semana 05 pode ser visto por meio da Figura 21.

Figura 21: Cronograma Empresa Gama – Semana 05

HORA	SEGUNDA-FEIRA	TERÇA-FEIRA	QUARTA-FEIRA	QUINTA-FEIRA	SEXTA-FEIRA
9:00					
10:00	Equipe1 - Reavaliação	Equipe6 - Reavaliação	Equipe11 - Reavaliação	Apresentação Resultados 01	Apresentação dos Resultados Gerais
11:00	Equipe2 - Reavaliação	Equipe7 - Reavaliação	Equipe12 - Reavaliação		
12:00					
13:00					
14:00	Equipe3 - Reavaliação	Equipe8 - Reavaliação	Equipe13 - Reavaliação	Apresentação Resultados 02	
15:00	Equipe4 - Reavaliação	Equipe9 - Reavaliação	Organização dos dados		
16:00				Apresentação Resultados 03	
17:00	Equipe5 - Reavaliação	Equipe10 - Reavaliação			
18:00	Organização dos dados	Organização dos dados			

Fonte: O Autor

Com base na Figura 21, é possível observar a última semana de implantação do *roadmap*, que foi destinada à reavaliação de cada uma das equipes de desenvolvimento de software, organização e análise dos dados do cenário alcançado, além da apresentação dos resultados obtidos aos gestores e liderança. Devido ao grande número de equipes de desenvolvimento de software, a pedido dos gestores da organização, foram incluídas 3 reuniões de apresentação dos resultados. A primeira reunião de apresentação de resultados contou com o Gestor 01 e suas 4 equipes de desenvolvimento de software. A segunda reunião de apresentação de resultados contou com o Gestor 02 e suas 5 equipes de desenvolvimento de software. A terceira reunião de apresentação de resultados contou com a Gestora 03 e suas 4 equipes de desenvolvimento de software. Além dessas reuniões específicas, houve também uma reunião geral de apresentação dos resultados. Na ocasião, foram apresentados os dados obtidos durante a implementação do RoMTeS, que indicaram melhorias significativas nos processos e testes de software, bem como o aumento do nível de maturidade em testes de software das equipes.

A etapa inicial da implantação do RoMTeS foi a de esclarecimentos, que consistiu em explicar aos membros das equipes de desenvolvimento de software e gestores da organização sobre a metodologia a ser aplicada, suas etapas e objetivos.

7.1. Etapa Esclarecimentos

A etapa de esclarecimentos teve duração de 2 horas em reunião com a superintendente e gestores da organização Gama.

A atividade chamada de "Esclarecimento do RoMTeS aos gestores" foi realizada por meio de uma reunião com a liderança da equipe. A equipe era composta por uma superintendente e três gestores responsáveis pelas equipes de desenvolvimento de software. A reunião teve duração de uma hora. Durante a reunião, foi apresentada a estrutura do *roadmap*, bem como cada etapa e tempo médio definido para cada atividade. As dúvidas que surgiram foram tratadas na reunião, de forma que, ao final da reunião, todos os envolvidos puderam entender o *roadmap* de melhoria de teste de software.

A segunda atividade nesta etapa do *roadmap* é chamada de "Esclarecimento da abrangência do RoMTeS". Esta atividade envolveu uma segunda conversa com os gestores e a superintendente para escolher quais equipes seriam abrangidas pela implementação do *roadmap* de melhoria de teste de software, utilizando o RoMTeS, bem como definir o tempo semanal a ser investido nessas equipes. Esta definição foi feita por meio de uma reunião de 1 hora e meia, na qual ficou acordado que o RoMTeS seria aplicado às 13 equipes de desenvolvimento de software, e que seriam realizados dois encontros semanais para reuniões de melhoria do processo de teste de software, com uma duração total de 3 horas semanais. Cada encontro semanal teria uma duração de uma hora e meia.

Durante essa reunião, além da seleção da equipe de testes de software, foram registrados os nomes e contatos de cada líder técnico das equipes de desenvolvimento de software. Ao final da reunião com a liderança, um e-mail contendo um resumo da reunião e a definição de tempo de atuação e responsáveis/líderes técnicos pelas equipes selecionadas foi enviado a todos os participantes. Além disso, um e-mail foi enviado aos membros das equipes selecionadas, com cópia para seus respectivos gestores e liderança, informando sobre o processo de melhoria de teste de software e agendando uma reunião com eles.

Finalizada a etapa de esclarecimentos, pode-se seguir para a etapa Definições, mais bem descrita a seguir.

7.2. Etapa Definições

A etapa de definições teve duração de 3 horas e 30 minutos, subdivididos em reuniões que envolveram os gestores, líderes técnicos das equipes de desenvolvimento de software e, as próprias equipes de desenvolvimento de software.

Nesta etapa do *roadmap*, temos a atividade "Apresentação do RoMTeS aos líderes técnicos". Esta atividade foi realizada por meio de uma reunião de uma hora com os líderes técnicos de cada uma das equipes de desenvolvimento de software, abordando detalhes mais técnicos sobre todo o processo de implementação do *roadmap* de melhoria de teste de software. Durante a reunião, foram apresentados os objetivos do *roadmap*, bem como foram solucionadas algumas dúvidas levantadas pelos líderes técnicos.

Durante essa reunião de uma hora, houve uma grande importância em pedir o engajamento dos líderes técnicos e de seus respectivos grupos de trabalho. Isso ocorreu devido aos benefícios obtidos ao final da implementação do *roadmap* de melhoria de teste de software, tais como a melhoria do processo de testes de software, a capacitação das equipes e o aumento da qualidade das entregas e aplicações desenvolvidas.

Outra atividade importante nesta etapa do *roadmap* é a "Apresentação do RoMTeS às equipes de desenvolvimento de software". Essa atividade envolveu todos os membros das 13 equipes de desenvolvimento de software e foi realizada por meio de uma apresentação dos objetivos do *roadmap*, além de esclarecer dúvidas conforme surgiam durante a reunião.

Durante a reunião de 1 hora e meia, foi acordado um horário específico para a próxima atividade: a identificação do cenário atual de cada equipe de desenvolvimento de software. Essa atividade foi realizada por meio de reuniões de 1 hora com cada equipe e sua liderança técnica, com a presença do gestor da equipe na primeira reunião. Foi essencial para que o aplicador do *roadmap* e a equipe de desenvolvimento de software pudessem analisar e definir o cenário atual da equipe, preenchendo um formulário com práticas de teste de software aplicadas na equipe.

A tabela 8 apresenta de forma compilada a análise inicial feita para cada uma das 13 equipes de desenvolvimento de software.

Tabela 8: Cenário Inicial – Organização Gama

Práticas em Teste de Software	ANTES DA IMPLEMENTAÇÃO DO ROMTES												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Desenvolvimento orientado a testes	F	N	N	S	F	N	N	N	N	R	F	N	F
Testes unitários	F	F	F	F	F	F	F	F	F	F	F	F	F
Testes unitários utilizam mocks	F	F	N	S	F	R	S	S	F	S	R	R	F
São aplicados testes de mutação	N	N	N	F	F	S	N	S	R	R	N	N	R
Testes unitários são necessários para conclusão da história	S	S	S	S	F	F	S	S	F	S	R	R	S
Análise estática de código	S	S	S	S	F	F	N	S	R	R	N	N	F
Testes regressivos	F	S	S	S	F	F	N	N	R	R	R	N	R
Há um ambiente específico para execução dos testes	S	S	S	S	F	F	S	R	R	N	S	S	S
O status de retorno das aplicações são validados	F	S	S	S	F	F	S	N	F	S	R	R	S
Os campos no banco de dados são validados	N	N	F	S	S	N	N	N	N	N	R	R	F
Testes funcionais automatizados	S	S	S	S	S	F	S	S	F	S	S	S	S
Testes automatizados estão integrados na esteira CI/CD	S	S	S	S	S	S	S	S	S	S	S	S	S
Erro nos testes acarreta falha na esteira CI/CD	S	S	S	S	S	S	S	S	S	S	S	S	S
Ocorre desativação de testes(bypass) na esteira CI/CD	N	N	N	N	R	R	N	N	R	N	N	N	R
Testes funcionais são necessários para conclusão da história	F	F	F	F	F	F	F	F	F	F	F	F	F
Testes de integração com banco de dados/outras aplicações	F	S	N	S	R	S	S	S	R	S	F	R	S
Testes de performance	N	N	R	N	F	F	N	N	N	S	N	R	F
Testes de stress	N	N	R	N	F	F	N	N	N	S	N	N	S
Testes não funcionais são necessários para as entregas	N	N	N	N	N	N	N	N	N	N	N	N	N
Testes exploratórios	N	S	R	S	R	S	N	S	N	R	N	N	R
São escritos cenários/planos de teste	R	N	N	S	F	R	N	R	N	S	N	N	R
Há padrões na escrita dos cenários/planos de teste	F	S	N	S	S	R	R	N	N	R	S	N	S
Revisão por pares dos cenários/plano de teste	N	N	N	N	N	N	N	N	N	N	N	N	N
São evidenciadas as execuções de teste	N	N	R	F	F	R	N	R	R	N	N	S	R
Ocorre a documentação dos defeitos encontrados	N	N	N	S	S	N	S	N	N	S	N	N	N
Resultados dos testes são usados para melhoria do produto	N	N	N	N	N	S	N	N	N	N	N	N	S
São estimadas horas para planejamento/execução dos testes	S	S	N	S	S	S	S	N	S	S	S	N	S
Os testes são baseados em requisitos de negócios	N	S	S	S	S	S	S	R	N	S	S	N	S
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenk	S	S	S	S	S	S	S	S	S	S	S	S	S
Execução dos testes são realizados com frequência	F	F	F	F	F	F	F	F	F	F	F	F	F
São planejados quantidade de testes por nível da pirâmide de testes	R	S	N	S	S	S	N	N	N	S	S	S	S
Aonde se encontra a maior quantidade de testes da Equipe?	TU	TM	TU	FA	TU	FA	TU	TU	TU	TU	FA	TU	FA
Nível de maturidade - TMMi	2	2	2	2	2	2	2	1	2	1	2	1	2

"S" significa Sempre	"TU" significa Testes de Unidade
"F" significa Frequentemente	"TM" significa Testes Manuais
"R" significa Raramente	"FA" significa Testes Funcionais Automatizados
"N" significa Nunca	

Fonte: O Autor

Como pode ser observado na Tabela 8, foi realizado um levantamento em conjunto com a cada equipe de desenvolvimento de software sobre as práticas de teste aplicadas no dia a dia. Além disso, foram coletados detalhes técnicos das aplicações desenvolvidas e peculiaridades de cada equipe para que, com base nessas informações, fosse possível desenvolver um plano de ação específico para melhorar o processo de teste de software da equipe.

Para a Equipe 01, destaca-se que a maioria dos testes aplicados pela equipe são testes de unidade, porém, não há um planejamento para organizá-los de acordo com a pirâmide de

testes. Além disso, a equipe informou que os testes nunca são baseados em requisitos de negócio, o que é um ponto de atenção que será abordado no plano de ação.

Na etapa atual, além da avaliação das práticas de teste de software, foi realizada uma avaliação do nível de maturidade em testes de software da Equipe01 com base no TMMI, o que permitiu o avanço do *roadmap* para a próxima etapa de averiguações. A análise do nível de maturidade da equipe resultou em um nível 2, ou seja, nível Gerenciado em maturidade de teste de software. Isso significa que a equipe apresenta uma estratégia e planejamento de testes, monitoramento e controle por meio de uma esteira de integração contínua, execução de testes e a criação de ambientes específicos para executar os testes.

Já em relação a Equipe 02, são utilizados principalmente testes manuais, apesar de ter um plano para organizá-los de acordo com a pirâmide de testes. Durante a avaliação das práticas de teste de software, verificou-se que a equipe tem um entendimento equivocado sobre a pirâmide de testes.

Além da avaliação das práticas de teste de software, a equipe foi avaliada quanto ao nível de maturidade em testes de software com base no TMMI, permitindo a continuidade do *roadmap* para a etapa de averiguações. A análise do nível de maturidade da Equipe 02 indicou o nível 2 de maturidade, ou seja, nível Gerenciado em maturidade de teste de software, devido à existência de uma estratégia e planejamento de testes, monitoramento e controle através da esteira de integração contínua, além da execução de testes e preocupação com a criação de ambientes específicos para a execução dos testes. Nesta equipe de desenvolvimento de software, observou-se que os testes unitários são de qualidade insuficiente, com validações pouco eficientes, o que aumenta o risco de falhas na aplicação, mesmo que os testes aparentem estar passando sem problemas. É necessário aprimorar as práticas de teste para garantir a qualidade das entregas e evitar problemas futuros.

A análise da Equipe 03 revelou que a equipe realiza a maior parte dos testes na forma de testes unitários, porém não há um planejamento adequado com base na pirâmide de testes. Além disso, destaca-se que a equipe utiliza informações de requisitos de negócios como base para os testes realizados, o que é um ponto positivo. Com base nessas informações, será possível desenvolver um plano de ação específico para melhorar o processo de teste de software da equipe.

Nesta etapa, além da avaliação das práticas de teste de software, também foi avaliado o nível de maturidade em testes de software que a Equipe03 apresentou, com base no TMMI, permitindo o avanço do *roadmap* para a etapa de averiguações. O resultado dessa análise indicou que a equipe se encontra no nível 2 de maturidade, o nível Gerenciado em maturidade

de teste de software. Isso se deve ao fato de que há uma estratégia e planejamento de testes, monitoramento e controle, por meio da esteira de integração contínua, e à preocupação da equipe em criar ambientes específicos para a execução dos testes e em executar testes não funcionais. Além disso, é importante destacar que a equipe utiliza informações de requisito de negócios como base para os testes realizados, apesar de não realizar o planejamento desses testes com base na pirâmide de testes. Esta equipe de desenvolvimento de software é muito participativa, todos muito bem integrados com as aplicações que possuem e durante a reunião foram muito participativos.

A análise da Equipe 04 demonstrou que ela tem como prática a aplicação de testes funcionais automatizados, porém, identificamos que há um entendimento equivocado da pirâmide de testes, o que pode impactar a organização dos testes em diferentes níveis.

Nesta etapa, além da avaliação das práticas de teste de software, também foi avaliado o nível de maturidade em testes de software da Equipe 04 com base no TMMI. Isso permitiu avançar no *roadmap* para a etapa de averiguações. A análise resultou no nível 2 de maturidade, ou seja, nível Gerenciado em maturidade de teste de software. Essa classificação é atribuída porque a equipe possui uma estratégia e planejamento de testes, realiza monitoramento e controle por meio da esteira de integração contínua, executa os testes e preocupa-se com a criação de ambientes específicos para a execução dos testes. Observou-se que a qualidade dos testes unitários desenvolvidos por esta equipe de desenvolvimento de software é insuficiente, o que indica a necessidade de uma reestruturação dos testes na equipe.

A análise de cenário atual da Equipe 05 demonstrou que ela realiza a maior parte dos testes na forma de testes unitários, em quantidade adequada conforme a estratégia de testes baseada na pirâmide de testes. Entretanto, é curioso notar que a equipe não utiliza os resultados dos testes como meio de melhorar a qualidade do produto.

Nesta etapa, além da avaliação das práticas de teste de software, foi também avaliado o nível de maturidade em testes de software que a equipe apresentou, com base no TMMI, permitindo o avanço do *roadmap* para a etapa de averiguações. Esta análise sobre o nível de maturidade da Equipe05 teve como resultado o nível 2 de maturidade, ou seja, nível Gerenciado em maturidade de teste de software, visto que há uma estratégia e planejamento de testes, monitoramento e controle, por meio da esteira de integração contínua, há também a execução dos testes e, a preocupação com a criação de ambientes específicos para a execução dos testes.

A análise de cenário atual da Equipe 06 demonstrou que a equipe realiza a maior quantidade de testes funcionais automatizados. No entanto, constatou-se que o planejamento não está totalmente alinhado com a pirâmide de testes, o que indica um entendimento

equivocado por parte da equipe de desenvolvimento. Além disso, verificou-se que há uma frequência elevada na execução de testes manuais. Dessa forma, é importante que a equipe revise seus conceitos e pratique uma abordagem mais estruturada em relação à pirâmide de testes, a fim de otimizar o processo de teste e minimizar o esforço manual.

Nesta etapa, além da avaliação das práticas de teste de software, também foi avaliado o nível de maturidade da equipe em testes de software, com base no TMMI. Esse processo permitiu avançar o *roadmap* para a etapa de averiguações. Como resultado da análise, foi identificado que a Equipe06 está no nível 2 de maturidade em testes de software, ou seja, no nível Gerenciado. Isso se deve à presença de uma estratégia e planejamento de testes, monitoramento e controle por meio da esteira de integração contínua, execução de testes e criação de ambientes específicos para a execução dos testes. Esse resultado é um indicador positivo do compromisso da equipe com a qualidade do software e sugere que eles estão no caminho certo para alcançar níveis mais elevados de maturidade em testes de software no futuro.

Durante a reunião, um dos integrantes da Equipe 06 relatou que a qualidade dos testes unitários desenvolvidos é muito baixa. Isso ocorre porque a equipe tem dificuldade em identificar os cenários de teste mais adequados e as validações mais relevantes a serem realizadas. Essa falta de clareza prejudica a eficácia dos testes unitários, comprometendo a qualidade do software. Portanto, é fundamental que a equipe desenvolva habilidades para a criação de testes unitários de alta qualidade, por meio de treinamentos, revisões e boas práticas de desenvolvimento de software. Com isso, será possível garantir uma cobertura adequada dos testes unitários e minimizar a ocorrência de erros e defeitos no software.

Em relação a análise de cenário da Equipe 07 revelou que os testes unitários são a principal estratégia de teste da equipe e, ela planeja seus testes de acordo com a pirâmide de testes. Essa abordagem é importante para garantir a qualidade do software desenvolvido. No entanto, uma das características dessa equipe é que ela não utiliza os resultados dos testes como forma de melhorar o produto. É importante que a equipe utilize os resultados dos testes para identificar áreas de melhoria e implementar ações corretivas que possam contribuir para o aprimoramento do software. Dessa forma, será possível aumentar a efetividade dos testes e a qualidade do produto entregue aos usuários.

Na presente etapa, além da avaliação das práticas de teste de software, a equipe também passou por uma avaliação do nível de maturidade em testes de software, baseada no TMMI. Isso permitiu o avanço do *roadmap* para a próxima etapa de averiguações. A análise revelou que a Equipe07 está no nível 2 de maturidade em teste de software, o que significa que possui

um nível Gerenciado de maturidade. Essa conclusão foi baseada na existência de uma estratégia e planejamento de testes, monitoramento e controle por meio de uma esteira de integração contínua, bem como na preocupação da equipe em criar ambientes específicos para execução de testes. Além disso, a equipe executa regularmente seus testes. Esses são aspectos fundamentais para um nível gerenciado de maturidade em teste de software.

Com essa avaliação, a equipe poderá identificar pontos fortes e oportunidades de melhoria em sua abordagem de teste de software. Isso ajudará a equipe a melhorar ainda mais a qualidade do software entregue aos usuários e garantir que os objetivos do projeto sejam alcançados. Durante a reunião, percebeu-se que esta equipe de desenvolvimento de software não demonstrou muito interesse nos assuntos tratados. Embora tenham respondido sobre a frequência das práticas de teste de software aplicadas no dia a dia da equipe, foi notada uma falta de entusiasmo em relação ao processo de melhoria de teste de software.

A análise de cenário Equipe 08 demonstrou que ela utiliza principalmente testes unitários, conforme indicado pela maior quantidade de testes nessa camada. Embora a pirâmide de testes seja usada para o planejamento, também foi identificada uma quantidade relativamente grande de testes manuais, e a equipe nunca executa testes de regressão. Esse comportamento sugere um entendimento equivocado da pirâmide de testes, que pode ser abordado por meio do plano de ação proposto.

Durante esta etapa, realizamos não apenas uma avaliação das práticas de teste de software da equipe, mas também uma avaliação do seu nível de maturidade em testes, com base no TMMI. Essa avaliação foi fundamental para avançarmos no *roadmap* e seguirmos para a próxima etapa de averiguações. Como resultado, verificamos que a Equipe08 possui um nível de maturidade 1 em testes de software, ou seja, um nível Inicial, o que significa que a equipe já possui uma estratégia e planejamento de testes, bem como uma esteira de integração contínua em operação.

A análise de cenário atual d a Equipe 09 demonstra que ela realiza a maioria dos testes na forma de testes unitários. No entanto, verificamos que esses testes não são planejados considerando a pirâmide de testes, o que pode impactar na qualidade do software produzido. Além disso, notamos que a equipe não utiliza adequadamente os resultados dos testes para otimizar as aplicações desenvolvidas. Portanto, é recomendável que a equipe revise suas práticas de teste e considere uma abordagem mais estruturada e completa, com o objetivo de garantir a qualidade e a confiabilidade do software produzido.

Durante esta etapa, avaliamos não apenas as práticas de teste de software da Equipe02, mas também seu nível de maturidade em testes, com base no TMMI. Essa análise permitiu

avançar no *roadmap* para a próxima etapa de averiguações. Como resultado, identificamos que a equipe apresenta um nível de maturidade 2 em testes de software, ou seja, um nível Gerenciado. Isso se deve ao fato de que a equipe possui uma estratégia e planejamento de testes, monitora e controla seu processo de teste por meio da esteira de integração contínua e realiza a execução dos testes. Além disso, notamos que a equipe tem preocupação com a criação de ambientes específicos para a execução dos testes, o que demonstra um esforço em aprimorar sua prática de teste de software.

A análise de cenário atual da Equipe 10 revela que a Equipe 10 prioriza os testes unitários em seu planejamento de testes, em consonância com a pirâmide de testes. Entretanto, notamos que a equipe realiza testes manuais com pouca frequência, o que pode ser um ponto de atenção em relação à eficácia do processo de teste. Além disso, chamou a atenção a alta frequência de execução de testes de desempenho (*performance*) e testes de estresse em comparação com os testes unitários. Isso pode indicar um entendimento equivocado da pirâmide de testes e da importância dos testes unitários como base do processo de teste. Com essas informações em mente, podemos desenvolver um plano de ação que ajude a Equipe 10 a repensar sua estratégia de teste e priorizar adequadamente os diferentes tipos de testes para garantir a qualidade e a confiabilidade de seus softwares.

Além da avaliação das práticas de teste de software, realizamos uma avaliação do nível de maturidade em testes de software da Equipe05, com base no TMMI. Esse diagnóstico permitiu avançarmos no *roadmap* para a etapa de averiguações e identificar áreas que necessitam de melhoria. O resultado da análise revelou que a equipe possui o nível 1 de maturidade em teste de software, ou seja, o nível Inicial. Isso indica que a equipe precisa aprimorar suas práticas de teste, incluindo o estabelecimento de uma estratégia e planejamento de testes mais robustos, bem como a adoção de medidas para garantir a qualidade e a confiabilidade do software desenvolvido. Com esse conhecimento em mãos, poderemos desenvolver um plano de ação específico para ajudar a Equipe05 a evoluir em seu nível de maturidade em testes de software.

A análise de cenário atual da Equipe 11 demonstra que a maior quantidade de testes são de testes funcionais automatizados, embora haja um planejamento para organizar os testes de acordo com a pirâmide de testes, é perceptível que a equipe de desenvolvimento tem um entendimento inadequado sobre esse conceito. Além disso, a equipe não utiliza os resultados dos testes como uma oportunidade de melhoria para o produto.

Durante esta etapa, além da avaliação das práticas de teste de software, também avaliamos o nível de maturidade em testes de software que a Equipe 06 apresentou, utilizando

o modelo TMMI. Isso permitiu o avanço do *roadmap* para a etapa de averiguações. Com base em nossa análise, a equipe alcançou o nível 2 de maturidade, ou seja, o nível Gerenciado em maturidade de teste de software.

A equipe de desenvolvimento de software demonstrou utilizar a pirâmide de testes como base para a criação de testes, conforme indicado na Tabela 8. No entanto, durante a reunião, foi identificado que a equipe não compreendeu completamente o conceito da pirâmide de testes, o que pode levar a uma aplicação inadequada das práticas de teste de software.

A análise de cenário atual da Equipe 12 revela que a Equipe 12 segue a pirâmide de testes, com a maior quantidade de testes sendo testes unitários. No entanto, a frequência de criação dos testes unitários é considerada baixa pela equipe, sendo raramente utilizados para finalizar as histórias/entregas. Além disso, é importante destacar que a equipe não utiliza os resultados dos testes como forma de melhorar o produto.

Nesta etapa, além da avaliação das práticas de teste de software, avaliou-se também o nível de maturidade em testes de software apresentado pela Equipe12, com base no TMMI, permitindo avançar o *roadmap* para a etapa de averiguações. O resultado da análise sobre o nível de maturidade da equipe foi o nível 1 de maturidade, ou seja, nível Inicial em maturidade de teste de software. Embora a equipe tenha práticas como planejamento dos testes e monitoramento e controle por meio da esteira de integração CI/CD, é necessário avançar em outras áreas para alcançar níveis mais altos de maturidade em testes de software.

A análise de cenário atual da Equipe 13 revela que ela tem a maior quantidade de testes funcionais automatizados, embora a equipe afirme que usa a pirâmide de testes como base para o planejamento dos testes, o que sugere que a equipe não entende corretamente o conceito da pirâmide de testes na aplicação dos testes. Além disso, é interessante notar que a equipe informou que executa testes de estresse sempre e testes de desempenho (*performance*) frequentemente, o que pode indicar equívocos no entendimento do conceito da pirâmide de testes.

Nesta etapa, além de avaliar as práticas de teste de software da Equipe02, também foi realizada uma avaliação do nível de maturidade em testes de software da equipe, com base no TMMI. Com isso, foi possível avançar no *roadmap* para a etapa de averiguações. De acordo com os resultados da análise, a equipe apresentou um nível 1 de maturidade em teste de software, o que indica que estão no estágio Inicial de maturidade.

Com a definição do cenário atual para cada equipe de desenvolvimento, pode-se evoluir para a próxima etapa do *roadmap* RoMTeS, a etapa de averiguações, mais bem descrita a seguir.

7.3. Etapa Averiguações

A etapa de averiguações no total, durou 27 horas, para a qual cada equipe de desenvolvimento deve um investimento de 2 horas, subdivididos em análise e estruturação dos dados de cenário atual, construção do plano de ação específico para a equipe e uma reunião geral com a liderança, reunião de duração de uma hora para apresentação e validação dos planos de ação.

A atividade de "Análise e organização dos dados do cenário atual da(s) equipe(s) de desenvolvimento de software" compreende a análise estruturada dos dados coletados em conjunto com cada equipe de desenvolvimento de software sobre as práticas de teste aplicadas e o nível de maturidade em teste de software de cada equipe. Após a reunião de levantamentos, esta atividade teve duração média de uma hora por equipe. A atividade seguinte, "Construção do plano de melhoria para cada equipe de desenvolvimento de software", durou uma hora em média por equipe e buscou identificar as ações necessárias para melhorar o nível de maturidade da equipe de testes com base no modelo de maturidade de teste de software TMMI. O plano de ação resultante foi desenvolvido com a definição das atividades necessárias para a melhoria do teste de software da equipe de testes. A imagem do plano de ação pode ser encontrada no capítulo sobre Melhorias.

De maneira geral, o plano de ação teve como principal objetivo melhorar o nível de maturidade em teste de software da equipe de testes, por meio de ações estruturadas para a organização dos testes. Foram definidas atividades como aprimoramento da documentação de testes, definição de padrões e diretrizes para a realização de testes, treinamento da equipe em práticas de teste adequadas e a implementação de ferramentas de automação de testes. Tais ações visam garantir maior eficiência e efetividade nos processos de teste de software, contribuindo para a melhoria da qualidade do produto e a satisfação dos clientes.

A implantação do *roadmap* foi planejada para ocorrer em três semanas e envolveu nove reuniões, cada uma com duração de uma hora e meia. Durante esse período, foram realizadas diversas atividades visando melhorar o nível de maturidade em teste de software da equipe de testes. Entre as atividades realizadas, destacam-se a aplicação de treinamentos sobre pirâmide de testes, testes unitários com uso de *Mocks*, documentação de cenários de teste e defeitos encontrados, testes regressivos e revisão por pares de cenários de teste, testes de código e código-fonte das aplicações. Além disso, cada equipe de desenvolvimento recebeu atividades específicas, que serão detalhadas no capítulo de Melhorias.

Outra atividade importante nesta etapa foi a apresentação do plano de ação para os gestores. Para isso, foi realizada uma reunião de meia hora na qual foram apresentados aos líderes técnicos todos os planos para melhorias nos testes de software. Nessa reunião, foram discutidos e alinhados os resultados esperados com a execução de cada atividade do plano de ação.

7.4. Etapa Melhorias

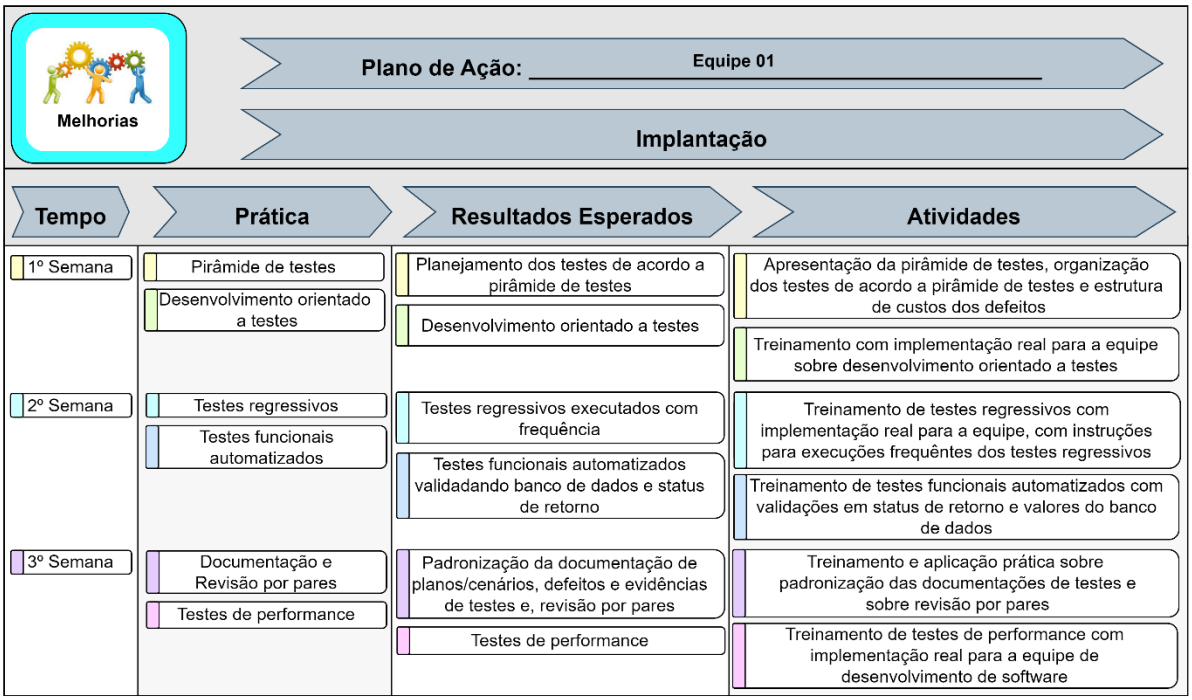
A etapa de melhorias teve a duração de três semanas, o que equivale a uma iteração da equipe de desenvolvimento de software, com um total de 13 horas de atividades. Essa etapa foi subdividida em duas atividades principais: a apresentação do plano de ação para cada equipe de desenvolvimento de software e a execução desse plano.

A primeira atividade consistiu em uma reunião objetiva com cada equipe de desenvolvimento para apresentar os caminhos traçados para melhoria do teste de software. Cada reunião teve duração média de uma hora e foi realizada remotamente. Foi solicitado aos participantes que mantivessem suas câmeras abertas durante a reunião para aumentar o engajamento das equipes. Foi observada uma empolgação das equipes em relação às melhorias que seriam realizadas.

Alocou-se um tempo semanal de 3 horas para cada equipe de desenvolvimento de software atuar nos planos de ação definidos para melhoria dos testes de software, conforme acordado com os gestores previamente. Esse tempo foi considerado na criação dos planos de ação para cada equipe, de modo que, ao final das três semanas de aplicação dos planos, pudesse-se observar melhorias efetivas no processo de teste de software.

A segunda atividade desta etapa consistiu na aplicação dos planos de ação por cada equipe de trabalho, com duração de três semanas e duas reuniões semanais de uma hora e meia para cada equipe. O plano de ação elaborado para a Equipe01, juntamente com sua descrição, pode ser visualizado na Figura 22 abaixo.

Figura 22: Plano de ação – Organização Gama – Equipe 01



Fonte: O Autor

Como pode ser visto na Figura 22, o plano de ação da Equipe 01 consistiu em uma série de atividades que foram executadas ao longo de três semanas, com forte participação da equipe de desenvolvimento de software e duas reuniões semanais, cada uma com duração de uma hora e meia.

Durante a primeira semana, as atividades foram focadas em práticas relacionadas à pirâmide de testes e ao desenvolvimento orientado a testes. Na primeira reunião da semana, foi realizado um treinamento sobre a pirâmide de testes e como organizar e planejar os testes de forma que a maioria dos testes sejam automatizados e dispostos na base da pirâmide, enquanto a menor quantidade de testes esteja no topo. O objetivo dessa atividade era que a equipe fosse capaz de realizar o planejamento dos testes levando em consideração a pirâmide de testes.

Na segunda reunião da semana, foi realizado um treinamento sobre desenvolvimento orientado a testes. A equipe foi orientada sobre os princípios de desenvolvimento que priorizam o teste unitário, bem como sobre como implementar códigos utilizando o desenvolvimento orientado a testes. O objetivo dessa atividade era fazer com que a equipe de desenvolvimento conseguisse criar testes unitários eficientes para suas aplicações e considerasse os testes unitários como uma etapa fundamental para a conclusão de uma história.

Na primeira reunião da segunda semana, foi realizado um treinamento sobre testes regressivos, destacando a importância de realizar esses testes com frequência, principalmente os testes automatizados, para garantir que a aplicação continue funcionando mesmo com constantes alterações no código-fonte. O objetivo dessa atividade é que a equipe execute os testes regressivos automatizados com frequência, garantindo a qualidade da aplicação.

Na segunda reunião da semana, a equipe recebeu treinamento sobre testes funcionais automatizados. Foi abordado como realizar testes funcionais automatizados eficientemente, identificando os principais pontos de falha da aplicação e criando scripts automatizados para garantir a execução desses testes de forma rápida e precisa. O objetivo dessa atividade é que a equipe possa executar testes funcionais automatizados de forma eficiente, garantindo a qualidade da aplicação e economizando tempo na execução dos testes.

Na terceira semana, foi realizado um treinamento sobre testes de desempenho, no qual a equipe aprendeu sobre a importância dos testes de desempenho e como executá-los de forma eficiente. Para esta atividade, o resultado esperado é que a equipe possa executar testes de desempenho efetivos em suas aplicações e que possa identificar gargalos e possíveis problemas de desempenho.

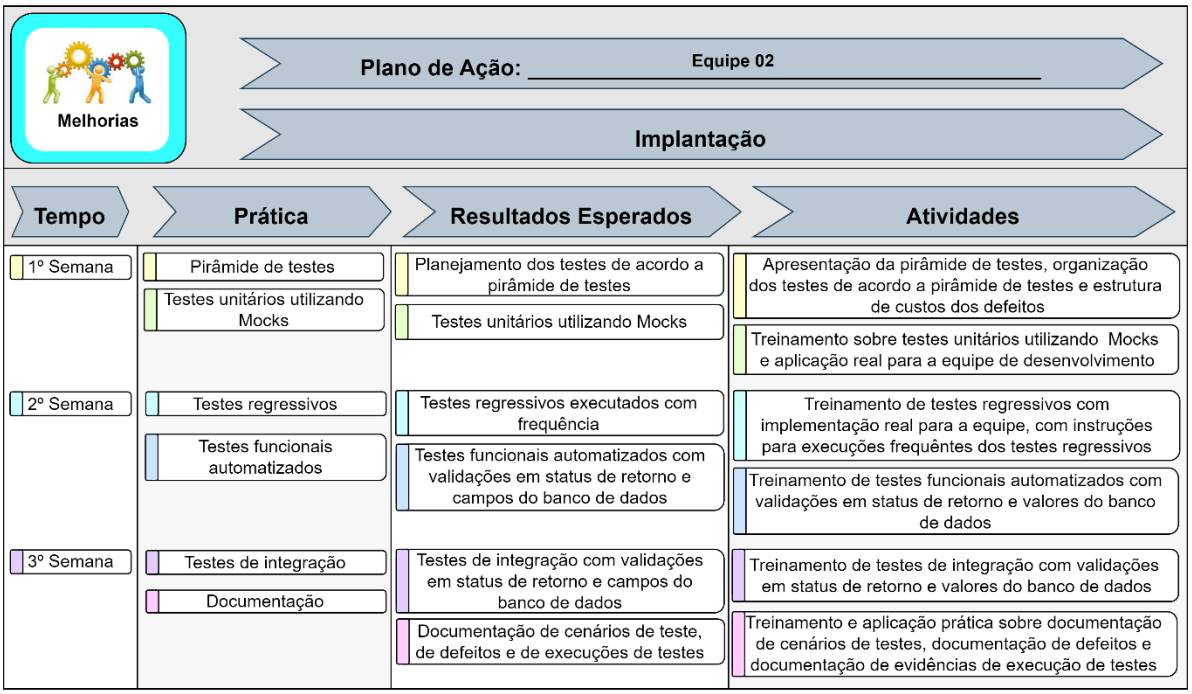
Além disso, foi abordada a importância da documentação dos testes, com foco na padronização da documentação de cenários de testes e defeitos. O objetivo é que a equipe possa documentar de forma clara e objetiva os cenários de testes e defeitos encontrados durante a execução dos testes.

Também foi destacada a importância das revisões por pares, tanto dos cenários de testes como do código-fonte da aplicação. Isso ajuda a garantir que o trabalho realizado esteja dentro dos padrões de qualidade e que possíveis problemas sejam identificados e corrigidos de forma mais rápida.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes de desempenho, que foi conduzido de forma prática, utilizando exemplos de aplicações desenvolvidas pela própria equipe para criar os testes. O objetivo dessa atividade foi capacitar a equipe para realizar testes de desempenho efetivos e utilizar os resultados obtidos como base para a melhoria das aplicações desenvolvidas pela equipe. O resultado esperado é que a equipe consiga aplicar os conceitos aprendidos na prática, de forma a identificar possíveis gargalos de performance nas aplicações e corrigi-los para garantir um melhor desempenho.

O plano de ação para a Equipe 02, bem como sua descrição, pode ser observado por meio da Figura 23.

Figura 23: Plano de ação – Organização Gama – Equipe02



Fonte: O Autor

Como pode ser observado na Figura 23, o plano de ação da Equipe 02 compreende atividades que serão realizadas ao longo de três semanas, através de duas reuniões de uma hora e meia cada, com uma participação ativa da equipe de desenvolvimento de software.

Esta equipe, em particular, apresentava uma estrutura de testes unitários muito limitada, com validações insuficientes que, apesar de serem contabilizadas para as métricas de cobertura de testes unitários, não eram efetivas para identificar erros que a aplicação poderia apresentar.

Considerando a equipe em particular, que tinha uma estrutura de testes unitários pouco efetiva, na segunda reunião da semana aplicou-se um treinamento sobre testes unitários utilizando *mocks*, uma técnica que permite simular a execução de objetos e dados para validar o comportamento da aplicação em diferentes cenários. O objetivo foi que a equipe aprendesse a criar testes unitários mais completos e efetivos, aumentando a cobertura de testes e reduzindo a incidência de erros. O resultado esperado é que a equipe consiga criar testes unitários mais robustos, utilizando a técnica de *mocks* para simular comportamentos, tornando a validação mais efetiva.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes unitários utilizando *Mocks*. Durante o treinamento, a equipe foi orientada sobre os princípios dos testes unitários e da utilização de *mocks* de objetos. A aplicação prática foi enfatizada, com a criação

de alguns testes unitários com *Mocks* nas aplicações desenvolvidas pela equipe, a fim de deixar exemplos de como utilizá-los em cenários futuros.

O objetivo dessa atividade foi capacitar a equipe de desenvolvimento a criar testes unitários eficientes, utilizando *Mocks* quando necessário. Além disso, a equipe deve considerar os testes unitários como uma etapa fundamental para a finalização de uma história. Dessa forma, espera-se que a equipe seja capaz de aplicar os conceitos aprendidos e produzir códigos de qualidade com testes robustos.

Na segunda semana, a equipe focou em atividades relacionadas aos testes regressivos e testes funcionais automatizados. Na primeira reunião, aplicou-se um treinamento que destacou a importância dos testes regressivos e a necessidade de realizá-los com frequência, especialmente com testes automatizados, para garantir que a aplicação continue funcionando mesmo com alterações frequentes no código-fonte. O objetivo desta atividade é que a equipe execute os testes regressivos com mais frequência, principalmente os testes automatizados, para garantir que a aplicação seja sempre funcional e estável.

Durante a segunda semana, a equipe se dedicou às atividades relacionadas aos testes funcionais automatizados. Na primeira reunião, foi ministrado um treinamento prático com aplicações da organização para automatizar fluxos funcionais, incluindo a integração desses testes na esteira de integração e entrega contínua. O objetivo desta atividade é garantir que a equipe possa realizar testes funcionais automatizados das aplicações por meio da esteira de integração CI/CD, obtendo assim maior eficiência e qualidade nos processos de desenvolvimento.

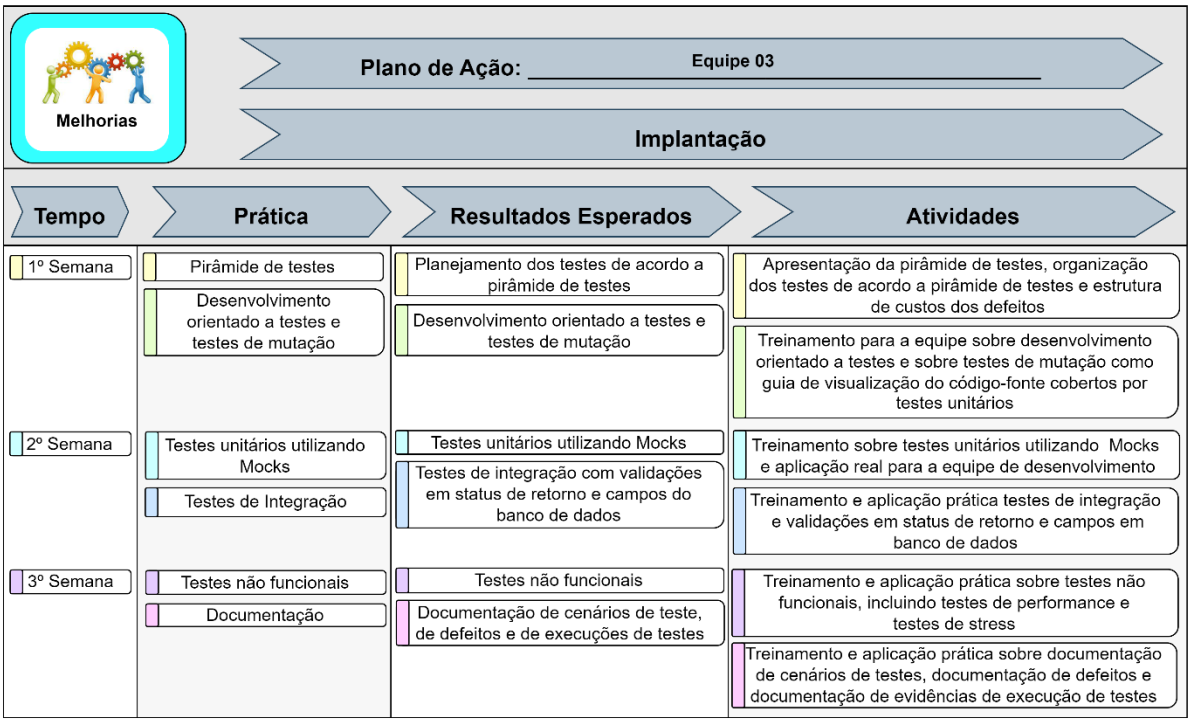
A terceira semana foi dedicada a atividades relacionadas a testes de integração, documentação dos testes e defeitos, e incluiu um treinamento sobre testes de desempenho. Na primeira reunião da semana, foi realizado um treinamento prático sobre testes de integração, utilizando aplicações já desenvolvidas pela equipe. Foram criados testes com validações automatizadas, incluindo a verificação do status de retorno das aplicações e dos valores dos campos no banco de dados. O resultado esperado é que a equipe possa criar e utilizar testes de integração com frequência, para garantir a qualidade da aplicação em diferentes cenários de uso. Além disso, a documentação dos testes e defeitos também foi abordada nessa semana, visando a organização e a fácil localização dessas informações.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre a padronização da documentação de cenários de testes e defeitos. O objetivo foi garantir que a equipe seja capaz de documentar adequadamente seus cenários de teste, relatar os defeitos encontrados durante as execuções e fornecer evidências claras das execuções dos testes. Como resultado esperado,

espera-se que a equipe possa documentar de forma padronizada, completa e clara, garantindo a rastreabilidade dos testes e a facilidade na comunicação entre os membros da equipe.

O plano de ação para a Equipe 03, bem como sua descrição, pode ser observado por meio da Figura 24.

Figura 24: Plano de ação – Organização Gama – Equipe03



Fonte: O Autor

Como observado na Figura 24, o plano de ação da Equipe 03 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi dedicada à prática de pirâmide de testes e desenvolvimento orientado a testes. Na primeira reunião, foi realizado um treinamento sobre a pirâmide de testes, com foco em organizar e planejar os testes de forma que a maioria deles seja automatizada e disposta na base da pirâmide, enquanto a menor quantidade está no topo. O objetivo dessa atividade foi garantir que a equipe possa planejar seus testes de maneira adequada, levando em consideração a pirâmide de testes.

Na segunda reunião da semana, foi aplicado um treinamento sobre desenvolvimento orientado a testes. A equipe foi orientada sobre os princípios do desenvolvimento priorizando o teste unitário e como implementar códigos utilizando o desenvolvimento orientado a testes.

Além disso, também foi apresentado sobre testes de mutação e como utilizar esses testes como guia para visualizar o código-fonte testado pelos testes unitários. O resultado esperado é que a equipe de desenvolvimento consiga criar testes unitários de forma eficiente para as suas aplicações e considere os testes unitários como uma etapa fundamental para concluir as histórias. Além disso, espera-se que a equipe utilize os testes de mutação para visualizar possíveis cenários de teste unitários e verificar as partes do código cobertas ou não pelos testes unitários.

A segunda semana foi dedicada a treinamentos sobre testes unitários utilizando *mocks* e testes de integração. Na primeira reunião da semana, a equipe recebeu orientação sobre os princípios dos testes unitários e de *mocks* de objetos, com uma forte ênfase na aplicação prática. Foram criados exemplos de testes unitários com *mocks* em aplicações já desenvolvidas pela equipe, com o objetivo de demonstrar como utilizar esses testes em cenários futuros. O resultado esperado dessa atividade é que a equipe de desenvolvimento seja capaz de criar testes unitários com *mocks* de forma eficiente, sempre que necessário, e considere os testes unitários como uma etapa fundamental para finalizar uma história.

Na segunda reunião da semana, foi aplicado um treinamento sobre testes de integração. Durante o treinamento, a equipe realizou atividades práticas utilizando aplicações já desenvolvidas pela equipe de desenvolvimento, com o objetivo de criar testes de integração e validar os status de retorno das aplicações, bem como os valores de campos no banco de dados. O resultado esperado desta atividade é que a equipe consiga criar e utilizar os testes de integração com frequência, garantindo assim a qualidade e confiabilidade das aplicações desenvolvidas.

A terceira semana foi dedicada a reuniões sobre testes não funcionais, incluindo testes de desempenho e testes de estresse, bem como sobre documentação de cenários de teste, defeitos e evidências de execução dos testes. Na primeira reunião, realizou-se um treinamento prático sobre testes não funcionais com exemplos em aplicações desenvolvidas pela equipe. Durante o treinamento, foram abordados testes de desempenho e testes de estresse. Como resultado esperado desta atividade, a equipe deve ser capaz de criar testes não funcionais automatizados, incluindo testes de desempenho e testes de estresse, de forma eficiente e eficaz.

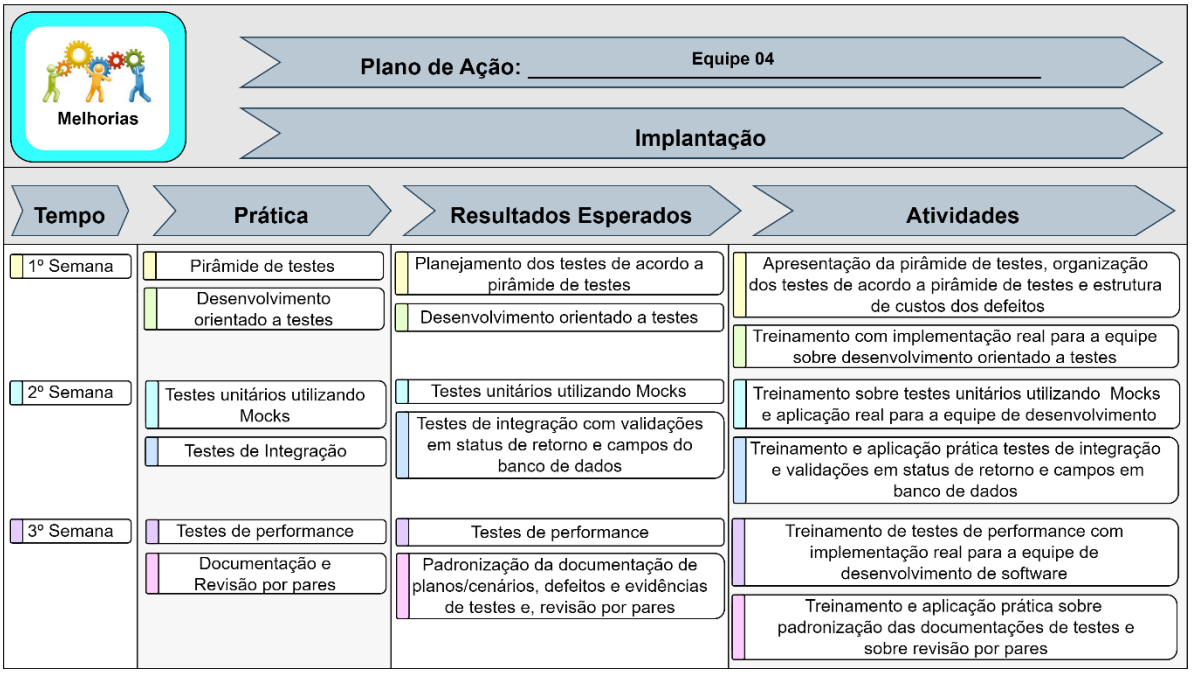
Na segunda reunião da semana, a equipe recebeu um treinamento sobre a padronização da documentação de cenários de testes e defeitos. O objetivo principal foi capacitar a equipe a documentar de forma adequada seus cenários de testes, bem como os defeitos encontrados nas aplicações e as evidências das execuções dos testes. Espera-se que, como resultado dessa

atividade, a equipe possa documentar seus testes de forma mais eficiente e padronizada, melhorando a rastreabilidade dos testes e a comunicação entre os membros da equipe.

É importante destacar que essa equipe vem apresentando evoluções significativas em relação aos temas abordados nas reuniões, o que demonstra o comprometimento e a dedicação dos seus membros.

O plano de ação para a Equipe 04, bem como sua descrição, pode ser observado por meio da Figura 25.

Figura 25: Plano de ação – Organização Gama – Equipe04



Fonte: O Autor

Como observado na Figura 25, o plano de ação da Equipe 04 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana teve como objetivo desenvolver práticas relacionadas à pirâmide de testes e ao desenvolvimento orientado a testes. Na primeira reunião, a equipe recebeu treinamento sobre a pirâmide de testes, aprendendo como organizar e planejar os testes de modo que a maioria deles seja automatizada e posicionada na base da pirâmide, enquanto uma quantidade menor de testes esteja no topo. O resultado esperado desta atividade é que a equipe consiga realizar o planejamento dos testes de maneira eficaz, levando em consideração a pirâmide de testes como uma estratégia importante para a qualidade do produto desenvolvido.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre desenvolvimento orientado a testes, que é uma abordagem de programação na qual o desenvolvimento de testes unitários é priorizado desde o início do processo de desenvolvimento. Durante o treinamento, a equipe aprendeu sobre os princípios do desenvolvimento orientado a testes e como implementar códigos utilizando essa abordagem. Foram realizadas atividades práticas para criar testes unitários em aplicações já desenvolvidas pela equipe, a fim de deixar exemplos de como utilizar em cenários futuros.

O objetivo desta atividade foi que a equipe de desenvolvimento pudesse criar testes unitários para as suas aplicações de forma eficiente, além de considerar os testes unitários como uma etapa fundamental para que uma história possa ser finalizada/concluída com sucesso. Com essa abordagem, espera-se que as aplicações sejam mais confiáveis e tenham menos erros, já que a equipe estará focada em testar cada funcionalidade à medida que é desenvolvida.

A segunda semana foi dedicada a treinamentos sobre testes unitários utilizando *Mocks* e testes de integração. Na primeira reunião, a equipe recebeu orientação sobre os princípios dos testes unitários e *mocks* de objetos, seguida por muita aplicação prática, onde foram criados testes unitários com *Mocks* em aplicações já desenvolvidas pela equipe. O objetivo foi deixar exemplos de como utilizar em cenários futuros. Espera-se que, ao final dessa atividade, a equipe de desenvolvimento seja capaz de criar testes unitários com a utilização de *Mocks*, quando necessário, para as suas aplicações de forma eficiente, bem como considerar os testes unitários como etapa fundamental para finalização de histórias.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes de integração com foco em aplicação prática. Utilizando aplicações já desenvolvidas, foram criados testes de integração que incluíam validações automatizadas tanto do status de retorno das aplicações quanto dos valores de campos no banco de dados. O objetivo desta atividade é capacitar a equipe para criar e utilizar testes de integração de forma eficiente e frequente.

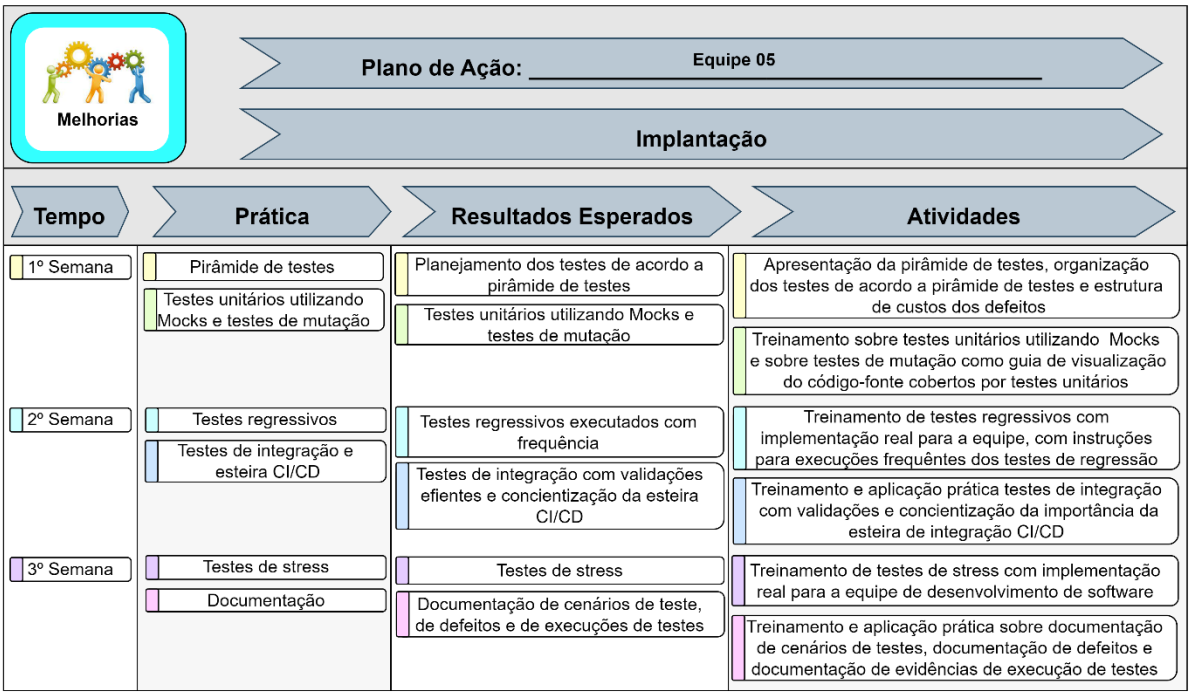
A terceira semana foi dedicada a reuniões sobre testes de desempenho e documentação de cenários de teste, defeitos e evidências de execução dos testes. Na primeira reunião, a equipe recebeu um treinamento prático sobre testes de desempenho, com exemplos em aplicações desenvolvidas pela equipe para ilustrar como criar cenários de testes de desempenho. O objetivo dessa atividade foi capacitar a equipe para criar testes de desempenho e utilizar os resultados para aprimorar os sistemas desenvolvidos.

Na segunda reunião da semana, aplicou-se um treinamento sobre padronização de documentação de cenários de testes e de defeitos, além de revisão por pares, não apenas dos cenários de teste, mas também do código-fonte das aplicações desenvolvidas pela equipe de

software. O objetivo foi garantir que todos os membros da equipe possam documentar seus cenários de teste, defeitos encontrados e evidências de execução dos testes de forma padronizada e eficiente, além de revisar o código para garantir sua qualidade e consistência. O resultado esperado é que a equipe melhore a qualidade de seus processos e entregas.

O plano de ação para a Equipe 05, bem como sua descrição, pode ser observado por meio da Figura 26.

Figura 26: Plano de ação – Organização Gama – Equipe05



Fonte: O Autor

Como observado na Figura 26, o plano de ação da Equipe 05 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

Durante a primeira semana, a equipe concentrou-se nas práticas de pirâmide de testes e testes unitários utilizando *Mocks*. Na primeira reunião da semana, realizou-se um treinamento sobre a pirâmide de testes, que enfatizou a importância de planejar e organizar os testes de forma que a maioria deles esteja automatizada e seja disposta na base da pirâmide, enquanto os testes menos frequentes ocupam o topo. O objetivo dessa atividade foi capacitar a equipe a planejar os testes de forma mais eficiente, levando em consideração a pirâmide de testes.

Na segunda reunião da semana, aplicou-se um treinamento sobre testes unitários utilizando *Mocks*, no qual a equipe foi orientada sobre os princípios dos testes unitários e de

mocks de objetos. Durante o treinamento, foram criados alguns testes unitários com *Mocks* nas aplicações desenvolvidas pela equipe, com o objetivo de deixar exemplos de como utilizar em cenários futuros. Nesta reunião, também foram abordados os testes de mutação como guia para os cenários de testes unitários.

Para esta atividade, o resultado esperado é que a equipe de desenvolvimento consiga criar testes unitários com a utilização de *Mocks*, identificar os melhores cenários para automatizar os testes unitários e considerar os testes unitários como etapa fundamental para a finalização das histórias. Além disso, espera-se que a equipe entenda como os testes de mutação podem ser utilizados para melhorar a qualidade dos testes unitários.

Durante a segunda semana, foram realizados treinamentos sobre testes regressivos e testes de integração com a esteira de integração contínua/entrega contínua CI/CD. Na primeira reunião, a equipe recebeu um treinamento sobre a importância dos testes regressivos e como realizar esses testes com frequência, especialmente os testes automatizados, para garantir que a aplicação continue funcionando mesmo com as constantes alterações no código-fonte. O resultado esperado dessa atividade é que a equipe execute com frequência os testes automatizados regressivos, a fim de garantir a estabilidade e a qualidade da aplicação.

Durante a segunda reunião da semana, foi ministrado um treinamento sobre testes de integração. Por meio de aplicações já desenvolvidas pela equipe de desenvolvimento, foram criados testes com validações automatizadas, incluindo o status de retorno das aplicações e valores de campos no banco de dados. Além disso, foi discutida a importância de uma esteira de integração e entrega contínua (CI/CD) e a necessidade de se seguir os padrões de qualidade configurados nessa esteira. Como resultado, espera-se que a equipe possa criar e utilizar os testes de integração com frequência, mantendo a qualidade e integridade dos sistemas desenvolvidos.

A terceira semana foi dedicada a reuniões sobre testes de estresse e documentação de cenários de teste, defeitos e evidências de execução dos testes. Na primeira reunião, foi aplicado um treinamento prático sobre testes de estresse, com exemplos nas aplicações desenvolvidas pela equipe para auxiliar na criação de cenários de testes. O objetivo desta atividade é capacitar a equipe a criar testes de estresse e utilizar os resultados para aprimorar os sistemas desenvolvidos.

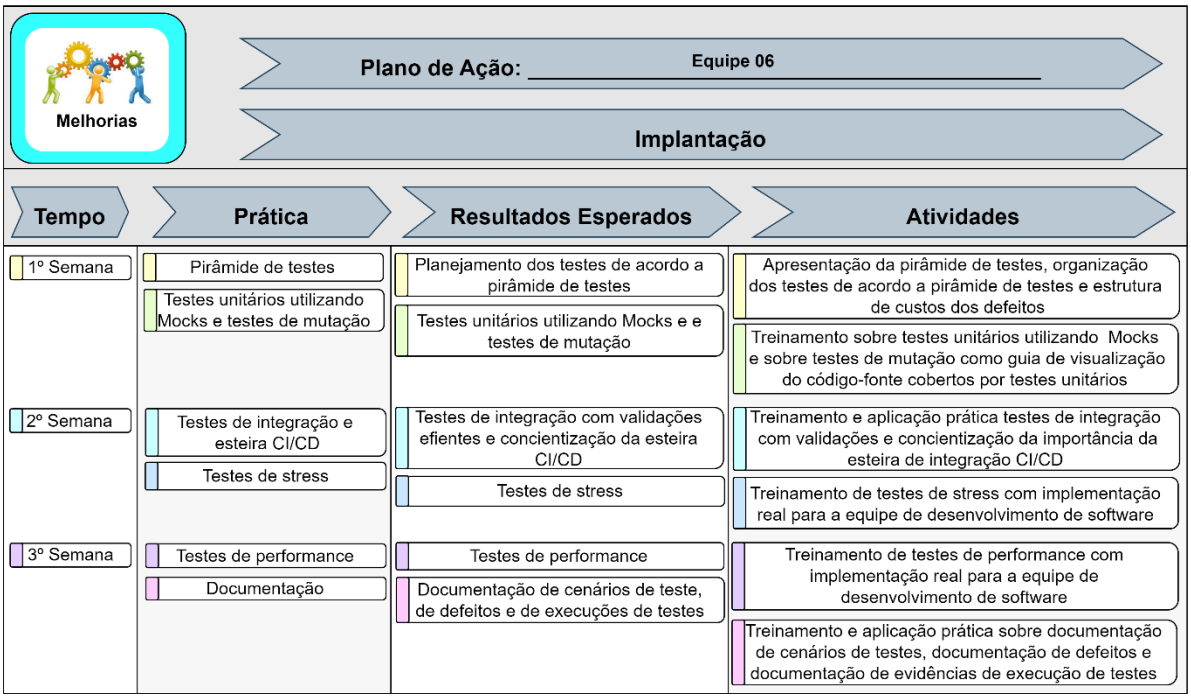
Na segunda reunião da semana, a equipe recebeu um treinamento sobre padronização da documentação de cenários de teste e defeitos. Durante o treinamento, foi enfatizada a importância da documentação para facilitar o entendimento e a comunicação entre a equipe.

Foram apresentados exemplos de documentação de cenários de teste, defeitos encontrados nas aplicações e evidências das execuções dos testes.

Como resultado esperado desta atividade, espera-se que a equipe adote a padronização da documentação de cenários de teste e defeitos, garantindo que todas as informações relevantes sejam registradas e compartilhadas com a equipe, melhorando a qualidade do processo de desenvolvimento de software.

O plano de ação para a Equipe 06, bem como sua descrição, pode ser observado por meio da Figura 27.

Figura 27: Plano de ação – Organização Gama – Equipe06



Fonte: O Autor

Como observado na Figura 27, o plano de ação da Equipe 06 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi dedicada às práticas de pirâmide de testes e testes unitários com uso de *Mocks* e testes de mutação. Na primeira reunião, foi realizado um treinamento sobre a pirâmide de testes, que ensinou à equipe como organizar e planejar os testes de forma que a maioria deles sejam automatizados e dispostos na base da pirâmide, enquanto uma menor quantidade esteja no topo. O objetivo dessa atividade foi fazer com que a equipe pudesse planejar seus testes com base na pirâmide, aumentando a eficiência e qualidade dos testes.

Na segunda reunião da primeira semana, a equipe recebeu um treinamento sobre testes unitários utilizando *mocks*. Durante a sessão, a equipe foi orientada sobre os princípios dos testes unitários e de *mocks* de objetos, e foram criados alguns exemplos práticos de testes unitários com *mocks* nas aplicações já desenvolvidas pela equipe. O objetivo do treinamento foi permitir que a equipe possa utilizar *mocks* em cenários futuros, bem como identificar os melhores cenários e validações para os testes unitários de forma eficiente. Além disso, a sessão também abordou o tema dos testes de mutação como uma ferramenta para visualização de cenários de testes unitários a serem construídos.

O resultado esperado dessa atividade é que a equipe de desenvolvimento possa criar testes unitários eficientes utilizando *mocks* quando necessário e considere os testes unitários como uma etapa fundamental para que uma história possa ser finalizada/concluída.

A primeira reunião da semana tratou sobre as práticas da pirâmide de testes e como organizar os testes de forma que a maioria deles sejam automatizados e dispostos na base da pirâmide. O treinamento teve como objetivo permitir que a equipe planeje os testes levando em consideração a pirâmide de testes.

O resultado esperado dessa atividade é que a equipe possa planejar seus testes de forma organizada, levando em consideração a pirâmide de testes e, dessa forma, ter uma base sólida para o desenvolvimento de testes automatizados.

Na segunda semana, foram realizados treinamentos sobre testes de integração, a esteira de integração e entrega contínua (CI/CD) e testes de estresse. Na primeira reunião, foi aplicado um treinamento prático sobre testes de integração. Utilizando aplicações já desenvolvidas pela equipe, foram criados alguns testes de integração com validações automatizadas em status de retorno das aplicações e valores de campos no banco de dados. Além disso, a equipe recebeu orientações sobre a importância da esteira de integração contínua e entrega contínua, bem como a necessidade de respeitar os padrões de qualidade definidos na esteira. O resultado esperado dessa atividade é que a equipe consiga criar e utilizar os testes de integração com frequência e que respeite os padrões mínimos de qualidade definidos na esteira de entrega contínua.

Na segunda reunião da semana, a equipe participou de um treinamento prático sobre testes de estresse, no qual foram apresentados exemplos de cenários e aplicações já desenvolvidas pela equipe. Durante o treinamento, os participantes tiveram a oportunidade de criar cenários de testes de estresse e realizar validações em aplicações reais, visando identificar possíveis gargalos e falhas que possam comprometer a desempenho dos sistemas.

O resultado esperado desta atividade é que a equipe consiga criar e aplicar testes de estresse de forma mais eficiente e com maior frequência, permitindo que os sistemas

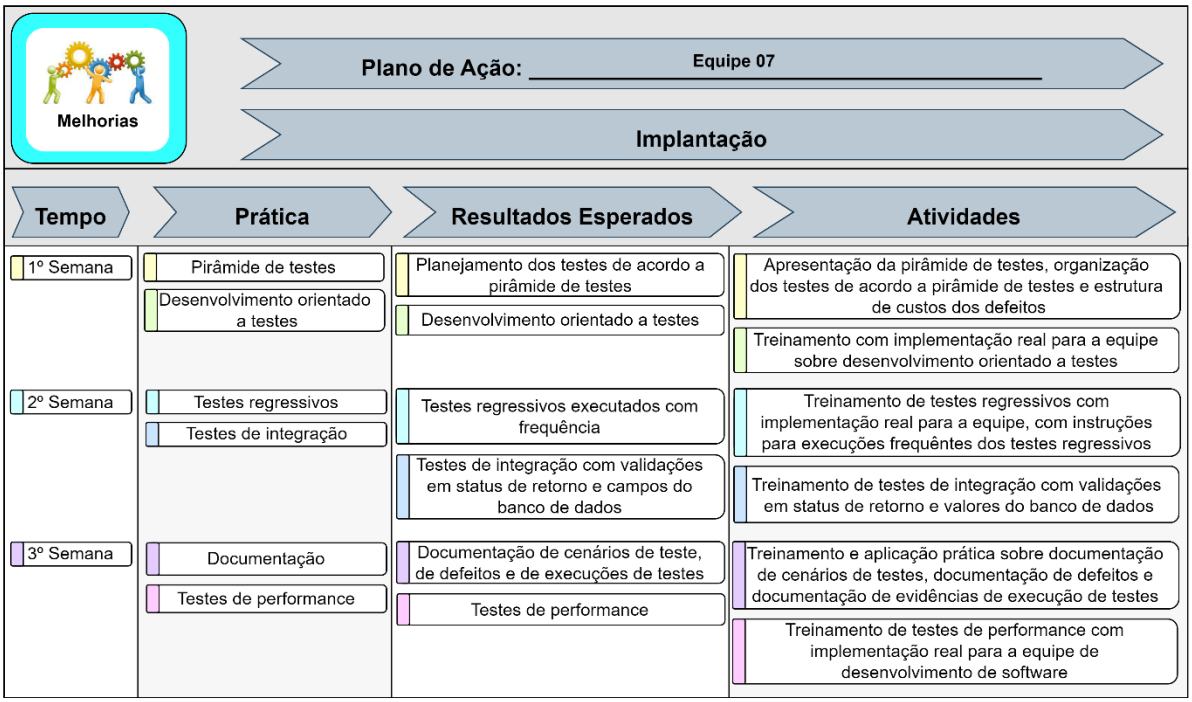
desenvolvidos atinjam um desempenho mais satisfatório. Além disso, espera-se que a equipe utilize os resultados obtidos nos testes de estresse para identificar e corrigir possíveis problemas de desempenho, contribuindo para a melhoria contínua dos sistemas desenvolvidos.

A terceira semana foi composta por reuniões sobre testes de desempenho, documentação de cenários de teste, documentação de defeitos e evidências de execução dos testes. Na primeira reunião da semana, foi aplicado um treinamento prático sobre testes de desempenho, com exemplos em aplicações desenvolvidas pela equipe para que possam criar cenários de testes de desempenho. O objetivo dessa atividade é que a equipe possa criar testes de desempenho e utilizá-los para melhorar os sistemas desenvolvidos.

Na segunda reunião da semana, foi ministrado um treinamento sobre a padronização da documentação de cenários de testes e defeitos. Durante a sessão, foram apresentados exemplos práticos de como documentar adequadamente os cenários de testes, os defeitos encontrados nas aplicações e as evidências das execuções dos testes. O objetivo dessa atividade é garantir que a equipe possa documentar de forma consistente e eficiente todas as informações relevantes relacionadas aos testes, o que ajudará na melhoria da qualidade dos sistemas desenvolvidos. Como resultado esperado, espera-se que a equipe possa aplicar as técnicas de documentação aprendidas e que todos possam adotar uma abordagem padronizada e consistente para documentar cenários de testes, defeitos e evidências de execução.

O plano de ação para a Equipe07, bem como sua descrição, pode ser observado por meio da Figura 28.

Figura 28: Plano de ação – Organização Gama – Equipe07



Fonte: O Autor

Como observado na Figura 28, o plano de ação da Equipe 07 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi dedicada a práticas relacionadas à pirâmide de testes e ao desenvolvimento orientado a testes. Na primeira reunião, a equipe recebeu um treinamento sobre a pirâmide de testes e como organizar e planejar os testes, de forma a automatizar a maioria deles e dispor na base da pirâmide, enquanto os testes menos importantes estarão no topo. O objetivo dessa atividade foi que a equipe pudesse planejar seus testes levando em conta a pirâmide de testes e seus princípios.

Na segunda reunião da semana, realizou-se um treinamento sobre o desenvolvimento orientado a testes, no qual a equipe foi orientada sobre os princípios de priorizar o teste unitário durante o desenvolvimento e como implementar os códigos utilizando essa abordagem. O treinamento teve como objetivo garantir que a equipe de desenvolvimento pudesse criar testes unitários de forma eficiente e entender a importância dessa prática como uma etapa fundamental para a finalização de histórias. Como resultado esperado dessa atividade, espera-se que a equipe seja capaz de criar testes unitários de forma consistente e incorporar o desenvolvimento orientado a testes em suas rotinas de trabalho.

A segunda semana foi composta por treinamentos sobre testes de regressão e testes de integração. Na primeira reunião da semana, foi abordado o treinamento sobre testes de regressão, onde a equipe foi orientada sobre a importância de realizar esses testes, especialmente os automatizados, com frequência, para garantir que a aplicação continue funcionando mesmo com alterações constantes no código-fonte. O resultado esperado dessa atividade é que a equipe execute regularmente os testes de regressão automatizados para garantir a estabilidade da aplicação.

A segunda reunião da semana teve como foco um treinamento prático em testes de integração. Durante a reunião, a equipe utilizou aplicações já desenvolvidas pela equipe de desenvolvimento para criar alguns testes de integração. Os testes envolveram validações automatizadas em relação ao status de retorno das aplicações, bem como a validação de valores de campos no banco de dados.

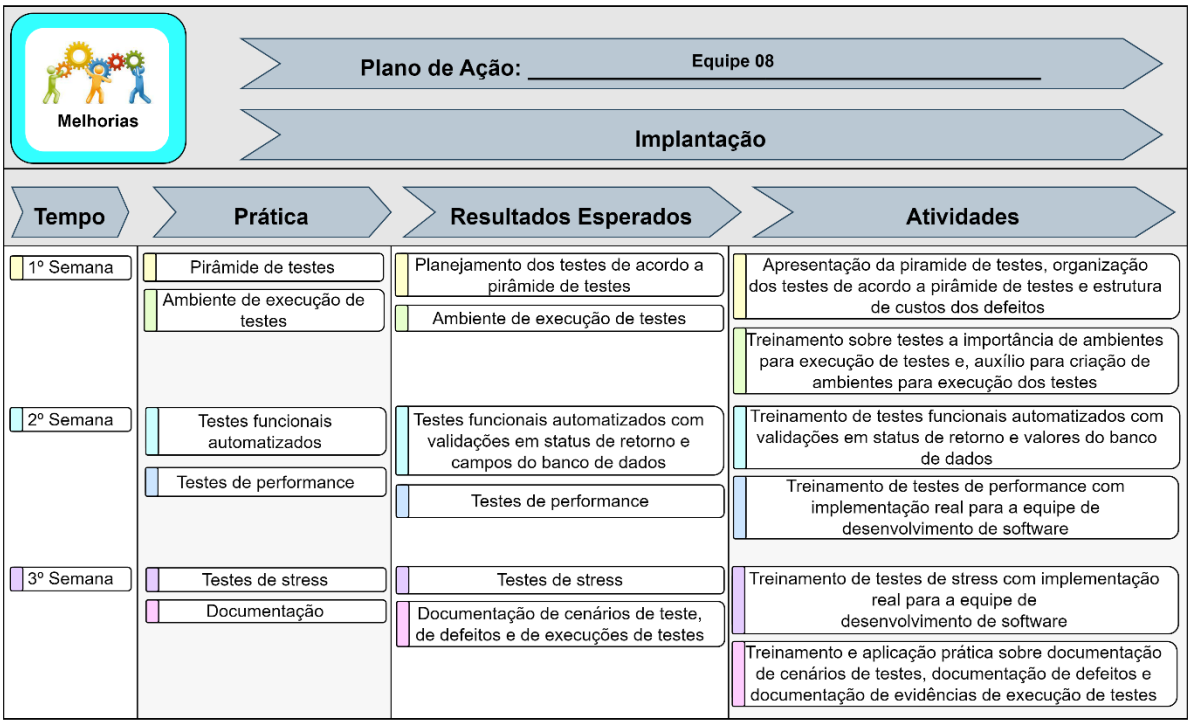
O objetivo desta atividade foi capacitar a equipe para criar e utilizar testes de integração com maior frequência e eficiência. Dessa forma, espera-se que a equipe seja capaz de aumentar a qualidade e a estabilidade das aplicações desenvolvidas, garantindo que elas funcionem de forma integrada e consistente.

Durante a terceira semana, foram realizadas reuniões com foco na documentação dos testes e na avaliação do desempenho dos sistemas. Na primeira reunião, a equipe recebeu um treinamento sobre a padronização da documentação dos cenários de teste e dos defeitos encontrados nas aplicações. O objetivo desta atividade foi capacitar a equipe para que pudesse documentar de forma adequada os testes realizados, assim como os defeitos encontrados e as evidências de execução dos testes. Espera-se que como resultado desta atividade, a equipe consiga registrar de forma clara e precisa todas as informações referentes aos testes, facilitando assim o processo de avaliação e melhoria dos sistemas desenvolvidos.

Na segunda reunião da semana, foi ministrado um treinamento prático sobre testes de desempenho, utilizando exemplos nas aplicações desenvolvidas pela equipe para criar cenários de testes de desempenho. O objetivo dessa atividade é capacitar a equipe para criar e executar testes de desempenho, bem como utilizar os resultados obtidos para identificar pontos de melhoria nos sistemas desenvolvidos. O resultado esperado é que a equipe possa realizar testes de desempenho de forma eficiente e efetiva, visando a melhoria contínua das aplicações.

O plano de ação para a Equipe 08, bem como sua descrição, pode ser observado por meio da Figura 29.

Figura 29: Plano de ação – Organização Gama – Equipe08



Fonte: O Autor

Como observado na Figura 29, o plano de ação da Equipe 08 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

Na primeira semana, realizamos atividades relacionadas às práticas de pirâmide de testes e ambiente de execução dos testes. Na primeira reunião, promovemos um treinamento sobre a pirâmide de testes e como organizar e planejar os testes de maneira eficiente, garantindo que a maioria dos testes seja automatizada e disposta na base da pirâmide. Dessa forma, a menor quantidade de testes estará no topo da pirâmide. O resultado esperado dessa atividade é que a equipe seja capaz de realizar o planejamento de testes considerando a pirâmide de testes.

Na segunda reunião da semana, a equipe participou de um treinamento sobre ambientes de execução de testes. Durante o treinamento, foram apresentados os benefícios de se ter ambientes específicos para a execução dos testes e, juntamente com a equipe, alguns ambientes foram criados. Como resultado esperado para esta atividade, está a capacitação do time para executar os testes em ambientes específicos, visando garantir que os testes sejam executados de forma consistente e padronizada, além de permitir a identificação de possíveis problemas relacionados ao ambiente de execução.

A segunda semana foi dedicada a treinamentos sobre testes funcionais automatizados e testes de desempenho. Na primeira reunião, foi realizado um treinamento prático sobre testes funcionais automatizados, no qual a equipe aprendeu a automatizar fluxos funcionais utilizando aplicações da organização e a integrá-los à esteira de integração e entrega contínua. O objetivo é que a equipe seja capaz de executar testes funcionais automatizados das aplicações por meio da esteira de integração CI/CD de forma eficiente.

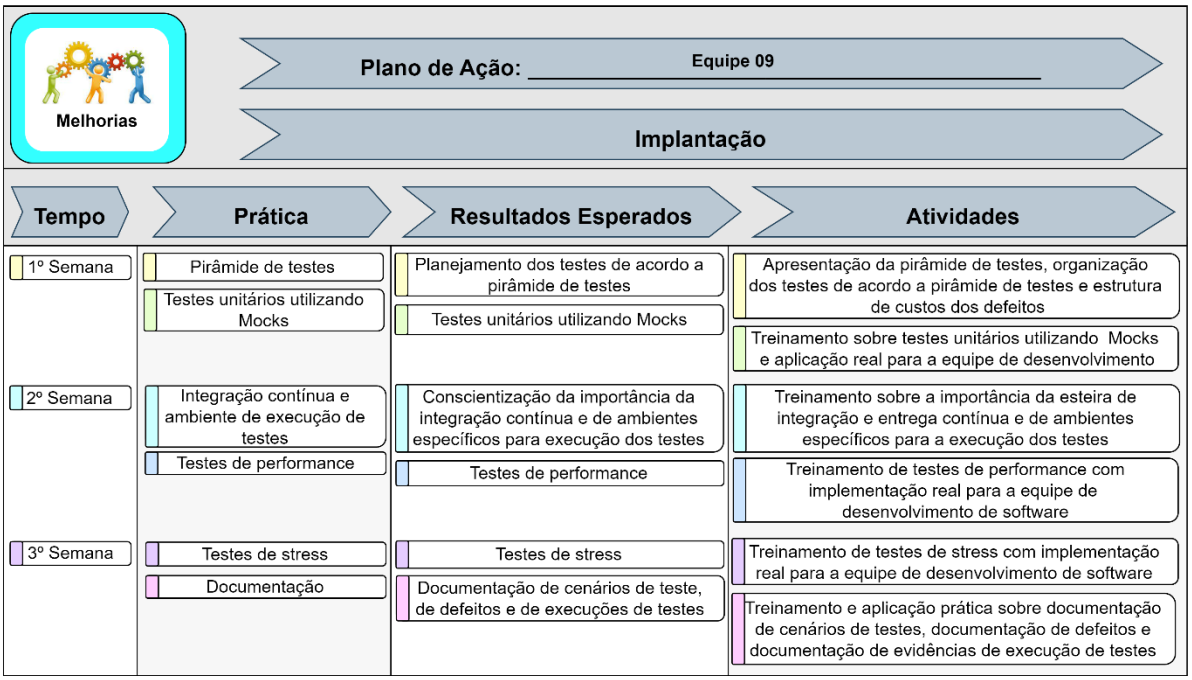
Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes de desempenho. Durante a atividade, foram apresentados exemplos práticos de como criar e executar testes de desempenho em aplicações desenvolvidas pela equipe, com o objetivo de identificar gargalos e problemas de desempenho. Espera-se que a equipe possa aplicar os conhecimentos adquiridos para realizar testes de desempenho de forma eficiente e utilizar os resultados para melhorar a qualidade das aplicações desenvolvidas.

A terceira semana foi composta por reuniões sobre testes de estresse e documentação de cenários de teste, documentação de defeitos e de evidências de execução dos testes. Na primeira reunião da semana, foi aplicado um treinamento prático sobre os testes de estresse, com exemplos práticos nas aplicações desenvolvidas pela equipe. O objetivo da atividade é capacitar a equipe para criar cenários de testes de estresse e utilizar os resultados para melhorar os sistemas desenvolvidos.

Na segunda reunião da semana, foi realizado um treinamento sobre padronização de documentação de cenários de teste e de defeitos. O objetivo dessa atividade foi capacitar a equipe para documentar seus cenários de teste, além de registrar os defeitos encontrados nas aplicações e as evidências das execuções dos testes de forma padronizada. O resultado esperado é que a equipe consiga realizar uma documentação clara e organizada, facilitando a comunicação entre os membros da equipe e melhorando a eficácia dos testes.

O plano de ação para a Equipe 09, bem como sua descrição, pode ser observado por meio da Figura 30.

Figura 30: Plano de ação – Organização Gama – Equipe09



Fonte: O Autor

Como observado na Figura 30, o plano de ação da Equipe 09 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana consistiu em atividades relacionadas às práticas de pirâmide de testes e testes unitários com o uso de *Mocks*. Na primeira reunião, a equipe recebeu um treinamento sobre a pirâmide de testes e como organizar e planejar os testes de modo que a maioria deles seja automatizada e disposta na base da pirâmide, enquanto a menor quantidade de testes deve estar no topo. O objetivo desta atividade é capacitar a equipe a planejar os testes levando em conta a pirâmide de testes.

Na segunda reunião da semana, a equipe recebeu treinamento sobre testes unitários utilizando *Mocks*. Durante o treinamento, foram abordados os princípios dos testes unitários e de *mocks* de objetos, com muita aplicação prática na criação de testes com *Mocks* em aplicações desenvolvidas pela equipe. O objetivo foi deixar exemplos de como utilizar essa técnica em cenários futuros.

O resultado esperado dessa atividade é que a equipe de desenvolvimento consiga criar testes unitários com a utilização de *Mocks* de forma eficiente quando necessário. Além disso, espera-se que a equipe considere os testes unitários como etapa fundamental para que uma

história possa ser finalizada/concluída, e entenda a importância de integrá-los ao processo de desenvolvimento.

Na segunda semana, foram realizados treinamentos sobre integração contínua, entrega contínua e ambientes de execução de testes, bem como testes de desempenho. Na primeira reunião da semana, a equipe recebeu um treinamento prático sobre a importância da integração contínua e da entrega contínua (CI/CD), utilizando aplicações da organização para demonstrar a importância dos fluxos automatizados de integração e entrega contínua. Foi destacado também a importância de seguir os critérios de qualidade definidos pela esteira e a má prática de burlar a esteira, publicando o código em ambiente produtivo sem atender essas regras. Além disso, foi discutido sobre a importância de se ter ambientes específicos para a execução dos testes. Como resultado esperado para esta atividade, a equipe deve estar ciente da importância de respeitar todas as etapas da esteira de publicação e utilizar os ambientes específicos para a execução dos testes, a fim de garantir a qualidade do software.

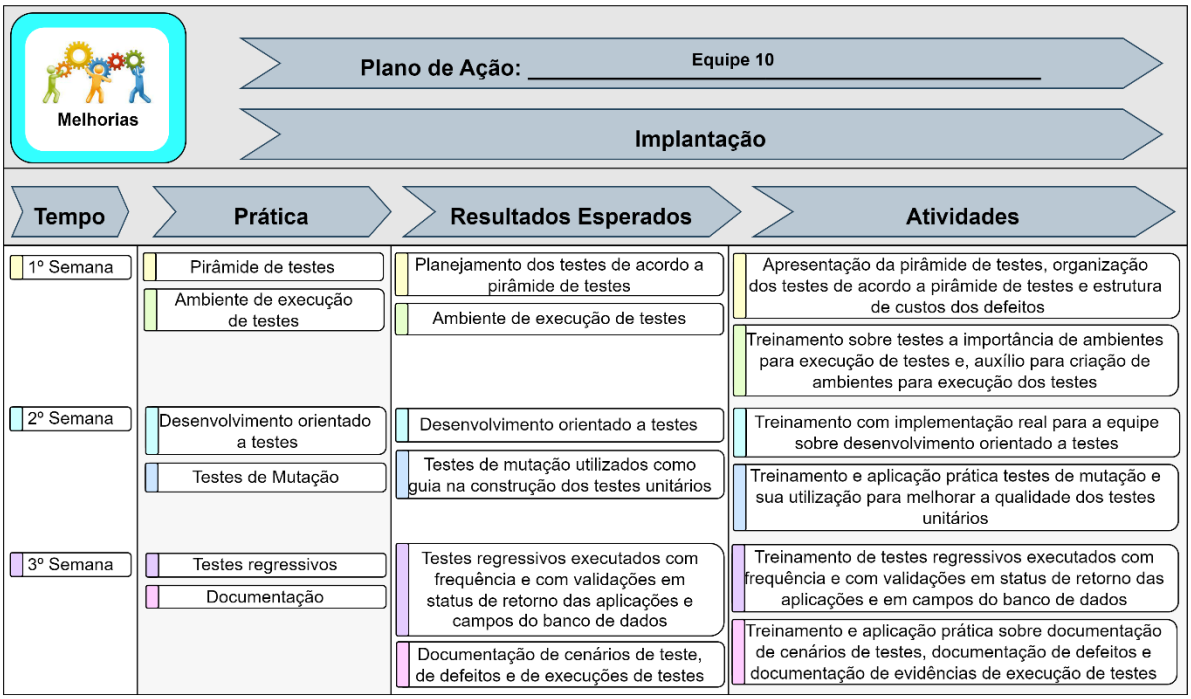
Na segunda reunião da semana, a equipe participou de um treinamento prático sobre testes de desempenho. Foram utilizadas aplicações desenvolvidas pela equipe para criar exemplos práticos de testes de desempenho. O objetivo da atividade foi capacitar a equipe a realizar testes de desempenho e utilizar os resultados obtidos como base para a melhoria das aplicações desenvolvidas. Como resultado esperado, espera-se que a equipe esteja apta a realizar testes de desempenho eficientes e aplicá-los regularmente no processo de desenvolvimento, visando aprimorar a qualidade dos sistemas produzidos.

Na primeira reunião da terceira semana, a equipe recebeu um treinamento prático sobre testes de estresse, onde foram apresentados exemplos práticos de como criar cenários de testes de estresse nas aplicações desenvolvidas pela equipe. O objetivo desta atividade é que a equipe aprenda a criar testes de estresse de forma eficiente e possa utilizar os resultados para melhorar os sistemas desenvolvidos.

Na segunda reunião da semana, realizou-se um treinamento sobre a padronização da documentação de cenários de teste e defeitos, com o objetivo de que a equipe consiga documentar de forma clara e padronizada seus cenários de teste, bem como os defeitos encontrados durante as atividades de teste e as evidências das execuções dos testes. Espera-se que, após a atividade, a equipe esteja apta a produzir documentações precisas e confiáveis, que possam ser utilizadas como base para a melhoria contínua dos sistemas desenvolvidos.

O plano de ação para a Equipe 10, bem como sua descrição, pode ser observado por meio da Figura 31.

Figura 31: Plano de ação – Organização Gama – Equipe10



Fonte: O Autor

Como observado na Figura 31, o plano de ação da Equipe 10 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi dedicada a práticas relacionadas à pirâmide de testes e ambiente de execução dos testes. Na primeira reunião, a equipe recebeu um treinamento sobre a pirâmide de testes, que mostrou como organizar e planejar os testes de forma que a maioria deles seja automatizada e esteja disposta na base da pirâmide, enquanto a menor quantidade de testes esteja no topo. O objetivo desta atividade é que a equipe consiga realizar o planejamento dos testes considerando a pirâmide de testes, visando aumentar a eficiência dos testes e garantir a qualidade das aplicações desenvolvidas.

Na segunda semana do treinamento, a equipe recebeu capacitação em desenvolvimento orientado a testes e testes de mutação. Na primeira reunião da semana, foi ministrado um treinamento sobre o desenvolvimento orientado a testes, no qual os princípios de priorizar os testes unitários foram abordados, juntamente com a implementação de códigos utilizando esse tipo de desenvolvimento. O objetivo dessa atividade foi capacitar a equipe de desenvolvimento para criar testes unitários eficientes em suas aplicações e considerar essa etapa fundamental para a finalização/conclusão de uma história.

Já na segunda reunião da semana, foi aplicado um treinamento sobre testes de mutação, no qual foram demonstrados os benefícios dessa prática para a qualidade do código, além de ter sido realizada uma demonstração prática sobre como implementá-los nas aplicações da equipe. Como resultado esperado para esta atividade, espera-se que a equipe consiga implementar testes de mutação em suas aplicações, visando a melhoria da qualidade do código e redução de possíveis defeitos.

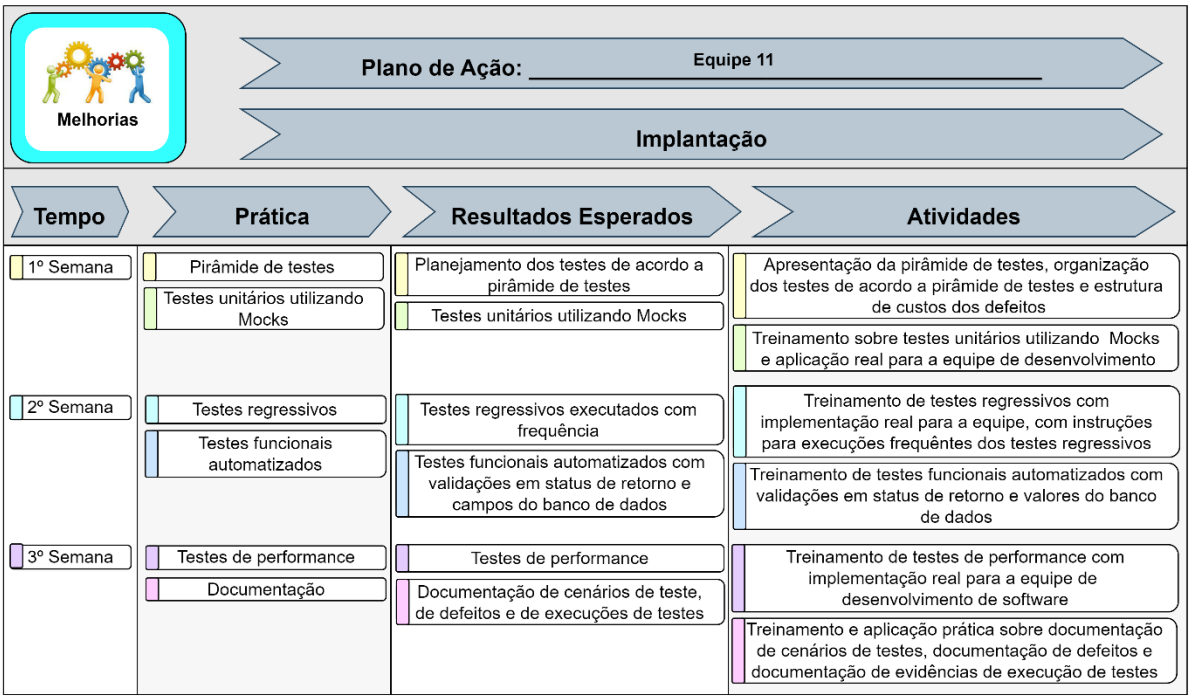
A segunda reunião da semana foi dedicada a um treinamento sobre testes de mutação. A equipe enfrentava dificuldades em identificar os cenários de testes unitários a serem criados e muitos testes unitários criados não possuíam validações eficientes. O objetivo da atividade era fornecer uma visão mais clara dos cenários de teste unitários que precisam ser criados e das validações necessárias para esses testes, a fim de melhorar o nível de cobertura de testes unitários da equipe. Como resultado esperado, a equipe deve conseguir criar testes unitários mais eficientes e, assim, melhorar a qualidade do código produzido.

A terceira semana de treinamento foi dedicada aos testes regressivos e à documentação de cenários de teste, defeitos e evidências de execução dos testes. Na primeira reunião, foi apresentado à equipe um treinamento sobre a importância dos testes regressivos e como eles devem ser realizados com frequência, especialmente os testes automatizados, para garantir que a aplicação continue funcionando mesmo após alterações constantes de código-fonte. Foi destacado a importância de se validar o status de retorno das aplicações e os valores do banco de dados. O objetivo desta atividade é que a equipe execute com frequência os testes automatizados regressivos, com pontos de validação nos status de retorno das aplicações e nos campos do banco de dados, garantindo assim que a aplicação permaneça estável e confiável.

Na segunda reunião da semana, foi realizado um treinamento sobre padronização na documentação de cenários de testes e defeitos. O objetivo dessa atividade é que a equipe seja capaz de documentar de forma padronizada seus cenários de teste, bem como os defeitos encontrados nas aplicações, além de registrar as evidências das execuções dos testes. O resultado esperado é que a equipe alcance uma maior eficiência na documentação dos processos de teste, facilitando a comunicação e o trabalho colaborativo.

O plano de ação para a Equipe 11, bem como sua descrição, pode ser observado por meio da Figura 32.

Figura 32: Plano de ação – Organização Gama – Equipe11



Fonte: O Autor

Como observado na Figura 32, o plano de ação da Equipe 11 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi focada em práticas relacionadas à pirâmide de testes e testes unitários utilizando *Mocks*. Na primeira reunião, aplicou-se um treinamento sobre a pirâmide de testes e como planejar os testes de forma eficiente, a fim de que a maioria dos testes sejam automatizados e localizados na base da pirâmide, enquanto a menor quantidade de testes esteja no topo. O objetivo dessa atividade é capacitar a equipe para realizar o planejamento dos testes de acordo com a pirâmide de testes e garantir que os testes sejam eficientes e bem distribuídos.

Na segunda reunião da semana, foi aplicado um treinamento prático sobre testes unitários utilizando *Mocks*, com o objetivo de orientar a equipe sobre os princípios desses tipos de testes e como utilizá-los de forma eficiente. Durante o treinamento, foram criados exemplos de testes unitários com *Mocks* em aplicações desenvolvidas pela equipe, para que eles pudessem ter uma noção prática de como utilizar essas técnicas em cenários futuros. O resultado esperado para essa atividade é que a equipe de desenvolvimento seja capaz de criar testes unitários com a utilização de *Mocks* de forma eficiente, sempre que necessário, e considere esses testes como uma etapa fundamental para a conclusão das histórias.

A segunda semana foi dedicada aos treinamentos sobre testes regressivos e testes funcionais automatizados. Durante a primeira reunião da semana, a equipe foi orientada sobre a importância de realizar testes regressivos com frequência, especialmente testes automatizados, para garantir que a aplicação continue funcionando mesmo com constantes alterações no código-fonte. O objetivo dessa atividade é que a equipe compreenda a importância dos testes regressivos e execute-os com frequência, garantindo assim a estabilidade da aplicação.

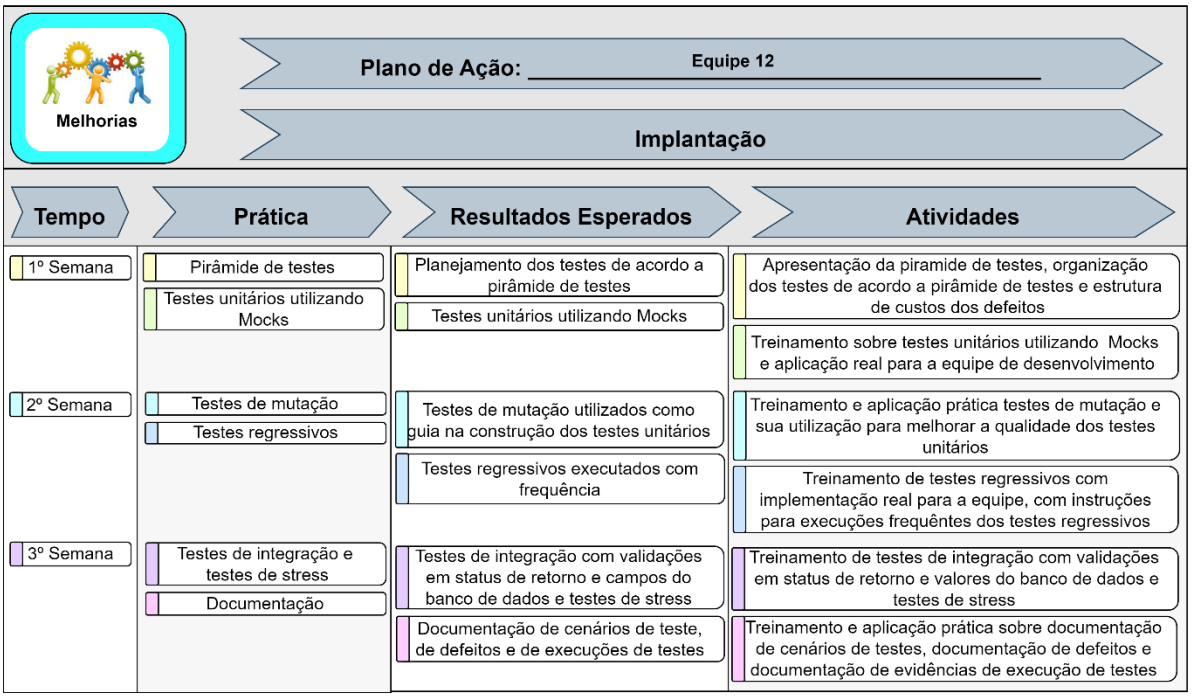
Na segunda reunião da semana, a equipe recebeu treinamento sobre testes funcionais automatizados, com foco na utilização de pontos de validação no banco de dados e no resultado de retorno das aplicações. Além disso, a reunião foi destinada a esclarecer dúvidas sobre cenários que a equipe possa ter encontrado dificuldades em automatizar. O objetivo dessa atividade é que a equipe seja capaz de realizar testes funcionais automatizados das aplicações, utilizando pontos de validação como o status de retorno das aplicações e valores do banco de dados. Dessa forma, espera-se aumentar a eficiência na validação das funcionalidades das aplicações desenvolvidas.

A terceira semana foi composta por reuniões dedicadas aos testes de desempenho e à documentação de cenários de teste, defeitos e evidências de execução dos testes. Na primeira reunião da semana, foi realizado um treinamento prático sobre testes de desempenho, com exemplos aplicados nas aplicações desenvolvidas pela equipe. O objetivo dessa atividade foi capacitar a equipe a criar cenários de teste de desempenho e utilizar os resultados obtidos para aprimorar os sistemas desenvolvidos. Como resultado esperado, espera-se que a equipe esteja apta a criar testes de desempenho de forma eficiente e a utilizá-los para a melhoria contínua dos sistemas.

Na segunda reunião da semana, a equipe participou de um treinamento para aprimorar a padronização da documentação de cenários de testes e defeitos. Durante o treinamento, foram apresentados exemplos de documentação de casos de testes, registros de defeitos e evidências de execução dos testes, e discutidos os melhores métodos para documentação. O objetivo desta atividade é que a equipe seja capaz de documentar seus cenários de teste e defeitos encontrados de forma clara e padronizada, além de coletar e registrar evidências das execuções dos testes para fins de rastreabilidade e análise.

O plano de ação para a Equipe 12, bem como sua descrição, pode ser observado por meio da Figura 33.

Figura 33: Plano de ação – Organização Gama – Equipe12



Fonte: O Autor

Como observado na Figura 33, o plano de ação da Equipe 12 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana foi dedicada a atividades sobre as práticas de pirâmide de testes e testes unitários com a utilização de *Mocks*. Na primeira reunião, aplicou-se um treinamento prático sobre a pirâmide de testes e como planejar a automação dos testes de forma que a maior quantidade de testes esteja na base da pirâmide e a menor quantidade no topo. O objetivo desta atividade é que a equipe possa planejar os testes levando em consideração a pirâmide de testes e assim garantir uma maior eficiência na execução dos testes.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes unitários utilizando *Mocks*, no qual foram abordados os princípios fundamentais desses testes e a utilização de *mocks* de objetos. O treinamento teve um enfoque prático e a equipe teve a oportunidade de criar alguns testes unitários com *Mocks* nas aplicações desenvolvidas, para que pudessem servir como exemplos em cenários futuros. O objetivo desta atividade foi capacitar a equipe para que consiga criar testes unitários eficientes com o uso de *Mocks*, quando necessário, e compreender que essa etapa é fundamental para a conclusão de uma história.

A segunda semana foi dedicada aos treinamentos sobre testes de mutação e testes regressivos. Na primeira reunião da semana, a equipe recebeu um treinamento sobre testes de

mutação, pois estava enfrentando dificuldades para identificar os cenários de testes unitários a serem criados. Além disso, muitos testes unitários criados pela equipe não possuíam validações eficientes. Durante o treinamento, a equipe aprendeu a identificar os cenários de teste unitários necessários e as validações adequadas para cada um deles. O resultado esperado desta atividade é que a equipe seja capaz de criar testes unitários eficientes, aumentando assim a cobertura de testes unitários do projeto.

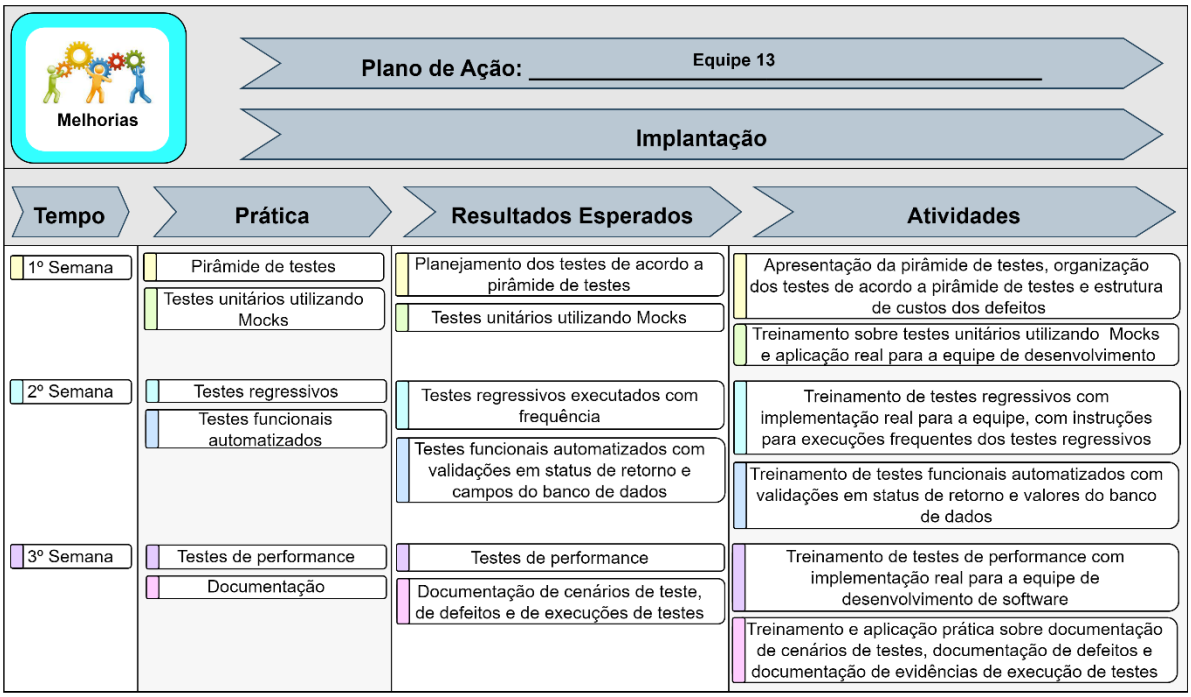
A segunda semana foi dedicada a treinamentos sobre testes regressivos e de integração. Na primeira reunião, foi enfatizada a importância de realizar os testes automatizados com frequência, principalmente os testes regressivos, para garantir que a aplicação continue funcionando mesmo com constantes alterações de código-fonte. A equipe também aprendeu como identificar e criar testes de integração eficientes, que validem a comunicação e integração entre diferentes módulos e componentes da aplicação. O objetivo desta atividade é que a equipe seja capaz de realizar testes automatizados de regressão e integração com eficiência e frequência, garantindo assim a qualidade contínua do produto.

A terceira semana foi composta por treinamentos sobre testes de integração, testes de estresse e documentação de cenários de testes, defeitos e evidências de execuções de testes. Na primeira reunião da semana, foi realizado um treinamento prático de testes de integração utilizando aplicações já desenvolvidas pela equipe. Foram criados alguns testes de integração com validações automatizadas de status de retorno das aplicações e valores de campos no banco de dados. Além disso, um treinamento prático sobre os testes de estresse foi realizado, com exemplos nas aplicações da equipe para criar cenários de testes de estresse. O objetivo desta atividade é que a equipe possa utilizar os testes de integração com frequência e criar testes de estresse, utilizando os resultados para melhorar os sistemas desenvolvidos. Também foi abordada a documentação de cenários de testes, defeitos e evidências de execuções de testes como parte importante do processo de testes para garantir a rastreabilidade e a efetividade dos testes realizados.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre padronização de documentação de cenários de testes e defeitos. Durante a sessão, a equipe aprendeu sobre a importância de documentar adequadamente os cenários de teste, os defeitos encontrados nas aplicações e as evidências das execuções dos testes. O treinamento teve como objetivo garantir que a equipe adote uma abordagem padronizada na documentação dos resultados dos testes, a fim de garantir a rastreabilidade e a facilidade na comunicação entre os membros da equipe. Como resultado esperado desta atividade, espera-se que a equipe esteja apta a documentar seus cenários de teste, defeitos e evidências de execuções de forma clara, objetiva e padronizada.

O plano de ação para a Equipe 13, bem como sua descrição, pode ser observado por meio da Figura 34.

Figura 34: Plano de ação – Organização Gama – Equipe13



Fonte: O Autor

Como observado na Figura 34, o plano de ação da Equipe 13 é composto por atividades realizadas ao longo de três semanas de trabalho, por meio de duas reuniões de duração de uma hora e meia, com forte participação da equipe de desenvolvimento de software.

A primeira semana teve como foco as práticas de pirâmide de testes e testes unitários utilizando *Mocks*. Na primeira reunião da semana, foi aplicado um treinamento para orientar a equipe sobre a pirâmide de testes e a importância de organizar e planejar os testes de forma que a maioria seja automatizada e disposta na base da pirâmide, e que a menor quantidade de testes esteja no topo da pirâmide. O objetivo da atividade foi garantir que a equipe fosse capaz de realizar um planejamento eficiente dos testes, considerando a pirâmide de testes. O resultado esperado é que a equipe consiga criar uma estratégia eficiente de testes, com a maioria dos testes automatizados e bem-organizados de acordo com a pirâmide de testes.

Na segunda reunião da semana, foi ministrado um treinamento sobre testes unitários utilizando *Mocks*, onde a equipe recebeu orientações sobre os princípios dos testes unitários e de *mocks* de objetos. Durante o treinamento, a equipe praticou criando testes unitários com *Mocks* nas aplicações desenvolvidas, a fim de deixar exemplos para futuros cenários. O

resultado esperado é que a equipe de desenvolvimento seja capaz de criar testes unitários eficientes com o uso de *Mocks* quando necessário, além de considerar essa etapa fundamental para a finalização de histórias.

Na segunda semana, além de treinamentos sobre testes regressivos, também foram abordados testes funcionais automatizados. Na primeira reunião da semana, foi aplicado um treinamento prático sobre testes regressivos, no qual a equipe foi orientada a realizar testes automatizados com frequência para garantir que a aplicação continue funcionando mesmo com constantes alterações de código-fonte. Além disso, foram apresentados exemplos práticos de testes funcionais automatizados, nos quais a equipe teve a oportunidade de aprender sobre a criação de cenários de teste que simulam a interação do usuário com a aplicação. Para esta atividade, o resultado esperado é que a equipe seja capaz de realizar testes automatizados de forma mais eficiente e com maior cobertura, tanto em relação aos testes regressivos quanto aos testes funcionais automatizados.

Na segunda reunião da semana, a equipe recebeu um treinamento sobre testes funcionais automatizados, utilizando pontos de validação no banco de dados e no resultado de retorno das aplicações. Durante a reunião, foram abordados cenários práticos e foram esclarecidas dúvidas sobre os casos em que a equipe não conseguiu automatizar os testes.

O resultado esperado desta atividade é que a equipe possa realizar testes funcionais automatizados de forma eficiente, utilizando pontos de validação para verificar o status de retorno das aplicações e os valores do banco de dados. Isso permitirá garantir que as aplicações estejam funcionando corretamente e que novas funcionalidades possam ser adicionadas sem comprometer a qualidade do sistema.

A terceira semana do treinamento teve como foco os testes de desempenho e a padronização da documentação de cenários de testes, defeitos e evidências de execução de testes. Na primeira reunião da semana, foi ministrado um treinamento prático sobre testes de desempenho, utilizando exemplos práticos nas aplicações desenvolvidas pela equipe para ajudar a criar cenários de teste de desempenho. O resultado esperado desta atividade é que a equipe consiga criar testes de desempenho eficazes e utilizar os resultados para melhorar os sistemas desenvolvidos.

Na segunda reunião da semana, a equipe participou de um treinamento sobre padronização da documentação de cenários de testes e defeitos, visando aprimorar a rastreabilidade das atividades de teste. Durante o treinamento, foram apresentados exemplos de documentos padronizados para cenários de teste e defeitos, bem como técnicas para documentar as evidências das execuções dos testes de forma clara e concisa. Como resultado esperado desta

atividade, espera-se que a equipe consiga documentar de forma padronizada seus cenários de teste, bem como registrar os defeitos encontrados e as evidências das execuções dos testes, permitindo uma melhor gestão e análise dos resultados dos testes.

7.5. Etapa Aferições

A etapa de aferição teve a duração de 21 horas, englobando as 13 equipes de desenvolvimento de software. Durante essa etapa, foi realizado o mapeamento do cenário atual, com duração de uma hora em cada reunião com as equipes de desenvolvimento de software, utilizado como comparativo antes e depois da análise do cenário de teste de software. Além disso, essa etapa incluiu a análise dos dados e resultados obtidos, com duração média de 30 minutos por equipe de desenvolvimento de software, e uma importante fase de comunicação dos resultados, com reuniões entre as equipes de desenvolvimento e o gestor responsável para divulgação dos resultados de melhoria do processo de teste de software. Também houve uma reunião geral com os gestores e superintendente para apresentação dos resultados de melhoria de teste de software.

A primeira atividade desta tarefa consistiu na identificação do cenário alcançado após a execução do plano de ação. Para isso, foi realizada uma reunião de uma hora com a equipe de testes de software, na qual houve uma análise conjunta das práticas de teste de software e sua frequência de utilização. A reunião contou com a participação ativa dos membros das equipes, que forneceram feedbacks muito positivos sobre as melhorias realizadas e os aprendizados adquiridos ao longo do processo.

A segunda atividade desta etapa consistiu na análise dos resultados obtidos após a execução do plano de ação, na qual foram comparados os dados coletados do cenário anterior à implementação das melhorias em relação ao cenário após a implementação. Além disso, foi elaborado um material de apresentação para os gestores e líderes, com o objetivo de mostrar a evolução da equipe de teste de software em relação à maturidade e à melhoria do processo de teste de software.

Os resultados da análise do cenário alcançado para as equipes de 01 a 13 encontram-se registrados na Tabela 9.

Tabela 9: Cenário Alcançado – Organização Gama – Equipes de 01 a 13

Práticas em Teste de Software	ANTES DA IMPLEMENTAÇÃO DO ROMTES													DEPOIS DA IMPLEMENTAÇÃO DO ROMTES											
	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3	4	5	6	7	8	9	10	11	12	13
Desenvolvimento orientado a testes	F	N	N	S	F	N	N	N	N	R	F	N	F	S ↑	N =	S ↑	S =	S ↑	R ↑	N =	R ↑	R ↑	F ↑	F =	N =	S ↑
Testes unitários	F	F	F	F	F	F	F	F	F	F	F	F	F	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =
Testes unitários utilizam mocks	F	F	N	S	F	R	S	S	F	S	R	R	F	F =	S ↑	S ↑	S =	S ↑	S ↑	S =	S =	F =	S =	F ↑	R =	S ↑
São aplicados testes de mutação	N	N	N	F	F	S	N	S	R	R	N	N	R	N =	N =	F ↑	F =	S ↑	R ↓	N =	F ↓	R =	F ↑	N =	R ↑	R =
Testes unitários são necessários para conclusão da história	S	S	S	S	F	F	S	S	F	S	R	R	S	S =	S =	S =	S =	S ↑	S ↑	S =	S =	S ↑	S =	F ↑	F ↑	S =
Análise estática de código	S	S	S	S	F	F	N	S	R	R	N	N	F	S =	S =	S =	S =	S ↑	F =	S ↑	S =	R =	S ↑	F ↑	S ↑	S ↑
Testes regressivos	F	S	S	S	F	F	N	N	R	R	R	N	R	S ↑	F ↓	F ↓	S =	S ↑	F =	S ↑	N =	R =	F ↑	F ↑	F ↑	F ↑
Há um ambiente específico para execução dos testes	S	S	S	S	F	F	S	R	R	N	S	S	S	S =	S =	S =	S =	S ↑	F =	S =	S ↑	F ↑	S ↑	S =	S =	S =
O status de retorno das aplicações são validados	F	S	S	S	F	F	S	N	F	S	R	R	S	F =	S =	S =	S =	S ↑	F =	S =	N =	F =	S =	F ↑	F ↑	S =
Os campos no banco de dados são validados	N	N	F	S	S	N	N	N	N	N	R	R	F	S ↑	N =	S ↑	S =	S =	F ↑	N =	N =	N =	S ↑	R =	F ↑	F =
Testes funcionais automatizados	S	S	S	S	S	F	S	S	F	S	S	S	S	F ↓	N ↓	F ↓	S =	S =	F =	S =	N ↓	R ↓	F ↓	F ↓	F ↓	S =
Testes automatizados estão integrados na esteira CI/CD	S	S	S	S	S	S	S	S	S	S	S	S	S	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =
Erro nos testes acarreta falha na esteira CI/CD	S	S	S	S	S	S	S	S	S	S	S	S	S	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =
Ocorre desativação de testes(bypass) na esteira CI/CD	N	N	N	N	R	R	N	N	R	N	N	N	R	N =	N =	R ↑	S ↑	N ↓	N ↓	N =	N =	N ↓	N =	N =	N =	S ↑
Testes funcionais são necessários para conclusão da história	F	F	F	F	F	F	F	F	F	F	F	F	F	R ↓	S ↑	S ↑	S ↑	S ↑	F =	S ↑	S ↑	R ↓	S ↑	S ↑	S ↑	S ↑
Testes de integração com banco de dados/outras aplicações	F	S	N	S	R	S	S	S	R	S	F	R	S	F =	S =	R ↑	S =	S ↑	F ↓	F ↓	S =	R =	S =	N ↓	R =	S =
Testes de performance	N	N	R	N	F	F	N	N	N	S	N	R	F	R ↑	N =	R =	F ↑	S ↑	R ↓	N =	R ↑	R ↑	S =	R ↑	R =	S ↑
Testes de stress	N	N	R	N	F	F	N	N	N	S	N	N	S	N =	N =	R =	N =	S ↑	R ↓	N =	R ↑	N =	S =	N =	R ↑	S =
Testes não funcionais são necessários para as entregas	N	N	N	N	N	N	N	N	N	N	N	N	N	F ↑	N =	S ↑	S ↑	F ↑	F ↑	S ↑	S ↑	F ↑	S ↑	S ↑	R ↑	S ↑
Testes exploratórios	N	S	R	S	R	S	N	S	N	R	N	N	R	S ↑	S =	S ↑	S =	S ↑	S =	S ↑	S =	S ↑	N ↓	N =	S ↑	S ↑
São escritos cenários/planos de teste	R	N	N	S	F	R	N	R	N	S	N	N	R	F ↑	S ↑	R ↑	S =	S ↑	F ↑	N =	F ↑	R ↑	S =	S ↑	F ↑	S ↑
Há padrões na escrita dos cenários/planos de teste	F	S	N	S	S	R	R	N	N	R	S	N	S	F =	S =	F ↑	S =	S =	F ↑	S ↑	F ↑	S ↑	F ↑	S =	R ↑	S =
Revisão por pares dos cenários/plano de teste	N	N	N	N	N	N	N	N	N	N	N	N	N	F ↑	N =	N =	S ↑	N =	F ↑	N =	N =	N =	N =	N =	N =	N =
São evidenciadas as execuções de teste	N	N	R	F	F	R	N	R	R	N	N	S	R	S ↑	S ↑	R =	S ↑	S ↑	F ↑	N =	F ↑	S ↑	S ↑	S ↑	S =	F ↑
Ocorre a documentação dos defeitos encontrados	N	N	N	S	S	N	S	N	N	S	N	N	N	S ↑	S ↑	S ↑	S =	S =	N =	S =	F ↑	S ↑	S =	F ↑	S ↑	N =
Resultados dos testes são usados para melhoria do produto	N	N	N	N	N	S	N	N	N	N	N	N	S	N =	S ↑	S ↑	N =	S ↑	S =	N =	S ↑	N =	N =	N =	N =	S =
São estimadas horas para planejamento/execução dos testes	S	S	N	S	S	S	S	N	S	S	S	N	S	S =	S =	N =	S =	S =	S =	S =	N =	S =	S =	S =	S ↑	S =
Os testes são baseados em requisitos de negócios	N	S	S	S	S	S	S	R	N	S	S	N	S	N =	S =	S =	S =	S =	S =	S =	S ↑	N =	N ↓	S =	S ↑	S =
Gestão de ciclo de vida de aplicação(Exemplo: SilkCentral, AzureDevOps, Jenkin	S	S	S	S	S	S	S	S	S	S	S	S	S	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =	S =
Execução dos testes são realizados com frequência	F	F	F	F	F	F	F	F	F	F	F	F	F	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =	F =
São planejados quantidade de testes por nível da pirâmide de testes	R	S	N	S	S	S	N	N	N	S	S	S	S	S ↑	S =	S ↑	N ↓	N ↓	S =	S ↑	N =	N =	S =	S =	S =	S =
Aonde se encontra a maior quantidade de testes da Equipe?	TU	TM	TU	FA	TU	FA	TU	TU	TU	TU	FA	TU	FA	TU	TM	TU	FA	TU	TU	TU	TU	TU	TU	FA	TU	TU
Nível de maturidade - TMMi	2	2	2	2	2	2	2	1	2	1	2	1	2	3	2	2	3	3	3	2	2	2	2	2	2	2

"S" significa Sempre, "F" significa Frequentemente, "R" significa Raramente e "N" significa Nunca "TU" significa Testes de Unidade, "TM" significa Testes Manuais e "FA" significa Testes Funcionais Automatizados

Fonte: O Autor

Ao analisarmos a Tabela 09, em relação a evolução que cada equipe de desenvolvimento de 01 a 13 apresentou em relação ao aumento na frequência de uso das práticas de teste de software e, no nível de maturidade da equipe em testes de software, com base no TMMi.

Em relação a Equipe01, ela teve melhorias significativas tanto na frequência de aplicação das práticas de teste de software quanto no nível de maturidade da equipe. Foi possível observar uma maior utilização de práticas como o desenvolvimento orientado a testes, a execução frequente de testes regressivos com validação em status de retorno e pontos de validação em campos do banco de dados. Tais melhorias contribuíram para um cenário de teste de software mais eficiente e confiável para a equipe.

Com base na comparação do antes e depois da aplicação do plano de ação, podemos observar uma evolução positiva em diversos aspectos das práticas de teste de software e sua frequência de utilização.

Em relação ao desenvolvimento orientado a testes, a equipe passou de frequentemente para sempre, o que indica um comprometimento maior com a qualidade do software desde o início do processo de desenvolvimento. Os testes unitários continuam sendo frequentes e foram mantidos o uso de *mocks*, o que é uma boa prática para testes unitários.

No entanto, a equipe ainda não está aplicando testes de mutação, o que poderia aumentar ainda mais a eficácia dos testes unitários. Outra boa prática que foi adotada pela equipe foi a realização de testes regressivos sempre, o que ajuda a garantir que as mudanças realizadas no software não afetem o seu comportamento anterior, além disso, a equipe passou a validar os campos no banco de dados sempre, o que pode contribuir para uma maior confiabilidade dos dados armazenados.

A equipe também começou a adotar testes funcionais automatizados frequentemente, embora ainda seja necessário avançar para torná-los sempre e, os testes não funcionais também passaram a ser necessários para a conclusão das histórias frequentemente, o que indica uma preocupação maior com a qualidade do software além da sua funcionalidade.

Os testes exploratórios também passaram a ser realizados sempre, o que é uma boa prática para encontrar defeitos que poderiam passar despercebidos pelos testes automatizados. Por fim, a equipe passou a realizar revisão por pares dos cenários/planos de teste frequentemente, o que pode ajudar a identificar possíveis falhas ou lacunas nos testes.

Com relação ao nível de maturidade em teste de software, a equipe conseguiu evoluir do nível Gerenciado para o nível Definido, o que indica que ela está mais organizada e estruturada em relação às práticas de teste de software.

Em resumo, a equipe teve uma evolução positiva em diversas práticas de teste de software, o que pode contribuir para a melhoria da qualidade do software produzido. No entanto, ainda existem algumas áreas que precisam ser trabalhadas para que a equipe alcance um nível de excelência em teste de software.

De acordo com a pirâmide de testes, a Equipe 01 foi capaz de adotar a quantidade adequada de testes, com um esforço maior nos testes unitários em relação aos testes de desempenho e funcionais automatizados. Além disso, houve um aumento nos testes de desempenho e exploratórios, bem como na documentação dos cenários de teste, defeitos encontrados e evidências de execução dos testes. A equipe também adotou o processo de revisão por pares não apenas para cenários de teste, mas também para a revisão de código-fonte da aplicação.

A equipe demonstrou uma evolução significativa em relação ao nível de maturidade em teste de software, avaliado pelo TMMI. Passou do nível 2 - Gerenciado para o nível 3 - Definido, uma vez que agora possui política e estratégia de testes, planejamento, monitoramento e controle dos testes, execução e ambientes implementados, práticas referentes ao nível I de maturidade e, adotou práticas relacionadas à organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais, características do nível II de maturidade de testes, conforme o TMMI.

Como uma organização de grande porte, a cobertura de código por testes unitários é uma métrica fundamental. Ela reflete a média da cobertura de testes unitários das aplicações desenvolvidas pela equipe. Antes da implementação do RoMTeS, a cobertura de testes unitários estava em torno de 50%. Após as três semanas de uso do RoMTeS, essa métrica subiu significativamente para 75%.

Em relação a Equipe 02, ela apresentou melhorias significativas tanto na frequência de aplicação das práticas de teste de software, quanto nas atividades necessárias para aumentar o nível de maturidade de teste da equipe. Na análise das práticas de teste, observou-se um aumento na frequência de uso de práticas como a utilização de *mocks* nos testes unitários quando necessário, estruturação adequada da quantidade de testes baseada na pirâmide de testes, documentação mais frequente dos cenários de teste, e uso dos resultados dos testes para aprimorar o produto.

A execução dos testes regressivos agora conta com validações mais rigorosas do status de retorno das aplicações e pontos de validação nos campos do banco de dados, assim como nos testes de integração. Além disso, a equipe também conseguiu aumentar a cobertura de testes

unitários, passando de uma média de 50% para 75%, um resultado muito significativo para uma organização grande como esta.

A Equipe 02 apresentou alguns pontos positivos em suas práticas de teste de software, como a realização frequente de testes unitários, a aplicação de análise estática de código e a validação do status de retorno das aplicações. No entanto, existem algumas áreas que precisam de melhorias, como a falta de desenvolvimento orientado a testes, a ausência de testes de mutação e testes não funcionais, além da baixa presença de testes automatizados e de planos de teste escritos.

Houve melhorias em relação a frequência de uso de práticas em teste de software. Antes, a equipe não utilizava desenvolvimento orientado a testes e nunca aplicava testes de mutação, testes funcionais automatizados e testes regressivos eram frequentes, mas nem sempre eram executados. Além disso, a equipe não documentava defeitos encontrados, nem evidenciava as execuções de teste e os resultados dos testes não eram utilizados para melhoria do produto.

Após as melhorias implementadas por meio do plano de ação, a equipe passou a aplicar testes regressivos frequentemente, os testes unitários são sempre utilizados e sempre utilizam *mocks*. A equipe também passou a escrever cenários/planos de teste, documentar defeitos encontrados e evidenciar as execuções de teste. Além disso, os resultados dos testes agora são utilizados para melhoria do produto.

Outras melhorias incluem a adoção de um ambiente específico para execução de testes, a integração de testes automatizados na esteira CI/CD e a realização de estimativas para planejamento e execução de testes. A equipe também passou a analisar os testes de acordo com a pirâmide de testes e a realizar testes de integração com banco de dados ou outras aplicações.

De acordo com a pirâmide de testes, a Equipe 02 teve dificuldades em estruturar a quantidade adequada de testes automatizados, com uma maior quantidade de esforços de testes ainda sendo dedicada aos testes manuais devido à grande quantidade já existente.

Ao avaliarmos o nível de maturidade em teste de software da Equipe 02, com base no TMMI, percebemos que ela permaneceu no nível 2-Gerenciado em maturidade de testes, mas com um aumento de práticas que contribuem para elevar seu nível de maturidade, incluindo a organização dos testes, que é uma prática relacionada ao nível 3-Definido do modelo TMMI. Apesar disso, é importante ressaltar que a equipe obteve grandes avanços na qualidade dos testes unitários, o que é fundamental para garantir a qualidade do software desenvolvido.

Considerando a métrica de cobertura de código por testes unitários, que representa a média de cobertura dos testes unitários nas aplicações desenvolvidas pela equipe, observamos que, antes da aplicação do RoMTeS, essa métrica era de 50%. Após as três semanas de

utilização da ferramenta, foi possível alcançar uma cobertura de 55% de nível de cobertura de código por testes unitários, indicando um avanço no nível de qualidade dos testes unitários da equipe.

Em relação a Equipe 03, ela apresentou melhorias tanto em relação a frequência na aplicação das práticas de teste de software como em atividades necessárias para aumentar o nível de maturidade de teste de software da equipe. Ao analisarmos as práticas de teste de software da equipe, é notável a melhoria na frequência de uso de práticas como o desenvolvimento orientado a testes, a utilização de *Mocks* nos testes unitários, o uso de testes de mutação e a validação dos testes regressivos e de integração por meio de validações no status de retorno das aplicações e valores do banco de dados. A equipe também passou a incluir os testes não funcionais como parte importante para conclusão de histórias e a prática de testes exploratórios. Além disso, houve uma melhoria na documentação dos cenários de teste, evidências de execução dos testes e defeitos encontrados. A equipe também estruturou os testes de acordo com os níveis da pirâmide de testes, priorizando a maior quantidade de testes no nível base da pirâmide, ou seja, nos testes de unidade. Tais melhorias contribuíram significativamente para aumentar o nível de qualidade dos testes de software realizados pela equipe.

De forma geral, a Equipe 03 implementou várias melhorias em suas práticas de teste de software. Em particular, houve um aumento significativo na frequência de uso de práticas como desenvolvimento orientado a testes, utilização de *mocks* em testes unitários, testes de mutação e testes funcionais automatizados. Além disso, a equipe agora realiza testes regressivos com mais frequência e validação de campos no banco de dados.

Outra melhoria significativa é que a equipe agora integra testes automatizados na esteira de CI/CD e os erros nos testes agora acarretam falhas na esteira. Além disso, a equipe agora realiza testes funcionais para concluir histórias, o que mostra uma maior ênfase na qualidade do produto.

A Equipe 03 também melhorou suas práticas de documentação, evidenciando as execuções de teste e documentando os defeitos encontrados. A equipe agora utiliza resultados de testes para melhorar o produto e implementou uma estratégia de planejamento de testes por níveis da pirâmide de testes.

Considerando a estruturação dos testes de acordo com a pirâmide de testes, a equipe adotou orientações para utilização de maior quantidade de testes no nível base da pirâmide de testes, ou seja, nos testes de unidade, o que permitiu uma maior cobertura de código por testes unitários. Com isso, houve um avanço significativo na qualidade dos testes unitários, o que contribui para o aumento da confiabilidade das aplicações desenvolvidas.

Apesar de permanecer no mesmo nível de maturidade (2-Gerenciado) no modelo TMMI, a equipe demonstrou evolução nas práticas de teste de software, incorporando novas técnicas e ferramentas que contribuem para um processo de testes mais eficiente e eficaz. Com isso, a equipe está mais preparada para lidar com os desafios do desenvolvimento de software, garantindo maior qualidade e confiabilidade dos produtos entregues.

Em relação à métrica de cobertura de código por testes unitários, que mede a proporção de código coberto pelos testes unitários das aplicações desenvolvidas pela equipe, houve um avanço significativo após a aplicação do RoMTeS. No início do processo, a cobertura era de apenas 40%, mas ao final das três semanas, a equipe alcançou um impressionante nível de cobertura de 94%, demonstrando um excelente progresso na qualidade dos testes unitários.

Em relação a Equipe 04, observa-se que houve uma maior utilização de práticas como os testes funcionais e não funcionais para a conclusão das histórias/entregas, bem como a criação de testes de desempenho e revisão por pares tanto do código-fonte das aplicações como dos cenários de teste. Houve também um aumento na adoção da prática de documentação dos cenários de teste, documentação dos defeitos encontrados e das evidências das execuções de testes.

A equipe de desenvolvimento selecionou a opção de nunca para o planejamento dos testes serem realizados de acordo a pirâmide de testes apesar de, ter iniciado esta prática com a aplicação do RoMTeS devido a maior quantidade dos testes da equipe serem testes funcionais automatizados, porém, a equipe aprendeu a sempre utilizar os testes unitários como parte necessária para as entregas de software.

Sobre o resultado obtido com a implementação do plano de ação, a Equipe 04 apresentou melhorias significativas em relação à frequência na aplicação das práticas de teste de software e ao nível de maturidade em teste de software. Foi observado um aumento na adoção de práticas como os testes funcionais e não funcionais, criação de testes de desempenho e revisão por pares, tanto de código-fonte das aplicações como dos cenários de teste. Além disso, houve aumento na documentação dos cenários de teste, dos defeitos encontrados e das evidências das execuções de testes.

A Equipe 04 demonstrou uma grande evolução em suas práticas de teste de software. Em comparação a antes da aplicação do plano de ação, a equipe manteve um alto nível de frequência no uso de práticas como desenvolvimento orientado a testes, testes unitários, testes unitários utilizando *mocks*, aplicação de testes de mutação, testes regressivos, análise estática de código, uso de ambiente específico para execução de testes, validação de retorno de aplicações e campos no banco de dados, testes funcionais automatizados, testes automatizados

integrados na esteira CI/CD, desativação de testes nunca ocorrendo e testes exploratórios sempre sendo realizados.

Além disso, a equipe melhorou significativamente em áreas como testes não funcionais, testes de performance e processos de revisão por pares dos cenários/planos de teste. A documentação dos defeitos encontrados foi mantida, porém, agora há uma utilização mais efetiva dos resultados dos testes para melhoria do produto. Houve uma manutenção na estimativa de horas para planejamento e execução dos testes e a base dos testes ainda está nos requisitos de negócios.

Em resumo, a Equipe 04 demonstrou uma evolução significativa em suas práticas de teste de software, mantendo a frequência de uso de práticas já utilizadas e aprimorando outras áreas para se tornarem uma equipe de nível 3 em maturidade de teste de software.

Em relação ao nível de maturidade em teste de software, a equipe avançou para o nível 3-Definido, pois agora possui política e estratégia de testes, planejamento, monitoramento e controle dos testes, execução e ambientes implementados, práticas referentes ao nível 1 de maturidade. Também foram adotadas práticas de organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais, relacionadas ao nível 2 de maturidade. Por fim, foram implementadas práticas como organização e programa de treinamento de testes, ciclo de vida e integração, testes não funcionais e revisão por pares, referentes ao nível 3, de acordo com o TMMI.

Considerando a sua mensagem anterior em que você mencionou que houve um avanço surpreendente no nível de cobertura de testes unitários, talvez seja interessante revisar essa parte do texto para manter a coerência. Aqui está uma sugestão de melhoria:

Em relação à métrica de cobertura de código por testes unitários, verificou-se que, antes da aplicação do RoMTeS, o nível de cobertura era de 95%. Após três semanas, o valor caiu para 94%, mas é importante destacar que houve uma melhoria na qualidade dos testes, com validações mais precisas e assertivas. Essa melhoria pode ser percebida na evolução do nível de maturidade em testes da equipe e no aumento da frequência na aplicação das práticas de teste de software, como a revisão por pares e os testes não funcionais.

Em relação a Equipe 05, a mesma obteve melhorias significativas tanto na frequência de aplicação das práticas de teste de software quanto nas atividades necessárias para aumentar o nível de maturidade em teste de software da equipe. A equipe obteve progresso ao parar de utilizar bypass para publicar código em ambiente produtivo quando os objetivos de qualidade não estavam completos.

A implementação do RoMTeS ajudou a equipe a aumentar a frequência de uso de várias práticas de teste como testes de mutação, testes regressivos, testes de performance e testes exploratórios. Além disso, a equipe passou a aplicar testes unitários utilizando *mocks* e a validar campos no banco de dados com mais frequência.

A equipe também evoluiu sua estratégia de documentação de cenários de testes, evidenciando execuções de teste e defeitos encontrados. Além disso, a equipe passou a utilizar resultados de testes para melhorar o produto, o que demonstra um foco maior na qualidade do software produzido.

Outra melhoria importante foi a integração de testes automatizados na esteira CI/CD e a garantia de que erros nos testes acarretam falhas na esteira. Isso aumenta a confiabilidade da equipe em relação aos resultados dos testes e ajuda a identificar rapidamente problemas no software.

Por fim, a equipe passou de um nível 2 para um nível 3 no TMMI, demonstrando que suas práticas de teste estão se tornando mais definidas e estruturadas. Com isso, é possível esperar que a equipe continue a melhorar suas práticas de teste e aumentar ainda mais a qualidade do software produzido.

Em relação a quantidade de testes, estes foram estruturados de acordo com os níveis da pirâmide de testes, com ênfase na realização de testes de unidade na base da pirâmide. Dessa forma, foram fornecidas orientações para aumentar a quantidade de testes realizados no nível de base, a fim de garantir uma melhor cobertura de testes.

Com base no TMMI, é possível afirmar que a equipe em questão avançou para o nível 3-Definido em maturidade de testes. Isso porque a equipe implementou políticas e estratégias de teste, bem como estabeleceu processos de planejamento, monitoramento e controle dos testes, execução e ambientes de teste. Essas práticas são consideradas nível I de maturidade.

Além disso, a equipe também implementou práticas referentes ao nível II de maturidade, incluindo organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais. Agora, a equipe também implementou práticas como organização e programa de treinamento de testes, ciclo de vida e integração, testes não funcionais e revisão por pares, que são práticas referentes ao nível III de maturidade de testes, de acordo com o TMMI.

Em resumo, a equipe avançou significativamente na maturidade de testes, estabelecendo processos e práticas mais avançados e abrangentes para garantir a qualidade do software. Isso demonstra um compromisso com a excelência e uma melhoria contínua dos processos de teste.

A cobertura de testes unitários é uma métrica fundamental para avaliar a qualidade dos testes realizados pela equipe de desenvolvimento de software. Antes de implementar a ferramenta RoMTeS, a equipe apresentava uma média de 75% de cobertura de testes unitários em suas aplicações. No entanto, após três semanas de uso da ferramenta, o valor da métrica saltou para 96%.

Esse resultado é impressionante e demonstra claramente a eficácia da ferramenta em melhorar a qualidade dos testes unitários realizados pela equipe. A ferramenta RoMTeS ajudou a identificar as áreas críticas que precisavam ser testadas, permitindo que a equipe focasse na melhoria da cobertura de testes nessas áreas. Além disso, a ferramenta também ajudou a automatizar e simplificar o processo de teste, tornando-o mais rápido e preciso.

Em suma, a métrica de cobertura de código por testes unitários é um indicador importante da qualidade dos testes de software e a utilização da ferramenta RoMTeS se mostrou altamente eficaz em melhorar essa métrica para a equipe em questão.

Em relação a Equipe 06, observou-se uma maior frequência na utilização de práticas como desenvolvimento orientado a testes, utilização de *Mocks* nos testes unitários e consideração dos testes unitários como necessários para a conclusão de uma história/entrega. Além disso, houve um aumento na validação dos testes utilizando valores do banco de dados.

E Equipe 06 conseguiu, por meio da aplicação do plano de ação aumentar a frequência de uso de práticas como desenvolvimento orientado a testes, testes unitários com uso de *mocks*, validações em status de retorno e campos do banco de dados, além de melhorias em testes de performance e testes de stress. Eles também evoluíram sua estratégia de documentação de cenários de testes, evidências de execução e defeitos, e passaram a planejar a quantidade de testes por nível da pirâmide de testes, estruturando uma maior quantidade de testes mais próximo ao código-fonte das aplicações.

A equipe também melhorou a revisão por pares dos cenários/planos de teste, que passou de nunca para frequentemente, e implementou a frequente aplicação de testes não funcionais para conclusão da história. Com isso, a equipe 6 evoluiu para o nível 3 - Definido em maturidade de teste de software, com processos muito bem estruturados e melhoria nos processos de testes.

A Equipe 06 também passou a utilizar a documentação dos resultados dos testes para melhoria do produto, o que é uma prática importante para garantir a qualidade do software. Além disso, a equipe implementou a estruturação dos testes em ciclos de vida de aplicação, o que ajuda a garantir a integração dos testes ao longo do processo de desenvolvimento.

No geral, as melhorias implementadas pela equipe 6 em relação às práticas de teste de software foram significativas, ajudando a aumentar a qualidade do produto e a maturidade em teste de software da equipe.

Também houve avanços no abandono da prática de utilizar *bypass* para publicar código em ambiente produtivo quando os objetivos de qualidade não estavam completos e, ao ajustar a quantidade dos testes de acordo com a pirâmide de testes, apesar de ainda executar muitos testes manuais exploratórios. A padronização da documentação de cenários de teste e evidências de execução dos testes também foi adotada pela equipe.

Em resumo, a Equipe 06 apresentou melhorias significativas na aplicação de práticas de teste de software e atividades relacionadas à maturidade de teste de software. Ainda há espaço para aprimoramentos, mas as melhorias realizadas até o momento indicam um caminho promissor para a equipe.

Com base no TMMI, a equipe avançou do nível 2-Gerenciado para o nível 3-Definido em maturidade de testes de software. Agora, a equipe possui uma política e estratégia de testes definidas, além de um planejamento, monitoramento e controle dos testes, execução e ambientes implementados, práticas referentes ao nível I de maturidade. A equipe também adotou práticas de organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais, que são características do nível II de maturidade de testes. Adicionalmente, foram implementadas práticas de organização e programa de treinamento de testes, ciclo de vida e integração, testes não funcionais e revisão por pares, características do nível III de maturidade de testes, consolidando a evolução da equipe para o nível Definido.

Em relação à métrica de cobertura de código por testes unitários, que representa a média do nível de cobertura dos testes unitários das aplicações desenvolvidas pela equipe, foi constatado um avanço significativo após a implementação do RoMTeS. Antes da aplicação da metodologia, a cobertura de código por testes unitários era de 51%, enquanto ao final das três semanas, o valor subiu para 63%, demonstrando uma melhoria substancial na qualidade dos testes unitários realizados pela equipe.

A Equipe07 apresentou melhorias tanto na frequência da aplicação de práticas de teste de software quanto em atividades necessárias para aumentar o nível de maturidade da equipe nesse aspecto. Analisando as práticas de teste de software, foi observado que a houve um aumento na frequência de uso de várias práticas, como análise estática de código, testes regressivos, testes de integração com banco de dados ou outras aplicações, testes não funcionais e testes exploratórios. Além disso, a equipe agora está aplicando testes funcionais

automatizados com validações em status de retorno e em campos do banco de dados, o que não era feito anteriormente.

Outra melhoria importante é que os testes estão sendo planejados com mais cuidado e em diferentes níveis da pirâmide de testes. A equipe também está documentando melhor os cenários de teste, realizando revisões por pares e evidenciando as execuções de teste. Todos esses aspectos ajudam a garantir uma cobertura de teste mais abrangente e eficaz.

Outro ponto positivo é que a equipe está utilizando um ambiente específico para execução dos testes e integrando os testes automatizados na esteira CI/CD, o que ajuda a garantir que os testes sejam executados com frequência e os resultados sejam usados para melhorar o produto.

Além disso, a equipe estruturou seus testes de acordo com os níveis da pirâmide de testes, com orientações para a utilização de uma maior quantidade de testes no nível base da pirâmide de testes, ou seja, nos testes de unidade. Essas melhorias indicam um avanço significativo na abordagem da equipe em relação ao teste de software e contribuem para aumentar a qualidade e confiabilidade do produto.

Com base no modelo TMMI, a Equipe07 se manteve no nível 3 - Definido em maturidade de testes, demonstrando um bom nível de maturidade. No entanto, a equipe também implementou práticas que contribuem para aumentar ainda mais seu nível de maturidade em teste de software. Por exemplo, a equipe melhorou significativamente seus processos de testes unitários, utilizando técnicas como *Mocks* e testes regressivos.

A equipe também aprimorou seus testes funcionais, automatizando-os com validações em status de retorno e em campos do banco de dados, além de melhorar os testes de integração. A padronização da documentação de cenários de testes, evidências de execução e documentação de defeitos encontrados também contribui para aumentar o nível de maturidade da equipe.

Essas práticas bem estruturadas demonstram o compromisso da equipe com a qualidade do produto e evidenciam a evolução da equipe em relação aos processos de teste de software. Com essas melhorias, a Equipe07 está cada vez mais próxima de atingir o próximo nível de maturidade no TMMI e se consolidar como uma equipe de alta maturidade em teste de software. Ao analisar a métrica de cobertura de código por testes unitários da Equipe07, que consiste na média do nível de cobertura de testes unitários das aplicações desenvolvidas pela equipe, foi observado que antes da aplicação do RoMTeS, esse valor era de 50%. Ao final das três semanas, o nível de cobertura de código por testes unitários se manteve próximo ao inicial, com um avanço de apenas 1%, atingindo um valor de 51%.

É importante ressaltar que esse avanço foi bem pequeno em comparação com as demais equipes de desenvolvimento da empresa Gama. Isso pode ter ocorrido devido ao baixo interesse dos membros da equipe de desenvolvimento em melhorar as práticas de teste de software e compreender a importância da cobertura de código por testes unitários.

No entanto, é fundamental que a equipe compreenda que a cobertura de código por testes unitários é uma métrica importante que indica a qualidade dos testes realizados e a confiabilidade do produto. A equipe pode buscar melhorar essa métrica por meio da implementação de práticas de testes unitários mais eficientes e da conscientização sobre a importância do teste de software para o sucesso do projeto.

Os resultados da análise do cenário alcançado para a Equipe 08 revelou melhorias significativas tanto em relação à frequência de aplicação das práticas de teste de software quanto nas atividades necessárias para aumentar o nível de maturidade da equipe em testes de software.

Ao analisarmos as práticas de teste de software adotadas pela equipe, notamos a adoção de desenvolvimento orientado a testes, antes, a equipe nunca utilizava essa prática, mas após a implementação do plano de ação, a equipe passou a utilizá-la raramente. Isso indica que a equipe está se esforçando para incorporar a prática de TDD em suas atividades diárias, o que pode resultar em um código mais limpo e menos defeituoso.

Houve um aumento da frequência de uso de testes de mutação, antes a equipe já aplicava testes de mutação sempre, mas após a implementação do plano de ação, essa prática passou a ser utilizada frequentemente. Isso demonstra que a equipe está investindo em testes mais avançados e robustos, visando a detecção de defeitos que possam passar despercebidos pelos testes unitários convencionais.

Houve a adoção de testes de stress e performance, o que indica que a equipe está preocupada em garantir que o software seja capaz de lidar com altas demandas e desempenhe bem em situações extremas. Em relação a documentação de cenários de testes e evidências de execução, sua frequência passou a ser frequentemente, o que indica que a equipe está se esforçando para registrar e documentar suas atividades de teste, o que pode ser útil para análise posterior e para compartilhar informações com outros membros da equipe.

Em geral, a Equipe 08 demonstrou um aumento significativo em sua maturidade de teste de software, passando do nível 1 para o nível 2 do TMMI. Isso sugere que a equipe está mais consciente da importância dos testes e está investindo em práticas mais avançadas e robustas para garantir a qualidade do software. Com a implementação do plano de ação a equipe conseguiu melhorar suas práticas de teste de software em diversas áreas, o que pode resultar em um software de maior qualidade e menos defeituoso.

Com relação à métrica de cobertura de código por testes unitários, que é uma medida do nível de cobertura de testes unitários das aplicações desenvolvidas pela equipe, observou-se que os resultados obtidos antes da aplicação do RoMTeS eram de 79%. Após três semanas de trabalho, essa métrica se manteve próxima ao valor inicial, atingindo 81% de cobertura de código por testes unitários. Embora o avanço da equipe nessa área tenha sido modesto, é importante destacar o excelente trabalho realizado na estruturação dos testes, o que certamente trará muitos benefícios para a equipe de desenvolvimento de software e para a organização como um todo.

Os resultados da análise do cenário alcançado para a Equipe 09 apresentou melhorias significativas em relação à frequência de aplicação de práticas de teste de software, bem como em atividades necessárias para aumentar o nível de maturidade da equipe em testes de software. Em relação às práticas de teste de software, houve uma maior frequência na utilização de práticas como o desenvolvimento orientado a testes, testes unitários e funcionais como requisitos para a conclusão de histórias/entregas. A equipe também se conscientizou da importância de um ambiente específico para a execução de testes, e deixou de utilizar o bypass para publicar código em ambiente produtivo quando os objetivos de qualidade não eram atendidos.

Além disso, a equipe começou a incluir testes de desempenho e testes exploratórios, quando necessário, e padronizou a documentação dos cenários de teste, defeitos encontrados e evidências de execução dos testes. A pirâmide de testes passou a ser utilizada no planejamento da quantidade de testes a serem criados pela equipe de desenvolvimento de software, o que garantiu um processo de desenvolvimento mais eficiente e de maior qualidade.

Ao compararmos a frequência de uso das práticas de teste de software antes e após a execução do plano de ação, percebe-se que a equipe passou a incluir práticas como o desenvolvimento orientado a testes, testes unitários, testes automatizados integrados na esteira CI/CD, documentação de defeitos encontrados, planejamento e execução de testes estimados em horas, e gerenciamento do ciclo de vida da aplicação. A equipe agora pratica testes exploratórios, tem padrões para escrever cenários/planos de teste, realiza revisão por pares dos cenários/plano de teste, evidencia as execuções de teste, e documenta defeitos encontrados, o que indica uma melhoria significativa na qualidade de seus testes.

Em relação ao nível de maturidade em testes de software, a equipe demonstrou avanços significativos, incorporando práticas que aumentam o nível de maturidade da equipe, como processos bem estruturados e melhorias nos processos de planejamento dos testes com base na pirâmide de testes, ambientes específicos para a execução dos testes, realização de testes

funcionais automatizados e de desempenho. Além disso, a padronização da documentação dos cenários de teste, defeitos encontrados e das evidências de execução dos testes demonstra um esforço da equipe em documentar e manter um controle eficiente das atividades de teste.

Apesar dessas melhorias, a equipe ainda se mantém no nível 2-Gerenciado em maturidade de testes de acordo com o TMMI. No entanto, é importante destacar que a equipe demonstrou uma evolução significativa em suas práticas de teste de software, o que certamente contribuirá para um melhor desempenho na entrega de produtos de qualidade aos seus clientes.

Considerando a métrica de cobertura de código por testes unitários, que representa a média do nível de cobertura de testes unitários das aplicações desenvolvidas pela equipe, houve uma melhoria significativa após a aplicação do RoMTeS. Antes da intervenção, a cobertura de testes unitários era de 50%, e ao final das três semanas, esse valor aumentou para 55%. Esse avanço na métrica de cobertura de código indica que a equipe está dedicando mais atenção à criação de testes unitários para as suas aplicações, o que pode contribuir para uma maior confiabilidade e qualidade do software produzido.

Os resultados da análise do cenário alcançado para a Equipe 10 revelou melhorias significativas tanto em relação à frequência na aplicação das práticas de teste de software quanto em atividades necessárias para aumentar o nível de maturidade de teste de software da equipe. Ao analisar as práticas de teste de software, percebeu-se melhorias significativas na frequência de uso de práticas em teste de software após a implementação do plano de ação. Práticas como desenvolvimento orientado a testes, testes de mutação, análise estática de código, testes regressivos, ambiente específico para execução dos testes, validação de campos no banco de dados, testes não funcionais, entre outros, foram implementadas com maior frequência na equipe. Além disso, a equipe começou a planejar e estimar a quantidade de testes por nível da pirâmide de testes.

A equipe também avançou positivamente ao parar de utilizar bypass para publicar código em ambiente produtivo quando os objetivos de qualidade não estavam completos, o que reflete um maior comprometimento com a qualidade do produto.

A equipe também aprimorou suas validações por meio de testes automatizados, verificando não apenas o status de retorno das aplicações, mas também os valores dos campos do banco de dados. Além disso, houve uma padronização da documentação dos cenários de teste, defeitos encontrados e evidências de execução dos testes.

Em relação ao nível de maturidade em teste de software, com base no TMMI, esta equipe avançou do nível 2-Gerenciado para o nível 3-Definido, o que significa que agora a equipe possui uma política e estratégia de testes bem definidas, planejamento, monitoramento

e controle dos testes, execução e ambientes implementados, além de práticas referentes ao nível 1 de maturidade, como organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais. A equipe também implementou práticas de nível 2 de maturidade, como revisões de processo e testes de regressão automatizados. Essas melhorias permitiram que a equipe atingisse um nível mais avançado de maturidade em teste de software.

Considerando a métrica de cobertura de código por testes unitários, que é uma média do nível de cobertura de testes unitários das aplicações desenvolvidas pela equipe, é possível observar que houve um significativo avanço no desempenho da Equipe11. Antes da aplicação do RoMTeS, a equipe apresentava um nível de cobertura de 70%, enquanto ao final das três semanas, esse valor aumentou para 88%. Esse resultado evidencia o comprometimento da equipe em aprimorar a qualidade do código produzido, bem como a adoção de práticas de desenvolvimento orientado a testes, contribuindo para a detecção precoce de falhas e redução do tempo de correção de problemas no código.

Os resultados da análise do cenário alcançado para a Equipe 11 apresentou melhorias da Equipe11 em relação à aplicação de práticas de teste de software e no aumento do nível de maturidade da equipe nesse quesito. Foi possível identificar uma frequência maior na utilização das práticas de teste de software como desenvolvimento orientado a testes, testes unitários, testes unitários com *mocks*, análise estática de código, testes regressivos, validação de status de retorno de aplicações, testes funcionais automatizados, desativação de testes na esteira CI/CD, testes funcionais necessários para conclusão da história, testes não funcionais necessários para conclusão da história, documentação de defeitos encontrados, estimativas de horas para planejamento e execução de testes, planejamento de quantidade de testes por nível da pirâmide de testes e gestão do ciclo de vida da aplicação.

Destaca-se a estruturação dos testes com base na pirâmide de testes, que passou a ser utilizada no planejamento da quantidade de testes a serem criados pela equipe de desenvolvimento de software. Entretanto, apesar da equipe ter planejado a quantidade de testes com base na pirâmide, já possuíam uma grande quantidade de testes funcionais automatizados, fazendo com que os testes unitários criados não substituíssem os testes automatizados como a maior quantidade de testes da equipe.

Apesar das melhorias nas práticas de teste de software, a Equipe11 ainda se mantém no nível 2-Gerenciado em maturidade de testes, conforme o TMMI. No entanto, é importante destacar que houve um incremento significativo de práticas que aumentam o nível de maturidade da equipe, como a estruturação dos processos de planejamento dos testes com base na pirâmide de testes, a utilização de testes unitários com *Mocks* e testes regressivos, além da

padronização da documentação dos cenários de teste, defeitos encontrados e das evidências de execução dos testes. Tais práticas são fundamentais para a evolução do nível de maturidade da equipe em testes de software.

Em relação à métrica de cobertura de código por testes unitários, que representa a média do nível de cobertura dos testes unitários das aplicações desenvolvidas pela equipe, houve um aumento de 3% na cobertura após a implementação do RoMTeS. Embora a diferença não seja significativa, é importante destacar que a equipe realizou uma reestruturação em relação aos testes de software, com ênfase na aplicação da pirâmide de testes e no aumento da frequência de testes unitários com uso de *Mocks*. Isso demonstra um esforço para aprimorar a qualidade dos testes e, consequentemente, do software desenvolvido.

Os resultados da análise do cenário alcançado para a Equipe 12 demonstram melhorias significativas tanto em relação à frequência na aplicação das práticas de teste de software como em atividades necessárias para aumentar o nível de maturidade em teste de software.

Antes da aplicação do plano de ação, a equipe não utilizava práticas como desenvolvimento orientado a testes, testes de mutação, análise estática de código, testes regressivos, validação do status de retorno das aplicações, validação dos campos no banco de dados, testes não funcionais, testes exploratórios, escrita de cenários/planos de teste, estimativa de horas para planejamento/execução de testes e testes baseados em requisitos de negócios.

Após a implementação do plano de ação, a equipe passou a utilizar essas práticas, além de melhorar a frequência de uso das práticas que já utilizavam, como testes unitários, testes funcionais automatizados, ambiente específico para execução de testes, testes automatizados integrados na esteira CI/CD, desativação de testes na esteira CI/CD, testes funcionais necessários para conclusão da história e gestão de ciclo de vida de aplicação.

Essas melhorias levaram a equipe a evoluir de um nível inicial de maturidade de teste de software para um nível gerenciado, com processos bem estruturados e práticas mais maduras, como a utilização de *mocks* nos testes unitários, testes regressivos executados com certa frequência, validações em status de retorno e validações em banco de dados, além de testes de performance e de stress e documentação.

A equipe também passou a planejar a quantidade de testes por nível da pirâmide de testes, evidenciar as execuções de teste, realizar revisão por pares dos cenários/plano de teste e utilizar os resultados dos testes para melhoria do produto.

Com base no TMMI, a equipe avançou do nível 2-Gerenciado para o nível 3-Definido em maturidade de testes de software. Agora, a equipe possui políticas e estratégias para os testes, além de planejamento, monitoramento e controle das atividades relacionadas. As práticas

de organização, treinamento, ciclo de vida e integração dos testes, revisão por pares e testes não funcionais foram implementadas, elevando o nível de maturidade para o nível II. Com a implementação dessas práticas, a equipe alcançou o nível III de maturidade, garantindo a execução dos testes e a disponibilidade dos ambientes necessários para a execução dos testes.

Em relação à métrica de cobertura de código por testes unitários, que é uma medida do nível de cobertura de testes unitários das aplicações desenvolvidas pela equipe, foi observado que, antes da aplicação do RoMTeS, essa métrica era de 50%. No entanto, após três semanas de trabalho com a metodologia, a cobertura aumentou para 63%.

Essa melhoria foi possível graças à estruturação da equipe em relação aos testes de software, especialmente no que diz respeito à pirâmide de testes e aos testes unitários utilizando *Mocks*. Essas práticas permitiram que a equipe desenvolvesse testes mais eficazes e abrangentes, aumentando assim a cobertura de código.

Os resultados da análise do cenário alcançado para a Equipe 13 teve como resultados melhorias significativas tanto na frequência de aplicação das práticas de teste de software quanto nas atividades necessárias para aumentar o nível de maturidade de teste de software da equipe.

Um dos principais destaques é a implementação do desenvolvimento orientado a testes, que passou de frequentemente para sempre. Isso significa que os testes são considerados parte integrante do processo de desenvolvimento e são realizados de forma contínua e sistemática. Além disso, a equipe sempre utiliza testes unitários e *mocks*, o que indica um alto nível de qualidade no desenvolvimento de código.

Outra melhoria importante foi a implementação de testes regressivos frequentemente, o que contribui para a detecção de regressões em versões anteriores do software. Além disso, a equipe passou a realizar testes de performance frequentemente, o que demonstra uma preocupação em garantir que o software funcione de forma satisfatória em diferentes cenários de uso.

A equipe também passou a realizar testes não funcionais sempre, o que indica que a equipe tem considerado aspectos como segurança, usabilidade e escalabilidade em seu processo de teste. Além disso, houve uma conscientização da equipe sobre a importância de não utilizar bypass para publicar código em ambiente produtivo quando os objetivos de qualidade não estavam completos. Essa mudança de postura contribuiu para um aumento na qualidade do software desenvolvido.

A equipe de desenvolvimento de software adotou uma abordagem mais abrangente em relação aos testes, incluindo os testes funcionais e não funcionais como necessários para a

conclusão das histórias/entregas. Além disso, houve um aumento nos testes exploratórios e uma padronização na documentação dos cenários de teste, defeitos encontrados e evidências de execução dos testes.

Para estruturar melhor os testes, a equipe passou a utilizar a pirâmide de testes no planejamento da quantidade de testes a serem criados. Essa abordagem permitiu que a equipe aumentasse a quantidade de testes unitários em relação aos testes funcionais automatizados. Essa diferença na quantidade de testes foi possível porque a equipe começou com uma quantidade muito pequena de testes funcionais automatizados.

Essa mudança na abordagem de testes permitiu que a equipe aumentasse a eficácia e a eficiência dos testes realizados. Além disso, a padronização da documentação dos testes e a estruturação dos testes com base na pirâmide de testes contribuíram para uma melhor organização dos processos de teste e para uma melhor compreensão dos resultados obtidos.

A equipe de desenvolvimento de software conseguiu manter seu nível de maturidade em teste de software no nível 2-Gerenciado do TMMI. No entanto, a equipe implementou práticas adicionais que contribuíram para o aumento do nível de maturidade da equipe.

Entre essas práticas estão: processos muito bem estruturados, melhoria nos processos de planejamento dos testes com base na pirâmide de testes, testes unitários utilizando *Mocks*, testes regressivos e de desempenho, padronização da documentação dos cenários de teste, defeitos encontrados e evidências de execução dos testes.

Essas práticas adicionais contribuíram para um aprimoramento significativo dos processos de teste de software da equipe, permitindo que ela entregue um software de qualidade e atenda às expectativas dos usuários finais.

A métrica de cobertura de código por testes unitários é uma importante ferramenta para avaliar a qualidade do software desenvolvido pela equipe. Essa métrica representa a média do nível de cobertura de testes unitários das aplicações que a equipe desenvolve.

Antes da aplicação do RoMTeS, os resultados da equipe indicavam uma cobertura de 80% do código por testes unitários. Após três semanas de implementação do RoMTeS, essa cobertura aumentou para 88%. Isso representa um aumento significativo na qualidade do software, pois quanto maior a cobertura de testes unitários, menor a chance de ocorrerem defeitos no código.

Esse aumento na cobertura de testes unitários é resultado da estruturação da equipe em relação aos testes de software, principalmente em relação à pirâmide de testes e testes unitários utilizando *Mocks*. Essas práticas permitem que a equipe crie testes mais eficazes e de alta qualidade, resultando em uma maior cobertura de código.

Para a análise dos resultados alcançados pelas equipes de desenvolvimento de software, foi dedicado um tempo médio de meia hora por equipe para a preparação do material de análise comparativo. Esse documento foi usado como base para a apresentação dos resultados aos gestores e ao superintendente.

A análise comparativa permitiu identificar as melhorias alcançadas pelas equipes de desenvolvimento de software, bem como as práticas e técnicas mais eficazes na obtenção desses resultados. Com base nessas informações, a equipe pode definir novas metas e estratégias para continuar melhorando seus processos de desenvolvimento de software.

Na última atividade da etapa, a equipe de desenvolvimento apresentou os resultados obtidos por meio de uma reunião de uma hora de duração para cada um dos gestores, conforme solicitado em reunião de esclarecimentos iniciais. Essas reuniões contaram com a presença das equipes de desenvolvimento e seus respectivos gestores.

Além disso, foi realizada uma reunião geral com a participação dos gestores, líderes técnicos e superintendente para a apresentação dos resultados gerais obtidos pelas equipes de desenvolvimento. Essa reunião teve como objetivo compartilhar os resultados obtidos pelas equipes e discutir estratégias para manter e melhorar a qualidade do software produzido.

É importante destacar que a comunicação dos resultados é uma etapa fundamental para garantir que todos os envolvidos estejam cientes do progresso da equipe e comprometidos com a melhoria contínua. Além disso, a apresentação dos resultados individuais para cada gestor permite que as equipes recebam um feedback mais direcionado para cada equipe de desenvolvimento de software.

7.6. Entrevista

A entrevista teve como objetivo principal analisar a opinião dos gestores e liderança envolvida na aplicação do RoMTeS, após sua implementação. Foi realizada por meio de reunião com a liderança das 13 equipes de desenvolvimento de software e contou com sete questões chave para a avaliação do produto. Participaram da entrevista os três gestores das equipes de desenvolvimento de software, a superintendente e a liderança técnica de cada equipe. Apesar de relativamente curta, com duração de meia hora, a entrevista foi importante para coletar feedbacks valiosos sobre a aplicação do *roadmap* de melhoria de teste de software.

A primeira pergunta realizada na entrevista foi sobre a percepção da liderança em relação ao aumento do nível de maturidade em teste de software após a implantação do

RoMTeS. A resposta foi positiva, afirmando que a implantação do *roadmap* agregou valor para a organização como um todo, principalmente no aprimoramento das práticas de teste para identificar erros e defeitos mais cedo no ciclo de desenvolvimento, evitando impactos nos clientes. Além disso, a liderança destacou que os resultados obtidos com o RoMTeS não se limitam apenas ao momento de sua aplicação, mas também em práticas que serão mantidas pelas equipes para manter o nível de maturidade alcançado. Em resumo, a liderança teve uma percepção muito positiva do valor agregado pelo *roadmap*.

A segunda pergunta realizada foi sobre a percepção da liderança em relação à usabilidade do RoMTeS, ou seja, se o *roadmap* é de difícil ou fácil aplicação. A resposta da liderança foi que a aplicação do RoMTeS é relativamente fácil, mas requer um guia na apresentação e condução das equipes de desenvolvimento para aplicarem as melhores práticas em teste de software. Eles destacaram que a documentação é clara e bem estruturada, e que o RoMTeS fornece uma estrutura adequada para a implementação das práticas recomendadas. No entanto, enfatizaram que é importante ter alguém que conheça bem o *roadmap* para ajudar as equipes a aplicá-lo corretamente.

A terceira pergunta realizada foi sobre a percepção da liderança sobre a viabilidade do RoMTeS. A liderança concordou que o *roadmap* é viável e traz resultados positivos, pois permite a percepção e detecção de erros antes que eles cheguem ao ambiente produtivo, o que é um fator de grande importância para a organização.

A quarta pergunta realizada foi sobre a opinião da liderança a respeito da utilidade do RoMTeS em organizações que adotam o desenvolvimento ágil de software com arquitetura de micro serviços. De maneira geral, a liderança expressou a opinião de que o *roadmap* pode ser utilizado não apenas em organizações que utilizam essa arquitetura, mas também em outras arquiteturas de desenvolvimento.

A quinta pergunta feita foi sobre a viabilidade de comercialização do RoMTeS. A liderança concordou que o *roadmap* é um produto viável para ser comercializado. No entanto, por ser uma organização de grande porte, seria necessária uma equipe dedicada para atuar com a consultoria e aplicação do *roadmap*, ao invés de somente uma pessoa.

Como última pergunta, foi questionada a liderança sobre sua experiência com a implantação do RoMTeS. Em resposta, a liderança comentou que foi uma experiência positiva, já que a estruturação de alguns processos e práticas de teste levou as equipes a melhorarem a qualidade de suas entregas. Durante o processo de implantação, a liderança recebeu feedbacks positivos das equipes de desenvolvimento de software, o que fortaleceu a ideia de que o *roadmap* foi uma iniciativa importante para o amadurecimento da área de testes na organização.

Em resumo, a entrevista com a liderança após a implementação do RoMTeS teve como objetivo avaliar a percepção sobre a melhoria do nível de maturidade em teste de software e a utilidade do *roadmap*. A liderança confirmou que houve uma percepção de valor agregado, especialmente no aprimoramento das práticas de teste para encontrar erros e defeitos antes de irem para ambiente produtivo. Também foi destacado que o RoMTeS é fácil de aplicar, mas requer um guia para a apresentação e condução dos times. Quanto à factibilidade e utilidade do RoMTeS, a liderança concordou que é um produto viável e que pode ser aplicado em diferentes arquiteturas de software. Finalmente, a liderança relatou que a experiência com a implantação do RoMTeS foi muito boa, com melhoria na qualidade das entregas e feedbacks positivos das equipes de desenvolvimento. Inclusive, pessoas de outras equipes pediram a aplicação do *roadmap* em suas equipes por perceberem o valor agregado.

7.7. Considerações parciais

Durante a implantação do *roadmap* RoMTeS em uma organização de grande porte com diversas equipes de desenvolvimento, enfrentou-se diversos desafios, mas os resultados foram positivos. Durante as três semanas de implementação do *roadmap*, as equipes de desenvolvimento puderam perceber melhorias significativas em suas práticas de teste de software, levando a um aumento no nível de maturidade.

Apesar disso, houve uma exceção: uma equipe de desenvolvimento não apresentou melhorias significativas no nível de maturidade em teste de software. Esse resultado destacou a importância de avaliar e adaptar a abordagem do *roadmap* para diferentes equipes e contextos de desenvolvimento.

Em geral, a liderança percebeu que a aplicação do RoMTeS trouxe muito valor para a organização, permitindo a detecção de erros e defeitos mais cedo no processo de desenvolvimento, o que resultou em uma melhoria na qualidade das entregas e satisfação dos clientes. Além disso, a aplicação do *roadmap* não se limitou apenas ao momento de sua implementação, mas também na continuidade das práticas e processos adotados pelas equipes de desenvolvimento, garantindo a manutenção do nível de maturidade em teste de software alcançado.

No geral, a experiência da liderança com a implantação do *roadmap* RoMTeS foi muito positiva, com feedbacks positivos das equipes de desenvolvimento e até mesmo de outras áreas da organização solicitando a aplicação do *roadmap* em suas próprias equipes. Isso reforça a

importância de investir em processos de melhoria contínua em testes de software para garantir a qualidade e eficácia do desenvolvimento de software em organizações de grande porte.

Durante a implantação do RoMTeS em uma organização de grande porte com múltiplas equipes de desenvolvimento, houve desafios, mas também resultados positivos. As equipes elogiaram as práticas de teste de software abordadas nos planos de ação, mas sentiram que precisavam de mais tempo para implementá-las completamente. A liderança da equipe ficou satisfeita com os resultados e sugeriu repetir o processo em outras equipes após alguns meses, com um tempo reduzido de aplicação e mais reuniões de acompanhamento. Um gestor sugeriu que a aplicação do plano de ação deveria ser feita por seis semanas para obter mais resultados positivos, enquanto os outros gestores concordaram que três semanas seriam suficientes, aumentando apenas a quantidade de reuniões semanais para aprimorar os resultados. No geral, a experiência de aplicação do RoMTeS foi bem-sucedida e trouxe melhorias significativas para as práticas de teste de software na organização.

8. LIÇÕES APRENDIDAS

Ao longo da implementação do *roadmap* de melhoria de teste de software, várias lições valiosas foram aprendidas. Essas lições estão documentadas aqui para orientar a profissionais qeu queiram aplicar o RoMTeS e apoiar futuros pesquisadores no desenvolvimento de suas investigações na área.

O primeiro aprendizado destacado é que, ao entrar em contato com uma empresa para a implementação de um *roadmap*, é crucial e extremamente relevante que as negociações ocorram com antecedência. Isso permite que as organizações planejem adequadamente seus backlogs de produto, alocando tempo necessário para a implementação sem comprometer as entregas da equipe ou da organização como um todo.

Outra lição aprendida com a implementação do RoMTeS nas três organizações é a importância de realizar a implementação em uma única equipe ou organização de cada vez. Isso permite que o pesquisador se concentre na coleta de resultados e garanta a máxima disponibilidade para a implantação bem-sucedida de seu produto nas respectivas organizações.

Um aprendizado adicional deste estudo, especificamente relacionado à implementação do *roadmap* de melhoria de teste de software, envolve a elaboração de treinamentos prévios à implantação do RoMTeS nas organizações. Dessa forma, o responsável pela aplicação do plano

terá tempo suficiente para estudar e adaptar os treinamentos às necessidades específicas de cada organização.

Durante a implementação do *roadmap* de melhoria de teste de software, foram identificadas algumas ameaças que podem afetar o RoMTeS. A primeira delas é a falta de conhecimento dos gestores e lideranças sobre a importância dos testes de software para a organização e a falta de priorização dessas atividades. Isso pode comprometer a eficácia do RoMTeS.

Outra ameaça é a falta de interesse dos membros das equipes nas quais as melhorias são implementadas. Sem incentivos e interesse, os membros podem não aprender e aplicar as melhorias de teste de software.

Além disso, é importante destacar que a avaliação do nível de maturidade das equipes em teste de software foi realizada com base no modelo TMMI, que requer um profissional devidamente capacitado e certificado para a certificação da organização no nível de maturidade.

Por fim, a abertura da organização para a implementação do *roadmap* de melhoria de teste de software pode ser uma ameaça, uma vez que envolve processos internos de desenvolvimento e cuidados com a segurança para evitar o vazamento de informações confidenciais.

De maneira geral, a implementação do *roadmap* de melhoria de teste de software em três organizações, de diferentes portes, proporcionou resultados muito positivos para as equipes envolvidas. Na primeira organização, de pequeno porte, os membros das equipes, de maneira geral, foram atenciosos, estudiosos e entusiasmados em aplicar as melhorias propostas no plano de ação.

Na segunda organização, de médio porte, que possuía uma estrutura e maturidade de testes classificada como inicial pelo TMMI, a implementação do *roadmap* de melhoria de teste de software foi desafiador. Durante as três semanas de implantação, foram observadas melhorias significativas para a organização, que passou a adotar como métrica a quantidade de testes unitários realizados.

Para a terceira organização, de grande porte, a implementação do *roadmap* de melhoria de teste de software também foi desafiadora, devido ao grande número de equipes de desenvolvimento envolvidas. No entanto, as melhorias propostas foram bem recebidas pelas equipes e os resultados foram muito positivos.

Em resumo, as três organizações puderam se beneficiar com a implementação do *roadmap* de melhoria de teste de software, o que evidencia a importância do investimento em testes de software para o sucesso dos projetos de desenvolvimento.

Na implementação do *roadmap* RoMTeS em uma organização de grande porte, devido ao grande número de equipes de desenvolvimento envolvidas (13 no total), foi necessário dedicar muita atenção à preparação para os materiais de treinamento de cada equipe. Durante as três semanas de implantação do *roadmap*, foram percebidas melhorias significativas para a organização como um todo, especialmente para as equipes de desenvolvimento, que tiveram seu nível de maturidade em teste de software melhorado e seus processos de práticas de testes de software estruturados.

Um dos principais aprendizados da aplicação do *roadmap* foi a importância da utilização de câmeras abertas nas reuniões de aplicação dos planos de ação. Isso trouxe um comprometimento muito grande por parte da equipe, que ficou mais atenta e participativa nos processos de melhoria implementados. Outro fator perceptível para o sucesso na implementação das melhorias foi o engajamento dos líderes, que demonstraram interesse e apoiaram a iniciativa.

Nas três organizações que implementaram o *roadmap* de melhoria de teste de software, foi identificado que alguns membros das equipes possuíam entendimentos incorretos sobre os conceitos das práticas de teste de software.

Ao comparar as organizações Beta e Gama em relação ao tempo disponibilizado para que as equipes se dedicassem à aplicação das melhorias em teste de software, percebeu-se que um período maior dedicado à implementação das melhorias resultou em melhores resultados e, consequentemente, em um aumento no nível de maturidade em teste de software.

De forma geral, foi notada uma necessidade de enfatizar temas como a pirâmide de teste e a utilização de testes unitários com *Mocks* em grande parte dos planos de ação elaborados. Quanto aos resultados alcançados com a melhoria dos testes de software através da implementação do RoMTeS, observou-se um impacto muito positivo nas três organizações, com exceção de apenas uma equipe pertencente à organização de grande porte. A falta de interação da equipe durante os treinamentos e a pouca aplicabilidade das melhorias no cotidiano foram os possíveis principais fatores que contribuíram para essa falta de resultados positivos.

Por fim, a implementação do RoMTeS trouxe resultados positivos para as organizações que o adotaram. Foi possível identificar ameaças e oportunidades para a melhoria dos testes de software, bem como aplicar planos de ação específicos para cada organização e equipe, considerando seu nível de maturidade em teste de software. Além disso, foi observado que a dedicação e tempo das equipes para aplicação das melhorias e o engajamento dos líderes foram fatores cruciais no sucesso da implementação.

CONCLUSÃO

A transformação digital é responsável por uma série de benefícios e modernização para a sociedade como um todo e a tecnologia é essencial para a vida cotidiana. A presença de softwares em atividades como agendamento de reuniões de trabalho e previsão do tempo mudou radicalmente a forma como vivemos. Em um mundo cada vez mais tecnológico, a importância do teste de software é evidente, pois ele é usado para alinhar expectativas, detectar problemas e defeitos e manter a estabilidade dos sistemas antes que eles cheguem ao cliente final.

Com base no objetivo geral deste trabalho, foi desenvolvido um *roadmap* para melhorar os testes de software em processos ágeis de desenvolvimento de micro serviços. Foram levantadas 31 práticas de teste de software, que foram analisadas por meio de um questionário de validação com 182 profissionais da área de teste de software. O questionário trouxe respostas que indicaram que as práticas de teste de software identificadas na literatura são realmente as práticas de teste de software aplicadas pelas organizações que atuam com o ambiente de desenvolvimento e testes de software. Estas práticas de teste de software foram a base para a construção do RoMTeS.

Com o intuito de melhorar o teste de software, foi desenvolvido um *roadmap* que auxilia as organizações que atuam com arquitetura de micro serviços a implantar melhores práticas de testes de softwares. O *roadmap* foi construído a partir do estudo das melhores práticas de teste de software encontradas na literatura e validadas por meio do questionário.

Chamado de RoMTeS, o *roadmap* de melhoria de teste de software consiste em cinco etapas. A primeira etapa é "Esclarecimento", que envolve a apresentação do *roadmap* aos gestores e a definição da sua abrangência. A segunda etapa é "Definições", que envolve a identificação das equipes selecionadas, a apresentação do *roadmap* e a identificação do cenário atual de cada equipe. A terceira etapa é "Averiguações", que envolve a análise e organização dos dados do cenário atual, a construção do plano de melhoria para cada equipe e a apresentação dos planos aos gestores. A quarta etapa é "Melhorias", que envolve a aplicação dos planos de ação em cada equipe de desenvolvimento. A quinta e última etapa é "Aferições", que envolve a identificação do cenário alcançado após a execução do plano de ação, a análise dos resultados e a comunicação dos resultados obtidos.

A última etapa deste trabalho foi a aplicação deste *roadmap* de melhoria de teste de software em três organizações, uma de pequeno porte, chamada neste trabalho como Alfa, uma

organização de médio porte, chamada de Beta e, a implantação deste *roadmap* em uma organização de grande porte, chamada de organização Gama.

Na implementação em uma organização de pequeno porte todos os membros da equipe estiveram sempre empolgados com a aplicação do mesmo e, conforme as etapas avançavam os resultados começavam a aparecer, seja pela cultura da equipe de desenvolvimento que passou a ser mais questionador no sentido de qualidade e testes de software, como também, na aplicabilidade das práticas de teste de software.

Sua implementação em uma organização de médio porte com a estrutura e maturidade de testes classificada como inicial, pelo TMMI, foi um desafio grande e, durante o período das três semanas de implantação do *roadmap* de melhoria de teste de software, percebeu-se melhorias significativas para a organização, que passou a analisar com métricas a quantidade de testes unitários.

Sua implementação em uma organização de grande porte, com a atuação direta em 13 equipes de desenvolvimento de software foi um desafio enorme pela atenção e preparação dos materiais de treinamento para cada equipe de desenvolvimento e, durante o período das três semanas de implantação do *roadmap* de melhoria de teste de software, percebeu-se melhorias significativas para a organização, em especial, para as equipes de desenvolvimento de software que tiveram seu nível de maturidade em teste de software melhorados, além da estruturação de processos de práticas de testes de software.

Com a aplicação do *roadmap* de melhoria de teste de software nas três organizações pode-se perceber que esta ferramenta é útil para melhoria do processo de teste de software e aumento do nível de maturidade das equipes nas quais for utilizado.

Percebeu-se também que as três organizações, independentemente de seu porte, recomendaram a aplicação do *roadmap* de melhoria de teste de software para outras equipes e, para outras organizações, concordando na comercialização do RoMTeS. Percebeu-se também que, com a melhoria das práticas de teste de software e, o aumento do nível de maturidade das equipes em teste de software, houve uma percepção de aumento da qualidade das aplicações que as equipes eram responsáveis.

Em conclusão, o desenvolvimento de software de qualidade é fundamental para o sucesso das empresas que atuam no mercado atual, cada vez mais exigente. A aplicação de melhores práticas de teste de software, aliada à utilização de *roadmaps* de melhoria, pode ser uma estratégia eficaz para garantir a qualidade do software entregue aos clientes e manter a competitividade no mercado.

Para trabalhos futuros, o *roadmap* poderá ser aplicado em organizações que não trabalhem com a arquitetura de micro serviços, para validar sua utilização em diversos cenários por organizações com realidades diferentes das organizações mencionadas neste trabalho, as quais serviram de base para aplicação do *roadmap* de melhoria de teste de software.

REFERÊNCIAS

ALBUQUERQUE, Lindolfo Galvão de. **Strategic management.** In: FLEURY, Maria Tereza (coord.). **As pessoas na organização**. 4. ed. São Paulo: Editora Gente, 2002.

AUDY Jorge Luis Nicolas. **Desenvolvimento Distribuído de Software**, Rio de Janeiro, Elsevier 2008.

AWAD, M. A. A **Comparison between Agile and Traditional Software Development Methodologies**. 2005. 77 f. Tese (Doutorado) - Curso de Computer Science And Software Engineering, University Of Western Australia, Perth, 2005.

BARANYI P. and CSAPO Acta A. Polytechnica Hungarica (2012): Definition and Vol. Synergies 9, of No. 1, pp. 67-83, **Cognitive Infocommunications**, (ISSN 1785-8860).

BARTIÉ, Alexandre. Garantia da qualidade de software. Elsevier, 2002.

BARTIÉ, Gulf Professional Publishing (2002): **Garantia da qualidade de software** PARTE II. 1ª edição.

BASHAR, M. R. DASH, S. K. ROY, P. S. **An empirical study of the effectiveness of static analysis tools for identifying security vulnerabilities**, 2020. International Journal of Information Management.

BECK, K. et al. **Os doze princípios do software ágil**. [S.l:s.n], 2011. Disponível em: http://www.manifestoagil.com.br/principios.html. Acesso em: 13 out. 2021.

BECK, K. **Test-Driven Development By Example.** Addison Wesley, 2002. Estados Unidos.

BECK, K., & GAMMA, E. **Test-Driven Development: A Practical Guide**, 2003. Addison-Wesley Professional.

BECK, K. S. K. S. J. E. A.; **Manifesto for agile software development**. Disponível em: <http://agilemanifesto.org/> Acesso em: 13 out. 2021.

BELLOQUIM, A. Modelagem de Software: Ontem, Hoje e Amanhã. Developers' Magazine. Rio de Janeiro: **Axcel Books,** ano 6, n.70, p.10-13, jun.2002.
C. Jakobsen and J. Sutherland, "Scrum and CMMI – Going from Good to Great: are you ready-ready to be done-done?" in Agile 2009, Chicago, 2009.

BORGES, Eduardo N. **Conceitos e Benefícios do Test Driven Development.** Universidade Federal do Rio Grande do Sul / Instituto de Informática, 2006.

BRAUDE, Eric **Projeto de Software: Da programação à arquitetura: Uma abordagem baseada em Java,** 1a, ed , Artmed ed, 2004.

BRAY, O.H.; GARCIA, M.L. **Technology roadmapping**: the integration of strategic and technology planning for competitiveness. Proceedings of the Portland International Conference

on Management of Engineering and Technology (PICMET), Portland, 27- 31st July, Piscataway, p.25-28, 1997.

CARVALHO, A. A. d. **Scrummps 2.0:** Evolução de uma Ferramenta Interativa para Suporte ao Scrum e Mps.BR. 2013. Citado na página 32.

CERVO, A. L. Bervian, P. A. Metodologia científica. 5.ed. São Paulo: Prentice Hall, 2002.Cockburn, A. e Highsmith, J. "Agile Software Development: The Business of Innovation", **IEEE Computer,** Sept., pp. 120-122, 2001.

COELHO, José Antônio Farias. BOTELHO, Sérgio. Tahin, Elda Fontinele. (2012**). Roadmap tecnológico: um estudo preliminar.** Revista Eletrônica de Ciência Admnistrativa (RECADM).

COHN, Mike. **Succeeding with agile**: Software Development Using Scrum. Boston: Addison Wesley, 2010.

COKBURN, A.: **Agile software development: the business of innovation.** Computer 34(9), 120-127.

DABBISH, L. et al. Social coding in GitHub: transparency and collaboration in an open software repository. In Proc. of the ACM 2012 Conf. on Computer Supported **Cooperative Work.** ACM, 1277–1286. 2012.

DELGADO, R. N. & VARGAS, A. G. **A systematic mapping study on mutation** testing, 2016. Information and Software Technology, 78, 118-132.

DEWAYNE, E. PERRY, D. E, LEBLANC. D, BINKLEY. **Static analysis of object-oriented software: A survey**. 2010. Journal of Systems and Software

DIOGO, Gabriel. **8 ferramentas de automação de testes que ajudam você na gestão de TI**. Disponível em <https://www.impacta.com.br/blog/2019/06/03/8-ferramentas-automacao-testes-ajudam-gestao-de-ti/>

DUVALL, P. Continuous Integration: **Improving Software Quality and Reducing Risk.** 2007. Addison-Wesley Professional.

ENGHOLM, Júnior Hélio. **Engenharia de Software na prática**, São Paulo: Novatec Editora, 2010.

FALBO, Ricardo A. **Integração de Conhecimento em um Ambiente de Desenvolvimento de Software.** Tese de Doutorado, Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

FREEMAN, S. PRYCE, N. **The Art of Unit Testing: With Examples in C#**, 2014. Manning Publications.

FERREIRA, Avelino. **Como garantir a qualidade do software: Testes automatizados**.

FRANCO, Eduardo Ferreira. **Um modelo de gerenciamento de projetos baseado nas metodologias ágeis de desenvolvimento de software e nos princípios da produção enxuta**. 2007. Tese de Doutorado. Universidade de São Paulo.

GARCÍA RAMÍREZ F. **Testing ágil de software con herramientas libres y abiertas.** 2016.

GARG, S. KUMAR, R**. An empirical study on performance testing of software systems: Trends and challenges.** 2019. Journal of Systems and Software, [S.l.], v. 152, p. 34-47. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0164121219301019. Acesso em: 4 mar. 2023.

GASPARETO, Otavio. **Test Driven Development. Universidade Federal do Rio Grande do Sul** / Instituto de Informática, 2006.

GRAHAM, D. J. Veenendaal, E. V. **TMap NEXT: Business-driven test management**. 2008. UTN Publishers.

GIL, Antonio Carlos. Como elaborar projetos de pesquisa. São Paulo: Atlas, 2007.

HETZEL, B. **The art of software testing**. 1993. Wiley.

HETZEL, B. **The Complete Guide to Software Testing.** 1984. QED Information Sciences, Inc.

HIGHSMITH, J.Agile software development: The business of innovation. Computer, **IEEE**, v. 34, n. 9, p. 120–127, 2004.

HOHL, Philipp et al. **Forces that Prevent Agile Adoption in the Automotive Domain.** Product-focused Software Process Improvement, [s.l.], p.468-476, 2016. Springer International Publishing.

HUMBLE, J. FARLEY, D. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**. 2011. Addison-Wesley Professional.

ISTQB. **Certified tester:** foundation level syllabus 2011, br. 2011.

JAKOBSEN, C. R.; SUTHERLAND, J**. Scrum and CMMI Going from Good to GreatAgile Conference,** 2009. AGILE '09. Anais... In: AGILE CONFERENCE, 2009. AGILE '09. ago. 2009

JANZEN, D. S., & SAIEDIAN, H. **Test-driven development: Concepts, taxonomy, and future direction. Journal of Database Management**, 2008. 19(1), 25-40.

JOHNSON, H.A. Trello. **J Med Libr Assoc**. 2017 abril;105(2):209–11.

JORDÃO, Sônia. **A arte de Liderar: Vivenciando Mudanças num Mundo Globalizado**. 2ª ed. Belo Horizonte: Tecer, 2004

KAPPEL TA. 2001. **Perspectives on roadmaps: how organizations talk about the future.** *Journal of Product Innovation Management* 18: 39-50.

Kang, K. C. Cohen, S. G. Hess, J. A. Novak, W.E. Peterson, A.S. **Formality considerations in software testing.** 2009. IEEE Software, [S.l.], v. 26, n. 3, p. 43-50. Disponível em: https://ieeexplore.ieee.org/document/4812306. Acesso em: 4 mar. 2023.

KELLY, W. **Take your project management application open source with ProjectLibre**. Obtido de TechRepublic: 2012.

KNOCHE and HASSELBRING 2018] Knoche, H. and Hasselbring, W. (2018). **Using Microser- vices for Legacy Software Modernization.** *IEEE Software*, 35(3):44–49.

KOSCIANSKI, André; SOARES, Michel dos Santos. Qualidade de software. 2006.

KOSTOFF, R. N.; SCHALLER, R. R. **Science and technology roadmaps**. IEEE Transactions on Engineering Management, v. 48, n. 2, p. 132-143, 2001.

LE, H. **General UI Changes in Odoo** 9&10. 2014.

LEWIS, James; FLOWER, Martin. **Microservices.** Disponível em: <http://martinfowler.com/articles/microservices.html>. Acesso em: 13 out. 2021.

M. Müller, F. Padberg, About the Return on Investment of Test-Driven Development. Disponível em http://www.ipd.uka.de/mitarbeiter/muellerm/publications/edser03.pdf.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Metodologia do trabalho científico: procedimentos básicos, pesquisa bibliográfica, projeto e relatório, publicações e trabalho científico**. 6a. ed. São Paulo: Atlas, 2001.

MATHARU, Gurpreet Singh et al. **Empirical Study of Agile Software Development Methodologies**. Acm Sigsoft Software Engineering Notes, [s.l.], v. 40, n. 1, p.1-6, 6 fev. 2015. Association for Computing Machinery (ACM).

MINETTO, Elton Luís. Frameworks para Desenvolvimento em PHP. Chapecó : Novatec Editora Ltda., 2007.

MYERS, G. J. SANDLER, C. BADGETT, T. **The art of software testing (2nd ed.).** 2003 John Wiley & Sons.

NAGAPPAN, N. MAXIMILIEN, E. M. BHAT, T. WILLIAMS, L. **Realizing quality improvement through test driven development: Results and experiences of four industrial teams**, 2008. Empirical Software Engineering, 13(3), 289-302

NANDHAKUMAR, Joe; AVISON, David E.. **The Fiction of Methodological Development: a field study of information systems development**. Information Technology & People, Southampton, v. 12, n. 2, p.176-191, 1999.

NEWMAN, S. **Building microservices: designing fine-grained systems.** [S.l.]: ”O’Reilly Media,Inc.”,2015. Citado5vezesnaspáginas8,13,15,16e17.

OFFUTT, J. **The couverture method of software testing. Journal of Systems and Software**, 2002.

PÁDUA Filho, Wilson. Engenharia de Software: Fundamentos, Métodos e Padrões. 2a. ed. Rio de Janeiro: LTC-Livros Técnicos e Científicos Editora S.A., 2003.

PALESTINO, Caroline Munhoz Corrêa. **ESTUDO DE TECNOLOGIAS DE CONTROLE DE VERSÕES DE SOFTWARES.**2015. 72 f. TCC (Graduação) - Curso de Gestão da Informação, Ciências Sociais Aplicadas, Universidade Federal do Paraná, Curitiba, 2015.

PHAAL, R., Farrukh, C.J.P. and Probert, D.R. (2004) **Technology Roadmapping—A Planning Framework for Evolution and Revolution. Technological Forecasting and Social Change**, 71, 5-26. http://dx.doi.org/10.1016/S0040-1625(03)00072-6

PEREIRA, P.; Torreão, P.; Maçal, A. S. **Entendendo Scrum para Gerenciar Projetos de Forma Ágil**. 2007.

PETERS, James F. **Engenharia de Software Teoria e Prática**, 1a ed, Rio de Janeiro: Campus, 2001.

PRESSMAN, R.; **Engenharia de Software**. McGraw-Hill Interamericana do Brasil Ltda, 2011.

PRESSMAN, Roger S. Software Engineering: a practitioner’s approach. 5.ed. United States: McGraw-Hill, 2001.
RAINSBERGER, J. B. JUnit Recipes: practical methods for programmer testing. United States: Manning Publications Co, 2005.

QUATRANI, Terry. **Visual Modeling with Rational Rose and UML**. 3. ed. New York: Addison-Wesley, 2002

REZENDE Denis Alcides, **Engenharia de Software e Sistemas de Informação**, 3a, ed Rio de Janeiro, Brasport 2005.

RUSSO, Rosaria Fátima Segger Macri; DA SILVA, Luciano Ferreira; LARIEIRA, Claudio Luis Carvalho. Do manifesto ágil à agilidade organizacional. **Revista de Gestão e Projetos,** v. 12, n. 1, p. 1-10, 2021.

SAMANTA, D. D. DASGUPTA, S. **The Importance of Test Environment in Software Testing.** 2012. International Conference on Information and Knowledge Management. p. 2687-2690.

SCHACH Stephen R. **Engenharia de Software: Os Paradigmas Clássico e Orientação a Objetos**, 7a ed, 2010.

SCHWABER Ken; SUTHERLAND Jeff. **Guia do Scrumm: Um guia definitivo para o Scrum: As regras do jogo**, 2013.

SHWETA, R. PANDEY, S. K. DUBEY, S. R. Automated functional testing of mobile applications: Challenges and opportunities. 2016. Journal of Systems and Software, 114, 101-115.

SINGH, S.K. (2008) **Role of Leadership in Knowledge Management: A Study. Journal of Knowledge Management,** 12, 3-15. http://dx.doi.org/10.1108/1367327081088421.

SILVA, E. D. P. d.; FIGUEIREDO, L. S. **Um estudo qualitativo sobre qualidade de software no sistema de educação a distância da UFGD, o Moodle.** Sede da Universidade da África. UVA, 2019. 20

SOARES Michel dos Santos, **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**, **INFOCOMP**, 2004. Acesso em: 13 out. 2021. < http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> Acesso em 10 de novembro de 2014.

SOMMERVILLE, IAN. **Engenharia de Software**. 8a, ed, Pearson Education, 2011.

SOUZA, N. M. et al. **Relação entre arquitetura de software e teste de software:** um mapeamento sistemático. São Carlos - SP, 2017. 24, 26

SRINIVASAN, S. SRIPRIYA, S**. A comparative study of automated functional testing tools.** 2012. International Conference on Computer Communication and Informatics.

SUTHERLAND, J. et al. **Fully distributed Scrum - the secret sauce for hyperproductive offshored development teams.** In: AGILE CONFERENCE, 2008, Toronto. Proceedings. Toronto, 2005. p. 339-344.

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

WOODWARD, ELIZABETH. **A Practical Guide to Distributed Scrum,** (2012). Pearson Education India. Steffan Surdek, Matthew Ganis.

WOHLIN, C. THELIN, T**. The Role of the Test Environment in Software Testing**. 2002. 15th International Conference on Software Engineering and Knowledge Engineering

XENOS, M. et al. **Object-oriented metrics – a survey**. [S.1]. Federation of European Software Meansurement Asssociations, 2000.

ZARELLI, Guilherme B. **Pirâmide de Testes** — Definindo uma boa suíte de testes para seu Software. 31 ago. 2020. Disponível em: < https://medium.com/luizalabs/pir%C3%A2mide-de-testes-definindo-uma-boa- su%C3%ADte-de-testes-para-seu-software-a6864886f29b>. Acesso em: 13 out. 2021.

APÊNDICE

Apêndice 01 – Questionário de práticas de teste de software

Apêndice 02 – Entrevistas sobre a estruturação do RoMTeS

Apêndice 03 – Relação do RoMTeS com o TMMi

Apêndice 04 – Roteiro de entrevista da aplicação do RoMTeS

APÊNDICE 01 - QUESTIONÁRIO DE PRÁTICAS DE TESTE DE SOFTWARE

Práticas de testes de software emmetodologias ágeis

Prezado (a) Participante,

Você está sendo convidado a participar da pesquisa **"Melhores práticas em teste de software"**. Sua contribuição muito engrandecerá nosso trabalho, pois participando destapesquisa, você nos trará uma visão específica pautada na sua experiência sobre o assunto. Esclarecemos, contudo, que sua participação não é obrigatória. Sua recusa nãotrará nenhum prejuízo em sua relação com a pesquisadora ou com a instituição proponente.

Os objetivos deste estudo são avaliar quais das práticas de teste de software que foram identificadas na literatura são adotadas pelos profissionais da área, bem como, possibilitara descoberta de demais práticas de teste de software utilizadas nas organizações.

As informações obtidas por meio desta pesquisa serão confidenciais e asseguramos o sigilo sobre sua participação. Os dados serão divulgados de forma a não possibilitar suaidentificação, protegendo e assegurando sua privacidade.

A qualquer momento você poderá tirar suas dúvidas sobre o projeto e sua participação. Ao final desta pesquisa. O trabalho completo será disponibilizado no site do Programa de Mestrado Profissional em Gestão e Tecnologia em Sistemas Produtivos do Centro PaulaSouza.

Pesquisadora: Fernanda Pereira Aguiar,
E-mail: fernanda.aguiar@cpspos.sp.gov.br
Orientador: Prof. Dr. Marcelo Duduchi Feitosa,
E-mail: marcelo.feitosa@cpspos.sp.gov.br

Ao continuar com este questionário você concorda com o Termo de Consentimento Livre eEsclarecido

Declaro que entendi os objetivos de minha participação na pesquisa e concordo em participar. Registro também que concordo com o tratamento de meus dados pessoais parafinalidade específica desta pesquisa, em conformidade com a Lei nº 13.709 – Lei Geral deProteção de Dados Pessoais (LGPD).

Esse questionário deverá ser respondido em aproximadamente 10 minutos.

1. Você tem experiência com o ambiente de desenvolvimento de software utilizando métodos ágeis?

   [ ]Sim
   [ ]Não

2. Qual é o porte da organização que você atua?

   [ ] Microempresa – até 9 funcionários Pequena empresa – de 10 a 49 funcionários
   [ ] Média empresa – de 50 a 99 funcionários
   [ ] Grande empresa – de 100 – 499 funcionários
   [ ] Empresa com 500 ou mais funcionários

3. Qual a extensão de atuação da organização que você trabalha?
   É possível a seleção de mais de uma opção

   [ ] Multinacional
   [ ] Nacional
   [ ] Norte
   [ ] Nordeste
   [ ] Centro-Oeste
   [ ] Sudeste
   [ ] Sul

4. O desenvolvimento de software é qual atividade da organização que você trabalha?
   Considerar a principal atividade da organização

   [ ] Atividade fim (principal atividade da organização)
   [ ] Atividade meio (atividade necessária pra ofertar seu produto/serviço)

5. Qual o tipo de software desenvolvido pela organização que você trabalha?
   É possível a seleção de mais de uma opção

   [ ] Software aplicativo para dispositivos móveis
   [ ] Software aplicativo para desktops
   [ ] Software aplicativo para Web
   [ ] Software de sistema para atender a outros programas, sistema operacional e rede
   [ ] Software embarcado
   [ ] Software de engenharia ou científico
   [ ] Software ERP
   [ ] Software para linha de produtos (software de prateleira)
   [ ] Software de Inteligência Artificial
   [ ] Jogos
   [ ] Outro:_____

6. Na organização que você trabalha, a arquitetura de micro serviços é utilizada?

[ ] Não sei informar
[ ] Sim
[ ] Não

7. Em média, por quantos membros são compostos o time de desenvolvimento de um determinado projeto ágil na organização que você trabalha?

[ ] Menos de 3
[ ] De 3 a 5
[ ] De 6 a 9
[ ] Mais de 9

8. Como se dá a formação do time quanto a localização dos membros do grupo de trabalho na organização que você trabalha?

[ ] Todos trabalham no mesmo local
[ ] Equipes distribuidas

9. Qual é sua função principal no time?

[ ] Desenvolvedor ou funções afins
[ ] Testador ou ou funções afins
[ ] Gerente de projeto ou funções afins
[ ] Outro:_____

10. Quanto tempo de experiência você possui no ambiente de desenvolvimento de software utilizando métodos ágeis?

[ ] Menos de um ano
[ ] 1 a 3 anos
[ ] 4 a 5 anos
[ ] 6 a 10 anos
[ ] Acima de 10 anos

11. Qual(is) método(s) ou abordagem(s) de desenvolvimento de software são utilizados na organização que você trabalha?
É possível a seleção de mais de uma opção

[ ] Scrum
[ ] Dynamic Systems Development
[ ] (DSDM) Feature Driven
[ ] Development (FDD) Extreme
[ ] Programming (XP)
[ ] Crystal
[ ] Lean
[ ] Kanban
[ ] Outro:_____

12. Durante o planejamento de uma iteração, quais são os fatores considerados no âmbito das atividades de testes, na organização que você trabalha?
É possível a seleção de mais de uma opção

[ ] Criação e execução de testes de histórias
[ ] Testes em pares com outros testadores e desenvolvedores
[ ] Validações do negócio
[ ] Automação de novos testes funcionais
[ ] Execução de testes de regressão automatizados
[ ] Execução e automação de testes não funcionais
[ ] Demonstração para os stakeholders
[ ] Criação dos ambientes de testes
[ ] Revisão dos cenários de teste pelo time
[ ] Outro:_____

13. Na organização que você trabalha, qual o tempo de iteração para que seja possível codificar e executar testes das funcionalidades selecionadas para a iteração?

[ ] 1 semana
[ ] 2 semanas
[ ] 3semanas
[ ] 4 semanas
[ ] Deve-se estar pronto para entregar o software funcionando todos os dias
[ ] Outro:_____

14. Na organização que você trabalha, os testes são realizados durante toda a iteração?

[ ] Sim
[ ] Não

15. Selecione a opção correspondente ao seu conhecimento sobre as seguintes

	Não conheço	Conheço pouco	Conheço bem
Desenvolvimento orientado a testes	[ ]	[ ]	[ ]
Testes unitários Testes Testes de muttação	[ ]	[ ]	[ ]
Análise estática de código	[ ]	[ ]	[ ]
Esteira de integração CI/CD	[ ]	[ ]	[ ]
Teste funcional automatizado	[ ]	[ ]	[ ]
Testes regresivos	[ ]	[ ]	[ ]
Ambientes de execução de teste	[ ]	[ ]	[ ]
Documentação de testes	[ ]	[ ]	[ ]
Documentação de defeitos	[ ]	[ ]	[ ]
Validações/resultados esperados	[ ]	[ ]	[ ]
Gerrenciamento de ciclo de vida da aplicação	[ ]	[ ]	[ ]
Execução de testes	[ ]	[ ]	[ ]
Testes manuais	[ ]	[ ]	[ ]
Mocks	[ ]	[ ]	[ ]
Testes de integração	[ ]	[ ]	[ ]
Testes de performance	[ ]	[ ]	[ ]
Testes de stress Pirâmide de testes	[ ]	[ ]	[ ]

16. Em relação às práticas de teste de software, selecione a frequência de uso na organização que você trabalha:

	Não sei informar	Nunca	Raramente	Frequentemente	Sempre
Desenvolvimento orientado a testes	[ ]	[ ]	[ ]	[ ]	[ ]
Testes unitários	[ ]	[ ]	[ ]	[ ]	[ ]
Testes unitários utilizam *mocks*	[ ]	[ ]	[ ]	[ ]	[ ]
São aplicados testes de mutação	[ ]	[ ]	[ ]	[ ]	[ ]
Testes unitários são necessários para conclusão da história	[ ]	[ ]	[ ]	[ ]	[ ]
Análise estática de código	[ ]	[ ]	[ ]	[ ]	[ ]
Testes regressivos	[ ]	[ ]	[ ]	[ ]	[ ]
Há um ambiente específico para execução dos testes	[ ]	[ ]	[ ]	[ ]	[ ]
O status de retorno das aplicações são validados	[ ]	[ ]	[ ]	[ ]	[ ]
Os campos no banco de dados são validados	[ ]	[ ]	[ ]	[ ]	[ ]
Testes funcionais automatizados	[ ]	[ ]	[ ]	[ ]	[ ]
Testes automatizados estão integrados na esteira CI/CD	[ ]	[ ]	[ ]	[ ]	[ ]
Erro nos testes acarreta fallha na esteira CI/CD	[ ]	[ ]	[ ]	[ ]	[ ]
Ocorre desativação de testes(bypass) na esteira CI/CD	[ ]	[ ]	[ ]	[ ]	[ ]
Testes funcionais são necessários para conclusão da história	[ ]	[ ]	[ ]	[ ]	[ ]
Testes de integração com banco de dados ou outras aplicações	[ ]	[ ]	[ ]	[ ]	[ ]
Testes de performance	[ ]	[ ]	[ ]	[ ]	[ ]
Testes de stress					
Testes não funcionais são necessários para conclusão da história	[ ]	[ ]	[ ]	[ ]	[ ]
Testes explorratórios	[ ]	[ ]	[ ]	[ ]	[ ]
São escritos cenários/planos de teste	[ ]	[ ]	[ ]	[ ]	[ ]
Há padrões na escrita dos cenários/planos de teste	[ ]	[ ]	[ ]	[ ]	[ ]
Revisão por pares dos cenárrios/plano de teste	[ ]	[ ]	[ ]	[ ]	[ ]
São evidenciadas as execuções de teste	[ ]	[ ]	[ ]	[ ]	[ ]
Ocorre a documentação dos defeitos encontrados	[ ]	[ ]	[ ]	[ ]	[ ]
Resultados dos testes são utilizados para melhoria do produto	[ ]	[ ]	[ ]	[ ]	[ ]
São estimadas horas para planejamento/ execução dos teste	[ ]	[ ]	[ ]	[ ]	[ ]
Os testes são baseados em requisitos de negócios	[ ]	[ ]	[ ]	[ ]	[ ]
Gestão de ciclo de vida de aplicação (Exemplo:SilkCentral, AzureDevOps, Jenkins)	[ ]	[ ]	[ ]	[ ]	[ ]
Execução dos testes são realizados com frequência	[ ]	[ ]	[ ]	[ ]	[ ]
São planejados quantidade de testes por nível da pirâmide de testes	[ ]	[ ]	[ ]	[ ]	[ ]

17. Para se elaborar uma estratégia de automação, quais são os níveis da aplicação utilizados na organização que você trabalha, se relacionarmos à pirâmide de teste?
É possível a seleção de mais de uma opção

[ ] Não se utiliza automação de testes
[ ] Unitário (Automação de testes de unidade)
[ ] Serviço (Automação na camada de API - Application programming interface)
[ ] GUI (Automação na camada de GUI - Graphical user interface)
[ ] Outro:_____

18. Quais fatores são utilizados para se considerar a automação de um teste na organização que você trabalha?
É possível a seleção de mais de uma opção

[ ] Não se utiliza automação de testes
[ ] Utilização para teste de regressão
[ ] Facilidade de automação
[ ] Ganhos em relação à automação de processo manual
[ ] Necessidade de negócio
[ ] Impacto na aplicação
[ ] Feedback mais rápido quanto ao resultado dos testes (passou/falhou)
[ ] Todos os testes são automatizados
[ ] Outro:_____

19. Na organização que você trabalha, como se organizam as equipes de desenvolvimento em relação ao teste de software?

[ ] A equipe é integrada, mas com separação de papéis entre desenvolvedores e testadores
[ ] A equipe é integrada, com equipes multidisciplinares e sem separação de papeis
[ ] A equipe de teste é independente
[ ] Outro:_____

20. Na organização que você trabalha, qual é a contribuição da equipe de teste no processo de desenvolvimento?
    É possível a seleção de mais de uma opção

    [ ] Promove feedback durante todo o processo de desenvolvimento
    [ ] Ajuda com conhecimento técnico de implantação do código do teste a ser realizado
    [ ] Assume a liderança nos testes de aceitação
    [ ] Assume a liderança nos testes de regressão Desenvolve os planos de testes
    [ ] Revela cenários de teste adicionais através de testes exploratórios Garante que a cobertura de teste é adequada
    [ ] Lidera os esforços de automação
    [ ] Lidera os esforços de testes de integração Executa testes de nível de sistema
    [ ] Mantém ambientes de teste e os dados disponíveis Identifica problemas e partes técnicas de teste
    [ ] Faz a priorização das funcionalidades e cenários que devem ser automatizados
    [ ] Outro:_____

21. Na sua opinião, existe alguma prática relacionada ao teste de software que não foi mencionada no questionário?
    Sua sugestão será muito bem vinda na contribuição desta pesquisa

_____
_____
_____
_____

APÊNDICE 02 – ENTREVISTAS SOBRE A ESTRUTURAÇÃO DO ROMTES

Para assegurar que o *roadmap* seja adaptado à realidade das organizações seu desenvolvimento foi realizado observando a opinião de três especialistas em teste de software que opinaram de forma independente e livre sobre as características iniciais preconizadas na sua primeira versão, suas etapas e funcionalidades. O primeiro entrevistado é um profissional com mais de 10 anos de experiência no mercado, atuando na área de teste de software, arquitetura de software e pré-venda de soluções.

Para garantir a sinceridade das opiniões dos entrevistados em relação ao *roadmap* RoMTeS, a pesquisa foi esclarecida e os entrevistados foram informados sobre as motivações e justificativas do estudo, além de terem recebido um resumo da pesquisa e uma apresentação detalhada do *roadmap*.

Na segunda etapa, os entrevistados leram e analisaram o *roadmap*, sendo incentivados a questionar e opinar livremente sobre suas necessidades e possíveis melhorias.

Na terceira etapa, todas as considerações dos entrevistados foram registradas de forma cuidadosa, garantindo que nenhuma informação importante fosse perdida.

Na quarta etapa, cada entrevistado foi questionado sobre o que faltou no *roadmap*, bem como sobre qualquer informação desnecessária ou irrelevante contida nele. Além disso, foram feitas três perguntas a cada entrevistado sobre a utilidade, usabilidade e viabilidade do *roadmap*. Para manter a privacidade dos entrevistados, eles serão identificados como Respondente 01, Respondente 02 e Respondente 03.

Entrevista com Respondente 01

A primeira entrevista foi feita com profissional especialista na área de teste de software, com experiência de mais de oito anos atuando em ambiente de desenvolvimento de software ágil e com experiência em testes manuais, automatizados e de desempenho, além de experiência com o desenvolvimento de micro serviços nas linguagens de programação Java e Python. Esta profissional conta com experiência em organizações de grande porte da área de tecnologia e do setor financeiro.

As considerações feitas pela Respondente01 foram pautadas, principalmente sobre as etapas de Esclarecimentos, Definições e Aferição.

As sugestões da respondente 01 sobre a etapa de Esclarecimentos foi sobre o *roadmap* RoMTeS possuir 4 etapas chamadas de "1-Esclarecimento do RoMTeS aos gestores", "2-Esclarecimento sobre a importância das boas práticas em teste de software", "3-Questionário para entender as práticas de teste de software de maior prioridade para a gerência" e "4-Análise dos resultados sobre práticas de teste de software a serem priorizadas pelos gestores".

Na visão da Respondente 01, o item enumerado como "3-Questionário para entender as práticas de teste de software de maior prioridade para a gerência" seria inapropriado para algumas organizações, visto que, em sua visão, em muitas organizações os gestores não tem tanto conhecimento na disciplina de testes de software para opinarem de forma apropriada, sobre os níveis de prioridade dos testes de software e, isto poderia ser um problema para a aplicação do *roadmap* nas organizações que não possuem um nível considerado alto em conhecimento de testes de softwares. Em sua opinião, a Respondente 01 comenta sobre a possibilidade de, nesta etapa, o responsável pela aplicação do *roadmap*, com base em algum modelo de maturidade de teste de software, trazer uma sugestão sobre prioridades necessárias para se atingir uma maior maturidade de teste de software utilizando o *roadmap* RoMTeS.

Em relação a segunda etapa, a etapa de Definições, composta por "1-Definição da abrangência e identificação das equipes", "2-Apresentação do *Roadmap* aos key positions", "3-Apresentação do *Roadmap* às equipes técnicas de desenvolvimento de software" e "4-Identificação do cenário atual das práticas em teste de software", a Respondente 01 mencionou sobre agruparmos as reuniões de forma a unificar os passos 2 e 3, de forma a, em uma única reunião técnica, ocorrer a apresentação do *roadmap* RoMTeS aos líderes técnicos/key positions e equipes técnicas de desenvolvimento de software que terão seus processos de teste de software melhorados. Na visão da respondente 01, a união destas agendas seria interessante para dar um quórum maior para a reunião e, à medida que as algumas pessoas tiverem dúvidas sobre o *roadmap*, mais pessoas já estarão integradas na agenda, diminuindo chances de replicação das agendas.

Em relação a etapa de Averiguações, composta pelas etapas de "1- Análise do cenário atual e do conhecimento das equipes de desenvolvimento nas práticas de teste de software", "Construção do plano de atuação para cada equipe de desenvolvimento de software", "3- Apresentação dos planos de ação para os gestores", "4- Apresentação dos planos de ação para os key positions e líderes técnicos", "5- Apresentação do plano de

ação para a equipe de desenvolvimento", a Respondente 01 fez comentários muito interessantes também.

Sobre os itens "4- Apresentação dos planos de ação para os key positions e líderes técnicos" e "5- Apresentação do plano de ação para a equipe de desenvolvimento", a Respondente 01 comentou sobre sua junção, de forma a possibilitar um paralelismo entre a aplicação do *roadmap* em mais de uma equipe de desenvolvimento ao mesmo tempo, de forma a não prejudicar o tempo final de implantação do *roadmap*. Em sua opinião, isso facilitaria em grandes organizações, a implantação do *roadmap* em diversos grupos de trabalhado ao mesmo tempo.

Em relação a etapa de Melhorias, composta pelas etapas "1-Alinhamento de agenda com os grupos de trabalho" e "2-Aplicação dos planos de ação pelos grupos de trabalho", a Respondente 01 comentou sobre a possibilidade de deixar, desde o início da aplicação das melhorias, uma agenda fixa de uma hora diária com a equipe de desenvolvimento. Em sua visão, esta possibilidade daria mais fluidez a aplicação das melhorias nas práticas de teste de software, visto que todos os membros da equipe teriam um horário fixo para implantação destas melhorias.

De forma geral, a Respondente 01 gostou do *roadmap* RoMTeS e comentou da alta possibilidade de utilização do mesmo pelas organizações, especialmente das organizações que não possuem um processo estruturado de testes de software.

Ao final das análises da Respondente 01, foram feitas 3 questões em relação a utilidade, factibilidade e usabilidade do *roadmap* RoMTeS, bem como se havia algum ponto não mencionado no *roadmap*.

Para o primeiro item perguntando, a utilidade do *roadmap*, a Respondente 01 comentou que achou muito interessante a ideia do *Roadmap* e que, com atua com projetos diversos de desenvolvimento de software, é muito útil e ela mesma pode aplicar o *roadmap* em projetos futuros que atue, pois é muito útil para melhorar as práticas de teste de software aplicadas pelas organizações e grupos de desenvolvimento de software de forma geral, não somente as organizações que atuem com o desenvolvimento de aplicações que tenham a arquitetura de micro serviços. Sobre a factibilidade do *roadmap*, a Respondente 01 comentou que após aplicações das melhorias conversadas sobre o *roadmap* ele ficaria sim, muito simples de ler e de entender, tanto por profissionais da área como por profissionais de áreas semelhantes, o que facilita e muito, a utilização do

roadmap. Em relação a usabilidade do *roadmap*, a Respondente 01 comentou sobre ser muito útil, especialmente pela constante necessidade de oxigenação nas práticas de teste de software observadas por ela nas organizações e projetos por onde atuou.

Ao final da entrevista-, perguntei à Respondente01 sua opinião de forma geral, se o *roadmap* faz sentido para organizações no sentido de servir com um guia orientador para melhoria do processo de teste de software e ela comentou que achou muito interessante o *roadmap* e, ficando inclusive, empolgada para ver os resultados da implantação do RoMTeS nas organizações planejadas.

Entrevista com Respondente 02

A segunda entrevista foi feita com profissional com ampla experiência em desenvolvimento de aplicações e testes automatizados, com experiência de mais de cinco anos atuando em ambiente de desenvolvimento de software ágil. Esta profissional conta com experiência em organizações de médio e grande porte da área de tecnologia e desenvolvimento de aplicações.

As considerações feitas pela Respondente02 foram pautadas, principalmente sobre o tempo destinado a cada tarefa apresentada no *roadmap*. Em sua opinião, o tempo relativo para cada atividade dependeria da equipe de desenvolvimento e das características internas da organização na qual o *roadmap* seria implementado. Para ela, em uma organização de pequeno porte, seria necessário menos reuniões de alinhamento, pela quantidade reduzida de pessoas em cargos de liderança e, comparado a uma organização de grande porte, seria necessário um tempo maior para os alinhamentos por haver muitos cargos de liderança e, seria necessário passar por muitas lideranças até alcançar a gerência/liderança correta com permissões para liberar a implementação do *roadmap*.

As sugestões da respondente 02 sobre a etapa de Esclarecimentos foi sobre a quantidade de reuniões de esclarecimento do *roadmap* RoMTeS ser compatível com o tamanho da organização, de forma que uma organização pequena, os assuntos tratados na etapa possam ser definidos em uma única reunião enquanto, em uma reunião maior, com diversos níveis de gerencia, fossem necessárias mais de duas reuniões para decisões de equipes selecionadas e demais decisões.

Outro ponto muito importante levantado pela Respondente 02 é que seria interessante uma anotação/ressalva do tempo mínimo necessário para aplicação das melhorias em parceria com a equipe de desenvolvimento de software para que, dependendo da organização, seriam disponibilizados muitas horas e, em outras, pouquíssimo tempo seria destinado para a implantação das melhorias, trazendo um resultado muito pequeno pela falta de tempo para aplicação das melhorias.

De maneira geral, a Respondente 02 gostou do *roadmap* RoMTeS e viu a possibilidade de sua implantação não somente em organizações que utilizem a arquitetura de micro serviços, mais também, para organizações que utilizem outras arquiteturas e até organizações que não atuam com o ambiente ágil de desenvolvimento de software, sendo aplicável, segundo ela, também para organizações que utilizam metodologias tradicionais e metodologias híbridas de desenvolvimento de software.

Ao final das análises da Respondente 02, foram feitas 3 questões em relação a utilidade, factibilidade e usabilidade do *roadmap* RoMTeS, bem como se havia algum ponto não mencionado no *roadmap*.

Para o primeiro item perguntando, a utilidade do *roadmap*, a Respondente 02 comentou que a ideia do *roadmap* é fascinante, especialmente para pessoas que atuam na área da qualidade e desenvolvimento de software. Ela comentou que os resultados que o *roadmap* traz são facilmente mensuráveis na organização para a qual atua, sendo um projeto de grande visibilidade em caso de implantação, sendo um *roadmap* extremamente útil, especialmente para as organizações que trabalham com projetos.

Sobre a factibilidade do *roadmap*, a Respondente 02 comentou que o *roadmap* traz uma linguagem clara e direta, que todos os profissionais da área tem muita facilidade em entender e que isso facilita sua aplicação. Já em relação a usabilidade do *roadmap*, a Respondente 02 comentou sobre ser muito útil, desde que a organização dê a devida importância para os testes de software. Ela também mencionou que, uma constância na aplicação do *roadmap* traria ainda melhores benefícios para as organizações e propôs, a ideia de uma reaplicação do *roadmap* nas mesmas organizações e equipes após um período de 6 meses, tempo que, para ela, seria suficiente para estruturar os aprendizados e práticas obtidos e, continuar o processo de melhoria de teste de software e aumentar o nível de maturidade em teste de software.

Ao final da entrevista, perguntei à Respondente02 sua opinião de forma geral, se o *roadmap* faz sentido para organizações no sentido de servir com um guia orientador para

melhoria do processo de teste de software e ela comentou que sim, achou muito interessante e, achou uma prática interessante para não somente ser aplicada uma vez, mais com ciclicidade. A Respondente 02 também se colocou a disposição para conversar com os gestores e liderança da organização que atua para a implantação do *roadmap* de forma diferenciada, com um tempo destinado da equipe de testes de softwares somente para atuação nas melhorias e, se possível, realizar também uma ciclicidade após um período de amadurecimento das práticas incrementadas na equipe.

Entrevista com Respondente 03

A terceira entrevista foi feita com profissional com experiência em desenvolvimento de aplicações e, atualmente exerce o papel de liderança técnica em uma equipe que atua com a criação de micro serviços e modelos de ciência de dados. Este profissional tem bastante conhecimento em desenvolvimento de aplicações e, uma preocupação muito grande com qualidade das entregas e, com a satisfação dos usuários.

As considerações feitas pelo Respondente03 foram muito entusiásticas, percebeu-se uma animação muito grande conforme ele via o *roadmap* e suas etapas e no valor agradado em relação ao seu consequente aumento da qualidade das aplicações desenvolvidas, além de uma capacitação para todo o time de desenvolvimento, de forma que todos seguiriam os mesmos padrões e princípios e práticas de testes de software.

Em relação a comentários sobre o RoMTeS, o respondente 03 elencou que, para que a organização permitisse que uma pessoa de fora aplicasse o *roadmap* para melhoria das práticas de teste de software ela deveria ter confiança e, seguir a uma série de padrões e procedimentos para que não houvesse nenhum vazamento de informações, ficando preocupado somente com a aplicação do *roadmap* em sua organização que, por ser muito hierárquica, havia a necessidade de solicitação de diversas aprovações para a implantação do RoMTeS.

Em relação as etapas, o respondente 03 esteve de acordo com todas as etapas e atividades descritas no *roadmap* e, levantou a possibilidade de expansão do *roadmap* para abranger também, práticas de integração contínua e relacionadas também ao desenvolvimento de software.

Ao final das análises da Respondente 03, foram feitas 3 questões em relação a utilidade, factibilidade e usabilidade do *roadmap* RoMTeS, bem como se havia algum ponto não mencionado no *roadmap*.

Para o primeiro item perguntando, a utilidade do *roadmap,* o Respondente 03 disse ter gostado do *roadmap* e que, inclusive, seria muito interessante que o *roadmap* abrangesse também a área de desenvolvimento de software integrada a testes, de maneira a incluir conceitos de SOLID e outros conceitos relacionados a modelagem de bando de dados estruturados e não estruturados. Ele comentou também sobre os resultados que a implementação *roadmap* trariam para as equipes e organizações em geral, que seriam muito boas e, trariam um aumento da qualidade das aplicações desenvolvidas.

Sobre a factibilidade do *roadmap*, o Respondente 03 comentou que o *roadmap* é objetivo e fácil de entender, em relação a usabilidade do *roadmap*, ele comentou sobre ser muito útil para a melhoria dos processos de teste e qualidade de software.

Ao final da entrevista, perguntei ao Respondente03 sua opinião de forma geral, se o *roadmap* faz sentido para organizações no sentido de servir com um guia orientador para melhoria do processo de teste de software e ela comentou que sim, achou interessante e, achou que o *roadmap*, em versões futuras, poderia abranger também o desenvolvimento de software e modelagem de dados estruturados e dados não estruturados.

APÊNDICE 03 - RELAÇÃO DO ROMTES COM O TMMi

Recomenda-se que, ao aplicar o roadmap de melhoria de teste de software RoMTeS, seja utilizado o Modelo Integrado de Maturidade de Teste TMMi como modelo para análise de nível de maturidade das equipes de desenvolvimento de software.

O modelo TMMi, como observado anteriormente, é composto por cinco níveis de maturidade em testes de software. Cada um dos níveis do TMMi é abordado no RoMTeS por meio de práticas de teste de software que podem ser distribuídas entre os níveis de maturidade do modelo TMMi.

Ao analisarmos o primeiro nível de maturidade do modelo TMMi, percebe-se que neste nível inicial a organização ainda não estabeleceu processos formais de teste e os testes são executados de maneira improvisada e sem planejamento prévio. Para este nível, não há ações correlacionadas às práticas de teste de software levantadas nesta pesquisa.

Ao analisarmos o segundo nível de maturidade do modelo TMMi, o nível Gerenciado, percebe-se que é neste nível que a organização começa a implantar processos de teste mais estruturados e documentados, embora ainda não estejam completamente integrados em toda a organização. Para este nível Gerenciado, recomenda-se a utilização de algumas práticas de teste de software levantadas nesta pesquisa como o desenvolvimento orientado a testes, aplicação de testes unitários e *mocks*, quando necessário, inclusão de análise estática de código, a existência de um ambiente específico para execução dos testes, as validações dos testes incluírem campos no banco de dados e retorno das aplicações. Recomenda-se também a utilização da pirâmide de testes para planejamento da quantidade de testes distribuídos por nível da pirâmide de testes, bem como a estimativa de tempo para planejamento e execução dos testes e, que os testes sejam baseados nos requisitos de negócio. Neste nível recomenda-se também que testes de unidade e testes funcionais sejam necessários para as entregas de software, bem como, que os testes estejam integrados na esteira de integração e entrega contínua (CI/CD) e que a esteira falhe com a não execução ou a execução com erros dos testes.

Ao analisarmos o terceiro nível de maturidade do modelo TMMi, o nível Definido, percebe-se que é neste nível em que a organização já estabeleceu processos de teste devidamente definidos, documentados e integrados em toda a estrutura organizacional. Para este nível Definido, recomenda-se a utilização de algumas práticas de teste de software levantadas nesta pesquisa como a aplicação de testes de mutação, testes regressivos e funcionais automatizados, além de testes não funcionais como testes de

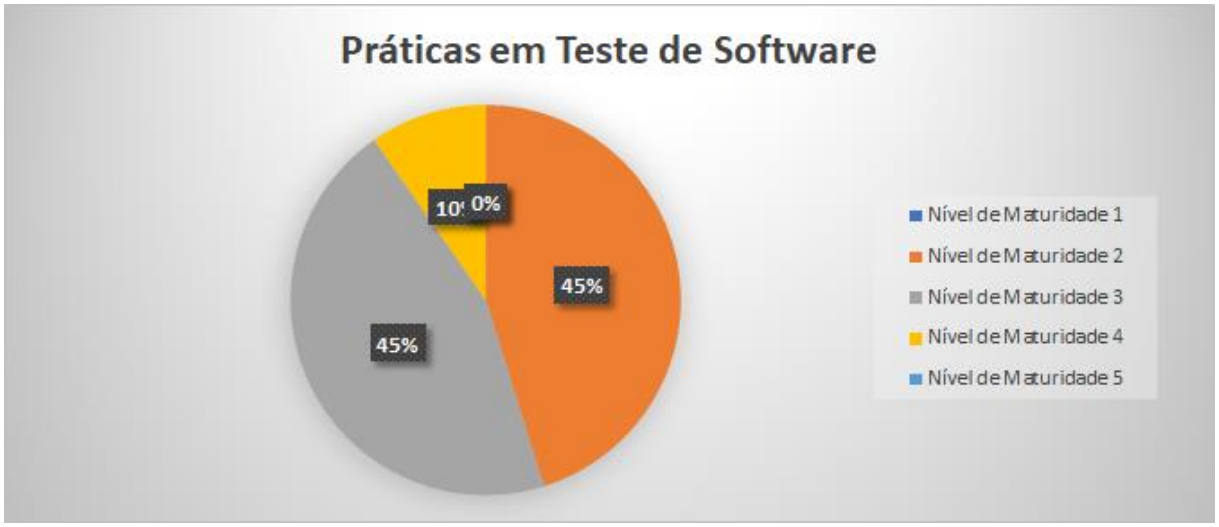
desempenho (performance) e de estresse (stress). Nesta etapa espera-se atividades como a gestão de ciclo de vida de aplicação e a integração dos testes na esteira de entrega contínua. Recomenda-se a execução de testes de integração com banco de dados ou outras aplicações e a documentação de cenários de teste de forma organizada e padronizada, assim como revisão por pares da documentação dos testes. Recomenda-se também a documentação de evidências de execução dos testes e defeitos encontrados, bem como a necessidade de testes não funcionais como parte da entrega de software.

Ao analisarmos o quarto nível de maturidade do modelo TMMi, o nível Medido (Gerenciado Quantitativamente), percebe-se que é neste nível em que a organização já possui métricas de desempenho de teste e está constantemente monitorando e aprimorando seus processos de teste com base nessas métricas. Para este nível Medido (Gerenciado Quantitativamente), recomenda-se a adoção de práticas como a execução de testes exploratórios, a utilização dos resultados dos testes como feedbacks e melhoria para o produto e, recomenda-se também, a execução frequente para os cenários de teste.

Ao analisarmos o quinto nível de maturidade do modelo TMMi, o nível Otimização, percebe-se que é neste nível em que a organização já alcançou um alto nível de maturidade em seus processos de teste e está constantemente em busca de melhorias e inovações para aprimorá-los ainda mais. Há a utilização de indicadores e métricas para medir e monitorar o desempenho do processo de teste como um todo. Em resumo, o nível otimizado representa a mais alta qualidade e excelência nos processos de teste da organização. Para este nível, por se tratar de um nível de maturidade máximo em teste de software, não houve a necessidade de levantamento de práticas de teste de software a serem incorporadas neste nível, uma vez que o objetivo do roadmap proposto é melhorar o nível de maturidade em teste de software.

Com base na categorização das práticas em teste de software com os níveis do modelo de maturidade TMMi, elaborou-se o Gráfico 26:

Gráfico 26: Práticas em teste de software e níveis de maturidade



Fonte: O Autor

Como observado no Gráfico 26, a maior quantidade de práticas de teste de software para melhoria de nível de maturidade está nos níveis 2 e 3 de maturidade com base no modelo TMMi.

Por meio do Gráfico 26 é possível visualizar que não há práticas de teste de software relacionadas ao nível de maturidade 1, o que significa que, por serem práticas relacionadas a melhoria do processo de teste de software e, este nível ser o nível inicial do modelo de maturidade, não houve a necessidade de especificação de práticas para este nível, uma vez que não há a possibilidade da melhoria de maturidade levando ao nível 1.

Percebe-se no nível de maturidade 2 um aumento significativo nas práticas, com um total de 14 práticas de teste de software, incluindo testes unitários, ambiente específico para execução de testes, testes regressivos, validação em status de retorno das aplicações, entre outros. Isto ocorre porque há a possibilidade da aplicação de melhorias de teste de software para que a equipe consiga evoluir do nível de maturidade 1 para o nível de maturidade 2 em teste de software. O mesmo ocorre no nível de maturidade 3.

O nível de maturidade 4 apresenta um total de 3 práticas de teste de software, práticas estas necessárias para o aumento do nível de maturidade da equipe de desenvolvimento para o nível 5 em maturidade em teste de software, de acordo o TMMi.

Por fim, no nível de maturidade 5, a organização já possui processos de teste altamente maduros e otimizados, não havendo a necessidade de inclusão de práticas para melhorar o nível de maturidade em teste de software.

É importante ressaltar que, embora nesta pesquisa haja uma recomendação das práticas de teste de software relacionadas a cada nível de maturidade da organização, cada organização possui processos e metodologias próprios, dependendo da organização em que o RoMTeS seja aplicado, haverá a necessidade da priorização de quais práticas de teste de software serão aplicadas a fim de se aumentar o nível de maturidade da organização/equipe de desenvolvimento em testes de software. Não necessariamente todas as equipes de desenvolvimento terão as mesmas necessidades em teste de software e, é necessário adequar as práticas de teste de software responsáveis por sanar as deficiências de cada equipe de desenvolvimento.

APÊNDICE 04 - ROTEIRO DE ENTREVISTA DA APLICAÇÃO DO ROMTES

01. Qual a sua percepção sobre o aumento do nível de maturidade em teste de software por meio da implantação do RoMTeS?

02. Qual sua percepção sobre a usabilidade do RoMTeS, ou seja, é um roadmap de difícil ou de fácil aplicação?

03. Qual sua percepção sobre a factibilidade do RoMTeS, ou seja, sua viabilidade?

04. Qual sua percepção sobre a utilidade do RoMTeS, ou seja, seu emprego em organizações que atuam com o desenvolvimento ágil de software utilizando a arquitetura de micro serviços?

05. Como você encherga a comercialização do RoMTeS?

06. De maneira geral, como foi sua experiência com a implantação do RoMTeS?

07. Você recomendaria a aplicação do Roadmap de Melhoria de Teste de Software para outras organizações e outras equipes que atuam com o ambiente de desenvolvimento de software?