
**Universidade do Estado do Rio de Janeiro**
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Isabelle Barbalho Oliveira de Souza

# Uma Proposta De Framework De Manutenção Evolutiva De Software Utilizando Práticas Ágeis

Rio de Janeiro
2021

Isabelle Barbalho Oliveira de Souza

# Uma Proposta De Framework De Manutenção Evolutiva De Software Utilizando Práticas Ágeis


Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof.ª Dra. Vera Maria Benjamim Werneck
Coorientador: Prof. Dr. Johnny Cardoso Marques

Rio de Janeiro
2021

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

S729 Souza, Isabelle Barbalho Oliveira de.
Uma proposta de Framework de manutenção evolutiva de software utilizando práticas ágeis/ Isabelle Barbalho Oliveira de Souza. – 2021.
123 f.: il.

Orientadora: Vera Maria Benjamin Werneck
Coorientador: Johnny Cardoso Marques
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Framework (Programa de computador) - Teses. 2. Software - Manutenção - Teses. I. Werneck, Vera Maria Benjamin. II. Marques, Johnny Cardoso. III.Título.

CDU 004.42

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica



_____ _____

Assinatura Data

Isabelle Barbalho Oliveira de Souza

**Uma Proposta De Framework De Manutenção Evolutiva De Software Utilizando Práticas Ágeis**

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 30 de novembro de 2021.
Banca Examinadora:

---

Prof.ª Dra. Vera Maria Benjamim Werneck (Orientador)
UERJ – Instituto de Matemática e Estatística

---

Prof. Dr. Johnny Cardoso Marques (Coorientador)
ITA – Instituto Tecnológico de Aeronáutica

---

Prof.ª Dra. Flavia Maria Santoro
UERJ – Instituto de Matemática e Estatística

---

Prof. Dr. José Maria Nazar David
UFJF – Departamento de Ciência da Computação

Rio de Janeiro
2021

## AGRADECIMENTOS

A Deus, pela minha saúde, família, força, amigos e conquistas.

À minha mãe, por sempre me apoiar nas minhas escolhas, ser minha melhor amiga, me ajudar a enxergar as oportunidades, me encher de forças, ser um exemplo de mulher, cuidar de mim e trabalhar muito para me proporcionar a oportunidade de estudar. Aos meus pais, por terem me proporcionado uma sólida estrutura familiar e sempre estarem presentes na minha vida.

Aos meus irmãos, Artur e Hugo, por sempre dar bons conselhos, me ajudar sempre que necessário, me incentivarem a estudar e serem meus melhores amigos. À minha cunhada Cinthya França, pelas conversas, bons conselhos durante todo o mestrado e ajuda no final com a dissertação.

Ao meu noivo Leonel que sempre me apoiou, me incentivou nos momentos mais difíceis e ajudou a tornar a minha rotina de trabalho e estudo menos cansativa.

À Rosa, por ser uma grande amiga, me apoiar e ser uma benção na minha vida. Obrigada pela dedicação e ajuda durante todos esses anos.

Ao meu amigo Yuri, pelas inúmeras conversas relacionadas ao mestrado e carreira.

À Prof.ª Vera Werneck, que me ensinou muito desde à faculdade e aceitou ser minha orientadora no mestrado. Sempre me apoiou e incentivou em todos os momentos da elaboração da dissertação. Obrigada por toda dedicação e compartilhamento de experiências que vou levar para sempre na minha carreira.

Ao Prof. Johnny Marques, que aceitou ser meu coorientador e por todo o ensinamento, compartilhamento de eventos acadêmicos e pelas valiosas dicas e revisões durante toda a elaboração da dissertação.

Aos professores participantes da banca examinadora, pela disponibilidade para a avaliação deste trabalho.

Aos meus amigos Carlos Luciano, Eduardo Martins, Lucas Sousa e Felipe Diniz pela amizade, companheirismo, conselhos, ensinamentos e incentivo.

À Suelen, por sempre me ajudar com dúvidas de latex e apoio durante as disciplinas.

A todos os participantes do estudo, que aceitaram contribuir com a pesquisa e disponibilizaram um pouco de seu tempo para responder o *survey*, fornecendo contribuições valiosas para este trabalho.

A persistência é o caminho do êxito.

*Charles Chaplin*

# RESUMO

SOUZA, Isabelle Barbalho Oliveira de. *Uma proposta de framework de manutenção evolutiva de software utilizando práticas ágeis*. 2021. 123 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2021.

Problemas encontrados durante a fase de manutenção de software é um tema importante discutido na área de desenvolvimento de software. As equipes de manutenção de software lidam com muitas questões, desde questões administrativas até questões técnicas. No entanto, as práticas ágeis permitem que muitas dessas dificuldades sejam minimizadas. Este trabalho tem como objetivo desenvolver um *framework* de manutenção evolutiva utilizando práticas ágeis, por meio de um estudo qualitativo baseado em um mapeamento sistemático da literatura e uma pesquisa com especialistas em desenvolvimento de soft- ware. A análise qualitativa dos dados foi realizada por meio da técnica de codificação aberta. A proposta final elaborada é resultado da comparação dos dois modelos gerados a partir do mapeamento sistemático da literatura e da pesquisa com especialistas, apresen- tando as práticas ágeis mais utilizadas na manutenção evolutiva de software. Uma prova de conceito é realizada para exemplificar e avaliar o uso do framework.

Palavras-chave: Manutenção ágil. Manutenção evolutiva. Práticas ágeis. Framework ágil.

# ABSTRACT

SOUZA, Isabelle Barbalho Oliveira de. *A proposed framework of software evolutionary maintenance using agile practices*. 2021. 123 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2021.

Problems found during the software maintenance phase are an important topic discussed in the software development area. Software maintenance teams deal with many issues, from administrative to technical. However, agile practices allow many of these difficulties to be minimized. This work aims to develop an evolutionary maintenance framework using agile practices, through a qualitative study based on a systematic literature mapping and a survey with software development experts. Qualitative data analysis was performed using the open coding technique. The final framework proposal elaborated is the merge of the two models generated from the systematic mapping of literature and the survey with experts, presenting the most used agile practices in the evolutionary maintenance of software. A proof of concept is performed to exemplify and evaluate the use of the framework.

Keywords: Agile maintenance. Evolutionary maintenance. Agile practices. Agile frameworks.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ACM | *Association for Computing Machinery* |
| ALM | *Application Lifecycle Management* |
| API | *Application Programming Interface* |
| ATD | Atividade de Desenvolvimento |
| ATR | Atividade de Requisitos |
| ATT | Atividade do Time |
| ATTE | Atividade de Testes |
| BDD | *Behavior Driven Development* |
| CAAE | Certificado de Apresentação de Apreciação Ética |
| CMMI | *Capability Maturity Model® Integration* |
| EF | Especificação Funcional |
| IEEE | Instituto de Engenheiros Eletrônicos e Eletricistas |
| PO | *Product Owner* |
| POC | Prova de Conceito |
| TDD | *Test Driven Development* |
| TI | Tecnologia da Informação |
| WIP | *Work in Progress* |
| XP | *eXtreme Programming* |

## SUMÁRIO

# INTRODUÇÃO

O processo de modificar um produto de *software* após a entrega, corrigindo falhas ou implementando novas equações funcionais, é chamado de manutenção de *software* (CHOUDHARI; SUMAN, 2012). A manutenção de *software* auxilia o desempenho, a confiabilidade e a adaptabilidade da solicitação de mudança no produto no ambiente modificado (CHOUDHARI; SUMAN, 2012).

A fase de manutenção é uma das fases que demanda grande parte de recursos financeiros e dependendo do *software*, recursos humanos e, consequentemente, gasta-se mais do que o esperado pois geralmente o orçamento previsto já foi utilizado ou está escasso (TARWANI; CHUG, 2016). Por isso, os custos com a fase de manutenção devem ser previstos já na fase de planejamento ou, em outros casos, ter um orçamento adicional para manutenção evolutiva do *software*.

Uma percepção comum da manutenção de *software* é que ela apenas corrige falhas, mas grande parte do custo dos projetos é dedicado à essa fase (VALLON et al., 2018). Existem diversas categorias de manutenção que ajudam a entender esse elevado custo da manutenção. Além disso, *softwares* que são desenvolvidos com modelo cascata apresentam pontos negativos que afetam diretamente no custo do projeto, são eles, (i) requisitos podem mudar depois da fase de *design*, (ii) entrega final do *software* é um risco pois ele não pode atender às expectativas do cliente, (iii) novas tecnologias podem surgir ao longo do desenvolvimento e diferentes das definidas na fase de *design* (VALLON et al., 2018).

Os pontos negativos citados aumentam o custo, pois será necessário ter disponibilidade da força de trabalho extra para desenvolvimento de novas funcionalidades não previstas na fase de *design*, retrabalho para adaptar novas tecnologias e retrabalho para execução de testes de regressão para garantir que as mudanças não impactaram em funcionalidades que já funcionavam (ALVES; MARTINS; PAULISTA, 2017) (KLOTINS; UNTERKALMSTEINER; GORSCHEK, 2018) .

Diante desse cenário e com a evolução dos modelos de desenvolvimento de *software*, a utilização de práticas ágeis é uma abordagem alternativa ao tradicional. Com as práticas ágeis, são realizadas pequenas entregas, de forma incremental. Assim, problemas que antes eram encontrados apenas na entrega final do projeto de *software*, são antecipados durante os *feedbacks* dos *stakeholders* sobre as entregas incrementais (WAZLAWICK, 2019a).

O desenvolvimento ágil de *software* é uma abordagem evolutiva, altamente colaborativa, disciplinada e focada na qualidade para o desenvolvimento e entrega do *software*, em que o produto é produzido em intervalos regulares para revisão e correção (DUKA, 2012). As práticas ágeis foram crescendo no final dos anos 90 devido a diversas vantagens, dentre elas (TARWANI; CHUG, 2016): (i) Concentra-se na relação da equipe com o cliente; (ii) O produto é entregue em pequenas iterações para garantir a qualidade e melhor

desempenho; (iii) As iterações deixam os requisitos mais maleáveis de serem alterados; (iv) Utilização de protótipos antes do lançamento do produto para o cliente aprovar; e (v) Monitoramento contínuo do usuário, facilitando na fase de manutenção.

Além das vantagens listadas, vale ressaltar que o uso de práticas ágeis reduz o ciclo de *feedback* entre a geração de uma ideia (por exemplo um requisito ou uma estratégia de *design*) e a realização dessa ideia. Esta prática não apenas minimiza o risco de mal-entendidos, como também reduz o custo de solucionar quaisquer erros, ou seja, a fase de manutenção (DUKA, 2012).

O maior desafio para a adoção de práticas ágeis era mudar o *mindset*[1] dos integrantes da equipe em relação ao ciclo de vida do *software*. Aos poucos, as vantagens das práticas ágeis ficaram mais populares e as equipes passaram a se adaptar e a adotar essas práticas em seus projetos de *software*. Essa medida também se refletiu nos clientes, que ficaram mais satisfeitos porque o trabalho de manutenção consome menos tempo e custo, além de melhorar a qualidade do produto (TARWANI; CHUG, 2016).

Apesar das vantagens e facilidades das práticas ágeis, um grande desafio de aplicá-las durante a fase de manutenção de *software* é a existência de tarefas de diferentes características, desde evolutivas a emergenciais, não podendo esperar o tempo da iteração para realizar atualizações no *software* (VALLON et al., 2018).

*Frameworks* têm sido propostos no desenvolvimento de software como soluções genéricas apoiando diversas práticas, métodos, atividades. Assim, para apoiar o uso de práticas ágeis na manutenção evolutiva, nessa pesquisa foi elaborado um *framework* de manutenção evolutiva de *software* utilizando práticas ágeis através da união de resultados de um mapeamento sistemático da literatura e de uma pesquisa qualitativa realizada com especialistas da área, de forma a minimizar o retrabalho com atividades que são encontradas na manutenção de *software* utilizando modelo cascata.

Como os princípios por trás da utilização de práticas ágeis para uma operação de manutenção são pouco pesquisados e precisam de esclarecimentos (HEEAGER; ROSE, 2015), pretende-se realizar uma análise teórica e prática a partir de um mapeamento sistemático da literatura e um *survey* com especialistas para identificar quais práticas vêm sendo mais utilizadas na manutenção evolutiva de *software*. Ao final da análise, um *framework* será elaborado com as práticas mais utilizadas na fase de manutenção evolutiva com o objetivo de auxiliar equipes de manutenção de software.

[1] É a maneira de pensar e de agir de acordo com o seu pensamento (SILVA; STRAUSS; MELLO, 2017).

### Objetivo Geral

Este trabalho tem como objetivo geral desenvolver um *framework* de manutenção evolutiva de *software* utilizando práticas ágeis, voltado para projetos ou produtos de *software*.

### Objetivos Específicos

Os objetivos específicos desta dissertação são:

- Analisar as práticas ágeis utilizadas em projetos e produtos de *software*;
- Identificar as vantagens na utilização de práticas ágeis na fase de manutenção;
- Elaborar um *framework* de manutenção evolutiva de *software* utilizando práticas ágeis;
- Realizar uma prova de conceito (POC) do *framework* elaborado.

### Contribuições Esperadas

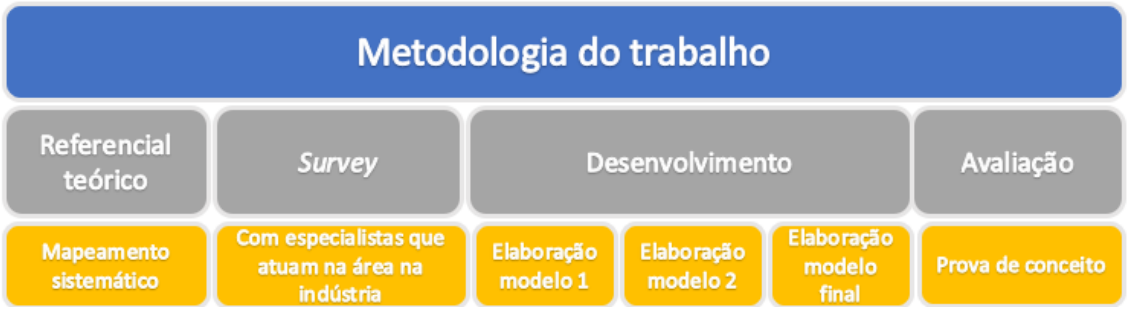
Esta dissertação visa fornecer informações necessárias para que pessoas que possuam interesse na área de atuação desse trabalho conheçam as práticas ágeis mais utilizadas na fase de manutenção de *software*.

### Metodologia e Organização do trabalho

Este trabalho foi desenvolvido utilizando análise de dados qualitativos a partir de mapeamento sistemático da literatura e *surveys*. A metodologia do trabalho está apresentada na figura 1 e foram definidas as seguintes atividades que estão apresentadas:

- Realizar um mapeamento sistemático da literatura para identificar as principais abordagens, técnicas, práticas, métodos e metodologias ágeis utilizadas na manutenção evolutiva de *software*;
- Realizar um *survey* com especialistas na área da indústria e analisar os dados coletados;
- Elaborar um modelo de *framework* através dos resultados obtidos no mapeamento sistemático da literatura;

Figura 1 - Metodologia do trabalho




Fonte: A autora, 2021.

- Elaborar um modelo de *framework* através dos resultados obtidos no *survey* com especialistas;
- Comparar os modelos criados e definir um *framework* final a partir das melhores práticas;
- Realizar uma POC com o *framework* elaborado.

O trabalho está dividido em seis capítulos que visam explicar os conceitos necessários para o desenvolvimento do *framework*, como ele foi desenvolvido e os resultados da POC.

O Capítulo 1 apresenta de forma geral os conceitos teóricos de manutenção de *software* e de metodologias de desenvolvimento de software ágeis e tradicional.

O Capítulo 2 apresenta o mapeamento sistemático da literatura realizado para analisar quais abordagens, técnicas, práticas, métodos e metodologias ágeis estão sendo utilizados em manutenção evolutiva.

O Capítulo 3 apresenta como foi realizado o *survey* com especialistas, a coleta de dados e a análise dos dados qualitativos utilizando a técnica de *coding*.

Os resultados obtidos dos Capítulos 2 e 3 foram utilizados para elaboração do *framework* de manutenção ágil que se encontra no Capítulo 4.

O Capítulo 5 apresenta uma POC de um cenário hipotético para apresentar como que o *framework* é utilizado.

No Capítulo 5.8.13 é apresentada a conclusão do trabalho que descreve as considerações finais, contribuições e sugestões de trabalhos futuros.

# 1 PRESSUPOSTOS TEÓRICOS

## 1.1 Manutenção de *software*

A manutenção de *software* é uma fase do ciclo de vida de *software* onde ocorre o processo geral de mudança em um sistema depois que ele é liberado para uso. As mudanças realizadas no *software* podem ter diferentes categorias, desde a correção de erros de codificação até mudanças mais extensas para correção de erros de requisitos do sistema, melhorias significativas para corrigir erros de especificação ou inserir novas funcionalidades no sistema (SOMMERVILLE, 2010). Essas alterações podem ser classificadas como (SOMMERVILLE, 2010): (a) manutenção corretiva, caracterizada pela correção de erros já mapeados; (b) adaptativa, inclui os esforços destinados a alterar o *software* para responder a um ambiente em mudança ou atualização do ambiente às tecnologias mais recentes; e (c) manutenção evolutiva, na qual funcionalidades são aperfeiçoadas ou criadas para melhor atender às necessidades dos usuários. Além das classificações citadas, Knippers (2011) também citou a manutenção preventiva que é composta de modificações com o objetivo de evitar falhas no futuro, ou seja, encontrar e corrigir erros que ainda não foram descobertos pelo time de desenvolvimento.

Figura 2 - Distribuição do esforço de manutenção


Fonte: Adaptado de Sommerville (2010)

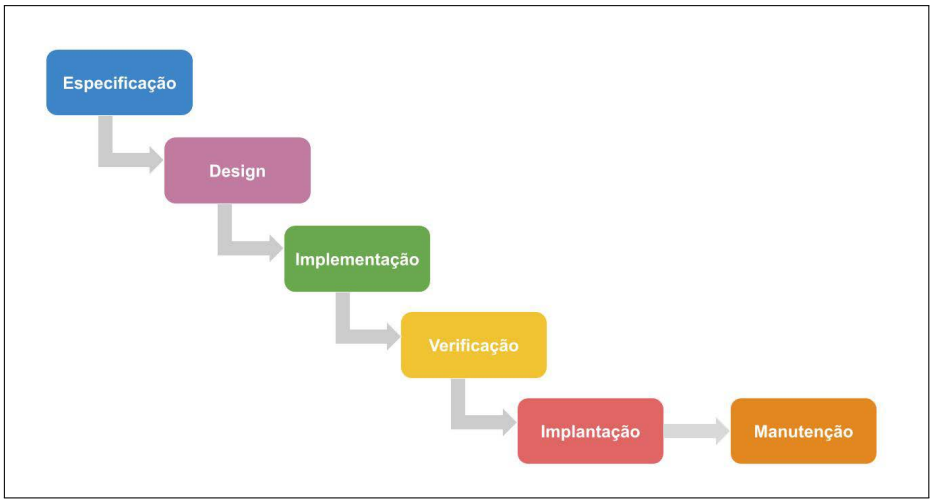
A manutenção de *software* ocupa uma grande proporção dos orçamentos de TI, geralmente dois terços do orçamento do desenvolvimento. Na Figura 2, pode-se observar que se gasta mais com adição de novos requisitos do que correção de *bugs* (SOMMERVILLE, 2010). A porcentagem de cada atividade varia de uma organização para outra,

mas a correção de defeitos não costuma ser a atividade de manutenção mais cara (SOMMERVILLE, 2010).

## 1.2 Modelos tradicionais

Durante muito tempo, o modelo Cascata foi considerado o modelo de desenvolvimento de *software* mais utilizado, com o processo fluindo sequencialmente a partir da fase de especificação, passando pelo *design*, codificação e testes antes do lançamento do *software*, assim como apresentado na Figura 3. Cada fase contém suas atividades definidas e resultados intermediários (KUHRMANN et al., 2016).

Figura 3 - Ciclo de vida de software - Cascata




Fonte: Adaptado de Stoica, Mircea e Ghilic-Micu (2013)

Verificando a necessidade de revisão e alteração dos resultados de fases anteriores, um modelo que permite a superposição entre fases e correções é o Sashimi (FILHO, 2003). Neste modelo, é possível realizar alterações de documentação/especificação durante a fase de implementação, conforme os erros forem sendo descobertos (FILHO, 2003).

Diferente dos modelos vistos anteriormente, no modelo Espiral, o produto é desenvolvido em uma série de iterações que permite a construção de produtos em prazos curtos (FILHO, 2003). Quando proposto, a ideia não era implementar ciclos iterativos, mas as iterações eram necessárias para identificar os riscos (WAZLAWICK, 2019b). A ideia do ciclo é iniciar pequenos projetos, verificando quais os principais riscos e ir expandindo para as outras fases de maneira que todos os riscos sejam identificados (WAZLAWICK, 2019b).

Um modelo variante do Espiral é o Prototipação Evolucionária, que permite uma melhor adaptação às mudanças de requisitos. Os requisitos que foram utilizados nos protótipos são detalhados progressivamente. Esse modelo apresenta alta flexibilidade e

uma boa visibilidade para os clientes (FILHO, 2003). Uma desvantagem do modelo Espiral e da Prototipação Evolucionária é que existe uma dificuldade de avaliação do quanto foi produzido efetivamente em cada fase (WAZLAWICK, 2019b).

O modelo de Entrega por estágios ou Implementação Incremental é uma variação bem estruturada do modelo de Prototipação Evolucionária, mas também é considerado uma variação do modelo de Cascata com Subprojetos (WAZLAWICK, 2019b). A diferença deste modelo para a Prototipação Evolucionária é que existe um cronograma e, portanto, este modelo torna-se bastante útil para equipes que possuem dificuldades com prazos (WAZLAWICK, 2019b). A entrega é realizada em liberações parciais do produto, aumentando a visibilidade do projeto para cliente (FILHO, 2003).

O modelo de Entrega Evolucionária é uma combinação da Prototipação Evolucionária com a entrega por estágios. Neste modelo, a equipe desenvolve um produto, o cliente visualiza e novas versões são criadas a partir do *feedback* do cliente (WAZLAWICK, 2019b). Este modelo pode tender tanto para Prototipação Evolucionária quanto para Entrega em Estágios, dependendo do *feedback* do cliente (WAZLAWICK, 2019b).

Diante de tantos modelos tradicionais, o ciclo de vida de *software* ágil surgiu com a ideia de não precisar esperar uma fase terminar para iniciar a outra. As fases são realizadas durante as iterações e sempre que o desenvolvimento de uma funcionalidade terminar, pode ser liberada para fase de verificação. Em cada iteração/ciclo, funcionalidades são entregues e ao final da última iteração/ciclo, é esperado que todas as funcionalidades priorizadas do *backlog*[2] tenham sido entregues (STOICA; MIRCEA; GHILIC-MICU, 2013).

Com as mudanças no mercado e avanços na tecnologia, muitos projetos que seguiam uma abordagem em cascata tiveram problemas, por exemplo, para desenvolver aplicativos que eram fundamentalmente diferentes do que se costumava a desenvolver por uma organização. Por isso, era muito difícil estimar o esforço necessário e, portanto, o tempo e o orçamento necessários para construí-lo (KUHRMANN et al., 2016). Além disso, em um mercado competitivo, a necessidade de reduzir custos e aumentar a velocidade no desenvolvimento exige a exploração de uma abordagem diferente do método tradicional de cascata (ZANOTTI; KAYLOR; DAVIDSEN, 2017).

A transição da utilização dos métodos de desenvolvimento de *software* tradicionais para os métodos ágeis é complicada e demorada. Para isso, é necessário que todos os envolvidos mudem sua mentalidade para se adequar a um método mais moderno. Isso é um desafio, visto que, como as equipes sempre utilizaram métodos tradicionais, era difícil mudar sua mentalidade (SURESHCHANDRA; SHRINIVASAVADHANI, 2008).

Nos métodos mais modernos, o projeto é dividido em iterações e o produto é liberado por pequenas entregas. A cada iteração, são realizadas atividades de análise,

[2] Backlog é um termo comumente usado nos modelos ágeis de gestão para se referir à lista das tarefas necessárias para a entrega de um produto (SANTOS, 2016).

design, codificação, teste de unidade e de integração e testes funcionais (TARWANI; CHUG, 2016). Antes do início da próxima iteração, é necessário garantir que a cobertura de testes definida foi realizada e os testes passaram.

Os métodos ágeis fornecem uma entrega mais rápida do produto, em um espaço de tempo mais curto, garantindo um alto nível de qualidade do *software*. E por essas vantagens que estão virando preferência no desenvolvimento de *software* (HUO et al., 2004).

Ao falar de processo de desenvolvimento de *software* tradicional e ágil, é necessário ressaltar os valores e princípios de cada um. A Tabela 1 descreve as diferenças entre o *mindset* ágil e o *mindset* tradicional (AITKEN; ILANGO, 2013) (STOICA; MIRCEA; GHILIC-MICU, 2013).

Tabela 1 - Tradicional x Ágil.

| **Tradicional** | **Ágil** |
|---|---|
| Dirigido por ferramentas e processos. | Dirigido por pessoas e colaboração. |
| Documentação formal e detalhada (ex: casos de uso, especificação funcional, etc). | Documentação no código. |
| Não aceita mudança de requisitos durante o andamento do projeto. | Os requisitos são alterados constantemente. |
| Entrega completa do *software* para o cliente. | Pequenas entregas para o cliente. |
| Fases longas com duração de meses. | Fases curtas com duração de 2 semanas a 2 meses no máximo. |
| Equipes não são auto gerenciáveis. | Equipes auto gerenciáveis. |
| Todos os requisitos decididos na fase de coleta de requisitos devem ser entregues no *software* final construído. | Entrega curta de *software* para recursos ou histórias individuais. Histórias de usuário podem evoluir com frequência. |
| Não há muita comunicação entre o time e o cliente/usuário durante o desenvolvimento do *software*. | Constante comunicação entre o time e o cliente/usuário final durante o desenvolvimento do *software*. |
| Testes executados no final do ciclo. | Testes executados no final de cada iteração. |
| Custos de manutenção evolutiva altos. | Custos de manutenção evolutiva baixos. |
| Remodelar o projeto é caro. | Remodelar o projeto é mais barato. |

Legenda: Diferenças entre características do modelo tradicional e do modelo ágil
Fonte: Adaptado de Aitken e Ilango (2013) e Stoica, Mircea e Ghilic-Micu (2013).

## 1.3 **Manifesto Ágil**

O termo "Metodologias Ágeis", também conhecido como "Métodos Ágeis", ficou conhecido em 2001, quando dezessete especialistas em processos de desenvolvimento de *software* encontraram um conjunto de princípios comuns que estavam trazendo valor para seus clientes. A partir deste compartilhamento, a Aliança Ágil foi criada e foi estabelecido o "Manifesto Ágil" (SOARES, 2004).

O "Manifesto Ágil" possui alguns conceitos que são chaves e que mudaram o *mind-set* do desenvolvimento de *software* (MANIFESTO, 2004): (i) Indivíduos e interações ao invés de processos e ferramentas - priorizando sempre as relações humanas e lembrando que não as pessoas da equipe são seres humanos; (ii) *software* executável ao invés de documentação - priorizar a documentação via código; (iii) Colaboração do cliente ao invés de negociação de contratos - fazer com que o cliente faça parte do time, colaborando e trabalhando em conjunto; (iv) Respostas rápidas a mudanças ao invés de seguir planos - ser adaptável à mudanças e respondê-las rapidamente.

O "Manifesto Ágil" prioriza os indivíduos, as entregas por interações, o *software* no modo executável, a colaboração do cliente e as respostas rápidas a mudanças. No entanto, ele não rejeita os processos, as ferramentas, documentação, negociação de contratos, ou planejamento, apenas demonstra que eles têm importância secundária (SOARES, 2004).

## 1.4 **Princípios Ágeis**

Os 12 princípios ágeis estabelecidos no "Manifesto Ágil" complementam os valores e formam os pilares sobre os quais são construídos os Métodos Ágeis (PRIKLADNICKI; WILLI; MILANI, 2014).

1. *"Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado:"* Este princípio resgata o maior objetivo que o time deve ter em mente que é entregar *software* funcionando e com qualidade, com base em iterações rápidas e contínuas, agregando valor ao negócio do cliente (PRIKLADNICKI; WILLI; MILANI, 2014).

2. *"Mudanças nos requisitos são bem recebidas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente":* Os métodos tradicionais são mais resistentes à mudanças, principalmente no final do projeto. Percebeu-se a ineficiência das práticas adotadas contra mudanças de escopo no decorrer do desenvolvimento e mudou-se a maneira de pensar (PRIKLADNICKI; WILLI; MILANI, 2014). Agilistas aceitam que as mudanças no escopo original são esperadas e bem-vindas para melhoria contínua do

*software*. É importante estar preparado para qualquer tipo de mudança que pode vir a ocorrer (PRIKLADNICKI; WILLI; MILANI, 2014).

3. *"Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo":* O maior objetivo desse princípio é entregar com frequência *software* funcionando para o cliente, agregando valor e confiança. Além disso, para responder ao princípio anterior de mudanças, é necessário realizar entregas pequenas para o impacto ser menor. Os períodos já predeterminados das entregas ajudam a equipe a prever cada vez melhor o quanto é capaz de produzir em cada ciclo.

4. *"Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto":* Um processo constante de colaboração entre clientes/partes interessadas e equipes de desenvolvimento facilita o sucesso do projeto. Essa interação direta com o cliente proporciona um fluxo constante de apresentação, discussão e *feedback*, que são muito importantes para manter a qualidade nas entregas (PRIKLADNICKI; WILLI; MILANI, 2014).

5. *"Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho":* Uma das maiores características das equipes ágeis é que elas são auto gerenciáveis e não existe uma pessoa para dar ordens e cobrar resultados. A comunicação direta e *feedbacks* são frequentes nos times ágeis (PRIKLADNICKI; WILLI; MILANI, 2014).

6. *"O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face":* Os problemas de comunicação existem em qualquer tipo de equipe, mesmo com tantas tecnologias, facilidades e conveniências. A conversa face a face é insubstituível, pois algumas sutilezas como gestos, expressões faciais e entonação da voz não podem ser transmitidas eficientemente por meios eletrônicos. Sendo assim, o manifesto ágil afirma que a troca de informação mais eficaz entre a equipe de desenvolvimento é frente a frente. Assim, quanto maior a frequência de conversas presenciais, menos conflitos surgirão (PRIKLADNICKI; WILLI; MILANI, 2014).

7. *"Software funcionando é a medida primária de progresso":* Este princípio afirma que o *software* funcionando é mais importante que uma documentação extensa, pois documentos e especificações possuem validades. De acordo com esse princípio, o *software* deve ser mensurado, principalmente, por meio da quantidade de *software* entregue e funcionando, que é o que mais importa para o cliente (MANIFESTO, 2004).

8. *"Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente:"* A produtividade de uma equipe ainda é vista pela quantidade de horas trabalhadas, um conceito herdado na indústria manufatureira. Com o objetivo de reverter tal cenário, este princípio prioriza a manutenção de ambientes que funcionem não em seus limites operacionais, mas em níveis nos quais sua sustentação seja viável (PRIKLADNICKI; WILLI; MILANI, 2014).

9. *"Contínua atenção à excelência técnica e bom design aumenta a agilidade":* Realizar um trabalho excelente no código faz com que se tenha um ambiente sustentável de alta produtividade, priorizando a entrega contínua do código e a agregação de valor. Um código bem executado com um projeto de qualidade elimina a necessidade de uma documentação exaustiva e reduz o retrabalho de atualizar a documentação (PRIKLADNICKI; WILLI; MILANI, 2014).

10. *"Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial":* Este princípio foca na identificação do que é realmente importante e trará valor de negócio e vantagem competitiva ao cliente. Este princípio é importante para o time realizar questionamentos como "Isso é realmente essencial?" ou "Como tornar isto mais simples?" (PRIKLADNICKI; WILLI; MILANI, 2014).

11. *"As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis":* Um time auto-organizado, composto por indivíduos de alta competência técnica e comportamental, se sente livre para desenvolver os melhores produtos (MANIFESTO, 2004).

12. *"Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo":* A cada iteração finalizada, deve-se refletir o que foi feito e aprender com o que funcionou e com o que não funcionou. Os primeiros ciclos são de adaptação e, portanto, são de muito aprendizado para toda a equipe. Conforme os ajustes são realizados, a tendência é que o ritmo de trabalho e a integração da equipe melhorem (PRIKLADNICKI; WILLI; MILANI, 2014).

## 1.5 **Práticas Ágeis**

As práticas ágeis são adotadas para servirem de guias e a falta delas leva à imprevisibilidade, erros repetidos e retrabalho. Durante o manifesto ágil e a elaboração dos 12 princípios ágeis, citados na Seção 1.4, um conjunto de práticas ágeis foram sendo reveladas. Esse conjunto de práticas ágeis tem como objetivo permitir a entrega rápida e de

alta qualidade do *software*. Existem diversos métodos ágeis e eles possuem práticas em comum, assim como práticas específicas. As práticas mais comuns são pequenas entregas, programação em pares, reuniões de planejamento, *daily meeting*[3], retrospectiva e cliente presente.

De acordo com Beck (2000), *"cada release deve ser tão pequena quanto possível, contendo os requisitos mais importantes para o negócio"*. Dessa maneira, entregas frequentes são realizadas, resultando em uma maior visualização para clientes e programadores e facilitando o aprendizado e a correção dos defeitos do sistema;

A programação em pares é a prática de um desenvolvedor ser responsável pela codificação e o outro pensar nos algoritmos e na programação torna o código mais simples, além do desenvolvedor observador conseguir identificar possíveis erros e falhas resultando na melhoria do código; As reuniões de planejamento geralmente são realizadas no início da iteração e tem como objetivo planejar a entrega da iteração com base nas histórias de usuário[4], contando com a colaboração de toda a equipe de desenvolvimento e do cliente (BECK, 2000);

A *Daily Meeting* é uma reunião diária na qual todos do time informam o que foi feito no dia anterior, o que vai ser feito no dia corrente e se existe algum impedimento interferindo no andamento das atividades. Esta reunião é muito importante para todo o time estar atualizado do andamento das tarefas, ter a troca de experiências e, caso algum membro esteja impedido, o responsável pelo projeto pode atuar;

A reunião de retrospectiva ocorre no final de cada *sprint*[5] e tem como objetivo reunir a equipe para avaliar o trabalho que foi realizado na entrega anterior. Nessa reunião, verifica-se se o trabalho se o trabalho atingiu os objetivos. Além disso, identifica pontos de melhoria para serem trabalhos nas próximas iterações, para aumentar a qualidade e produtividade (AUDY, 2015).

O cliente deve estar sempre presente nas atividades do time, e sempre que possível, dar *feedback* sobre o projeto e o trabalho do time (BECK, 2000).

## 1.6 **Métodos Ágeis**

No desenvolvimento de *software*, os métodos ágeis priorizam o valor que o projeto agrega e as interações entre as pessoas acima de prazos, custos ou atendimento ao escopo

[3] reunião diária

[4] é uma descrição concisa de uma necessidade do usuário do produto (ou seja, de um "requisito") sob o ponto de vista desse usuário. A História de Usuário busca descrever essa necessidade de uma forma simples e leve (K21, 2020)

[5] são eventos de duração fixa de um mês ou menos para criar consistência. Uma nova *Sprint* começa imediatamente após a conclusão da *Sprint* anterior (GUIDE, 2020)

que foi definido inicialmente (PRIKLADNICKI; WILLI; MILANI, 2014).

Os métodos ágeis possuem características adaptativas ao invés de preditivas. Com os métodos tradicionais, o processo de *software* é planejado em detalhes por um longo período, permanecendo em perfeito funcionamento, sem que haja grandes mudanças. Com o ágil, como as entregas são realizadas em curto espaço de tempo (MANIFESTO, 2004), ele tende a ser mais adaptativo.

Alguns métodos ágeis e suas principais práticas são abordados nas próximas subseções.

### 1.6.1 *eXtreme Programming* (XP)

Trata-se de um método eficiente, flexível e de baixo risco para equipes pequenas que desenvolvem requisitos dinâmicos ou em constante mudança. Bastante utilizado em projetos com requisitos vagos e com constante alteração, desenvolvimento de sistemas orientados a objetos, equipes pequenas e desenvolvimento incremental (TELES, 2014).

Pode-se dizer que o XP é um processo de desenvolvimento focado em garantir que o cliente receba o máximo de valor de trabalho da equipe de desenvolvimento (TELES, 2014). O XP baseia-se em valores como: **simplicidade, comunicação, *feedback* e coragem**, fazendo toda a equipe exercer uma comunicação rápida e eficaz entre as partes interessadas (TELES, 2005).

O valor comunicação contribui para que todos os detalhes dos projetos sejam tratados com atenção e agilidade. O XP procura assegurar que a comunicação ocorra da forma mais direta e eficaz possível e a simplicidade ensina o time a implementar somente o que é suficiente para atender a necessidade do negócio do cliente (TELES, 2014). Em conjunto com a comunicação e simplicidade, a cultura de *feedback* é utilizada para a equipe fazer generalizações quando necessárias, e como o desenvolvimento é incremental, mudanças para melhorar o *software* a partir dos *feedbacks* podem ser realizadas (TELES, 2014). Além disso, falhas nos sistemas podem existir, principalmente por se tratar de um desenvolvimento incremental e sujeito a mudanças. Por isso, a equipe precisa ser corajosa para enfrentar essas falhas e acreditar que as práticas e valores do XP auxiliam na evolução do *software* que está sendo construído.

#### 1.6.1.1 Práticas do XP

Beck (2000) afirma que o XP possui doze práticas que são essenciais para o processo e foram criadas nos ideais dos valores. As doze práticas são comentadas a seguir:

1. Jogo do planejamento: quando é realizado o planejamento das entregas e das ite-

rações com base nas histórias de usuário, contando com a participação de todo o time de desenvolvimento, inclusive o cliente. O jogo do planejamento se divide em planejamento da entrega e planejamento da iteração. O planejamento da entrega é a atividade em que o cliente vai apresentar as funcionalidades para todos do time, para que eles possam avaliar as dificuldades e preocupações. E o planejamento da iteração é quando a equipe recebe orientações através das histórias de usuário (BECK, 2000).

2. Pequenas entregas: realizar pequenas entregas do software funcionando com o propósito de agregar valor para o cliente.

3. Metáfora: Metáforas são usadas frequentemente durante o desenvolvimento de sistemas para que os clientes e desenvolvedores sejam capazes de estabelecer um vocabulário mais apropriado para o projeto, facilitando a compreensão e fornecendo um contexto único.

4. Projeto simples: é importante manter o projeto o mais simples possível, pois quanto mais inteligível, mais ágil ele se torna. Esta prática tem o objetivo de enfatizar que o projeto simples deve se concentrar em soluções descomplicadas e bem estruturadas para os problemas atuais (BECK, 2000).

5. Testes: utiliza-se muito a técnica de desenvolvimento guiado a testes (TDD). Além disso, a prática de testes de aceitação e testes unitários são frequentes (BECK, 2000).

6. Refatoração: técnica empregada na reconstrução do código, cujo principal objetivo é tornar o código mais fácil de utilizar e compreender (FOWLER, 2001). Na refatoração, vários passos são aplicados para melhorar o projeto do código existente, tornando-o mais simples e melhor estruturado sem alterar sua funcionalidade.

7. Programação em pares: mencionado na Seção 1.5.

8. Propriedade coletiva: em um projeto que utiliza o XP, todos os membros do time são responsáveis pelo conjunto, ou seja, no decorrer do projeto, todos possuem a abertura para realizar qualquer iniciativa que vá agregar valor a alguma parte do código (BECK, 2000).

9. Integração contínua: As equipes XP mantêm o sistema integrado com outros sistemas todo o tempo, sendo assim, o código das funcionalidades implementadas pode ser integrado várias vezes. Um modo simples de fazer isso é ter uma máquina dedicada para integração. O importante é que na integração as funcionalidades só podem ser agregadas se não houver erros, caso contrário eles devem ser corrigidos.

10. Semana de 40 horas: XP não recomenda horas extras, portanto, o aumento da carga de trabalho amplifica significativamente a possibilidade de ocorrência de erros. A sobrecarga de expediente é sintoma de sérios problemas na concepção do projeto (TELES, 2005).

11. Clientes devem estar presentes para auxiliarem nos testes de aceitação, definirem prioridades e histórias para as futuras entregas.

12. Padrões de codificação: Como o XP recomenda a propriedade coletiva de código, na qual todos podem alterar e fazer refatoração do código de qualquer parte dele a qualquer momento, então é mais do que necessário que se utilize padrões de codificação. (BECK, 2000).

### 1.6.2 Scrum

O Scrum estabelece um processo de desenvolvimento iterativo e incremental para gerenciamento de projetos ou produtos e desenvolvimento ágil de *software*. Não existe uma técnica de desenvolvimento de sistema específico para o Scrum, ele se concentra em como uma equipe deve trabalhar para produzir melhores resultados em um ambiente de mudança (ABRAHAMSSON et al., 2017).

No Scrum, existem papéis e responsabilidades que são importantes destacar:

- *Scrum Master*: é a pessoa responsável por garantir que as entregas sejam realizadas de acordo com as práticas, valores e regras do Scrum. Além disso, ele é responsável por garantir que os impedimentos sejam removidos para manter a produtividade e o bem estar da equipe (ABRAHAMSSON; OZA; SIPONEN, 2010).

- *Product Owner (PO)*: é a pessoa responsável pelo projeto, que gerencia, controla e torna visível o *Backlog* do Produto. Ele é selecionado em conjunto pelo *Scrum Master*, pelo cliente e pela gerência. Ele toma as principais decisões da entrega e deve estar sempre presente (ABRAHAMSSON; OZA; SIPONEN, 2010).

- *Scrum team*: O time Scrum é a equipe que tem autoridade para tomar decisões em relação a quais ações são necessárias para atingir o objetivo da *Sprint*. A equipe Scrum está envolvida na estimativa de esforço e na criação do *Sprint Backlog*, revisando a lista de *Backlog* do produto e analisando impedimentos que precisam ser removidos.

- Cliente/Partes interessadas: participa das tarefas relacionadas ao *Backlog* do produto.

Além dos papéis e responsabilidades, o Scrum possui práticas específicas para evitar problemas que geralmente são causados pela imprevisibilidade e complexidade (SCHWABER, 1997). A seguir, a descrição das cerimônias do Scrum é fornecida com base em Schwaber e Beedle (2002):

- *Sprint*: a equipe Scrum se organiza para produzir um novo incremento de produto executável em uma *sprint* que dura aproximadamente de duas a quatro semanas.
- *Product Backlog*: define tudo o que é necessário existir no produto final com base no conhecimento atual, ou seja, define o trabalho a ser realizado durante as *sprints*. Inclui uma lista priorizada e constantemente atualizada de requisitos comerciais e técnicos para o que está sendo construído ou aprimorado. Esta lista pode incluir recursos, funções, correções de *bugs*, defeitos, aprimoramentos solicitados e atualizações de tecnologia. O PO é responsável pelo *Backlog* do Produto.
- Estimativa de esforço: é um processo iterativo, no qual as estimativas dos itens do *Backlog* são detalhadas quando mais informações estão disponíveis sobre as tarefas a serem realizadas. A equipe Scrum é responsável por fazer a estimativa do esforço.
- *Sprint Planning meeting*: a reunião de planejamento da *sprint* é uma reunião organizada pelo PO, na qual todo o time participa. O PO apresenta as histórias que são prioridades para serem desenvolvidas e o time avalia e decide sobre os objetivos da *sprint* e as funcionalidades que podem ser desenvolvidas na *sprint*.
- *Sprint Backlog*: é o ponto de partida para cada *sprint*. É uma lista de itens do *backlog* do produto selecionados para serem implementados na próxima *sprint*.
- *Daily Scrum meeting*: são reuniões diárias organizadas para acompanhar o progresso da equipe.
- *Sprint Review meeting*: ocorre no último dia da *sprint*. A equipe Scrum e o *Scrum Master* apresentam os resultados para a gerência, clientes, usuários e o PO em uma reunião. Os participantes avaliam o incremento do produto e decidem sobre as próximas atividades.
- Retrospectiva: Ocorre no final da *sprint*. A equipe se reúne para pontuar os aspectos positivos, negativos e sugestões de melhorias para as próximas *sprints*.

No processo Scrum, existem três fases: pré-planejamento, desenvolvimento e pós-planejamento. Na fase de pré-planejamento, os requisitos são representados no *backlog* e, logo em seguida, são classificados de acordo com a prioridade identificada para cada um. Nesta fase é que se define os participantes da equipe, a avaliação de aprimoramento de conhecimento técnico, as ferramentas que serão utilizadas e os prováveis riscos para o

desenvolvimento. A fase é concluída com uma proposta de arquitetura de *software* e as alterações futuras devem ser descritas no *backlog* (ABRAHAMSSON et al., 2017).

Durante o período de desenvolvimento, os riscos identificados na fase de pré-planejamento são detalhados e devem ser monitorados ao longo do ciclo de vida do desenvolvimento, avaliando seus impactos. A cada ciclo iterativo, o *software* é aperfeiçoado com as novas funcionalidades implementadas. Os ciclos são desenvolvidos de forma tradicional considerando a análise, o *design*, a implementação e os testes (ABRAHAMSSON et al., 2017).

Na etapa de pós-planejamento acontece fase dos testes finais e da documentação de requisitos. A equipe se reúne para analisar e validar o sistema desenvolvido e o mesmo é apresentado ao cliente/partes interessadas (ABRAHAMSSON et al., 2017).

### 1.6.3 Kanban

O Kanban possui origem japonesa e significa "Cartão" ou "Sinalização". Como o Japão ficou devastado com o fim da segunda guerra mundial e com a crise econômica, foi necessário buscar estratégias para reduzir os custos e aumentar a produtividade. A partir de então, a empresa Toyota criou uma técnica que foi denominada "Kanban", onde colocava-se cartões coloridos para ajudar a controlar os estoques e a produção (SILVA, 2018).

O Kanban é um método ágil que não possui iterações. Ele desmembra as fases de planejamento, da priorização, do desenvolvimento e da entrega do projeto, facilitando o intercâmbio das atividades de cada uma das fases (MOURA, 1999). Além disso, MOURA (1999) afirma que o método ágil Kanban é um dos métodos de desenvolvimento de *software* menos prescritivos e possuem três preceitos: (i) Visualizar o fluxo de trabalho; (ii) Limitar o trabalho em progresso; e (iii) Gerenciar e medir o fluxo.

O Kanban tem como principal objetivo avaliar o trabalho em progresso (WIP) (ARRUDA, 2012), a partir da visualização das tarefas do quadro que é possível avaliar o progresso. O Kanban possui muitas vantagens, dentre elas: (i) a equipe pode realizar entregas a qualquer momento; (ii) o desenvolvimento fica transparente já que é possível visualizar o fluxo de trabalho; (iii) não há preocupações com iterações e tempo de *sprint*, como em outros métodos (ARRUDA, 2012).

## 1.7 **Pesquisa Qualitativa**

A ciência muitas vezes se baseia em dados quantitativos, nos quais o raciocínio depende fortemente de atributos lineares, medições e análises estatísticas (STAKE, 2016).

No entanto, em uma variedade de contextos, a ciência também pode usar dados qualitativos, cujos aspectos estão relacionados à experiência pessoal, intuição e ceticismo, usados para ajudar a refinar teorias e experimentos (STAKE, 2016).

A pesquisa que envolve dados qualitativos tende a gerar resultados mais ricos em conteúdo; entretanto, geralmente é mais trabalhoso e exaustivo alcançá-los (SOUZA; SCHOTS, 2018). Os métodos de pesquisa qualitativa variam, dentre eles, existe a observação participante, a entrevista discursiva, o grupo focal (*focus group*) e o *survey* (CARDANO, 2017). Dentre os diversos métodos, este trabalho utiliza a técnica *survey* com perguntas abertas.

### 1.7.1 Survey

O método de pesquisa *survey* é baseado em investigação e compreensão das informações coletadas (SILVA et al., 2019). Este tipo de pesquisa pode ser descrita como a obtenção de dados ou informações sobre características, ações ou opiniões de determinado grupo de pessoas, por meio de um instrumento de pesquisa, normalmente um questionário (SILVA et al., 2019). O *survey* é muito utilizado em pesquisas exploratórias e descritivas, indicada para quando se deseja responder questões do tipo “o que”, “porque”, “como” e “quanto”.

### 1.7.2 Análise de dados qualitativos

A análise qualitativa enfatiza como os dados se encaixam como um todo, reunindo contexto e significado. Existem muitas abordagens, mas uma delas é simplesmente usar as perguntas da pesquisa para agrupar seus dados e, em seguida, procurar por semelhanças e diferenças (TOLLEY et al., 2016).

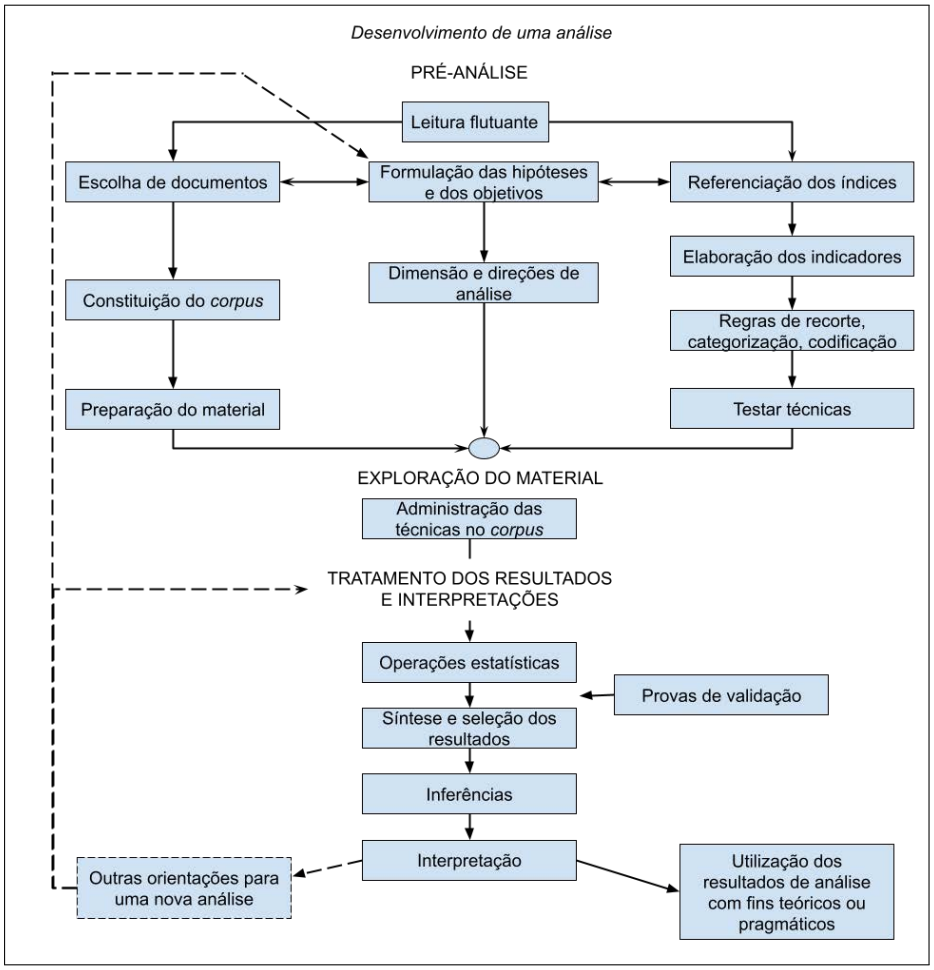
Nessa etapa da pesquisa, são realizados o tratamento, a inferência e a interpretação dos resultados, consistindo na captura dos conteúdos previamente coletados (entrevistas, documentos e observação) (SILVA; FOSSÁ, 2015). Durante a análise, os pesquisadores geralmente realizam o seguinte processo: (i) realiza a leitura flutuante do material coletado, tendo o primeiro contato com o material coletado; (ii) utiliza técnica de *coding* para formulação de categorias de análise; (iii) agrupa os trechos em categorias comuns; (iv) agrupa as categorias comuns em categorias menores (se necessário); e (v) realiza tratamento dos resultados, inferência e interpretação.

A etapa de análise dos dados é muito importante pois tudo que foi coletado através dos estudos empíricos será transformado em interpretações. É uma fase de reflexão do trabalho investigativo, constituindo em um processo árduo (CHARMAZ, 2006).

Na pesquisa qualitativa, a análise de dados difere da pesquisa quantitativa porque o pesquisador precisa usar a criatividade e a sensibilidade. A análise qualitativa não ocorre de forma linear, pois é necessário organizar e extrair significados dos dados que às vezes parecem não ter valor (TOLLEY et al., 2016).

Neste estudo, optou-se por utilizar algumas etapas da técnica proposta por (BARDIN, 1977), uma vez que é a obra mais citada em estudos qualitativos. Essas etapas, exibidas na Figura 4, estão organizadas em três fases:

Figura 4 - Desenvolvimento de uma análise




Legenda: Apresenta o desenvolvimento completo de uma análise de dados.
Fonte: Adaptado de (BARDIN, 1977)

- Pré-análise: envolve a preparação para estudar o material;
- Exploração do material: inicia-se após a pré-análise, exigindo a seleção dos dados e a divisão em categorias;
- Tratamento de resultados e interpretações: trata os dados usando técnicas para alcançar resultados significativos. As operações estatísticas podem ser usadas para

sintetizar e selecionar os resultados. Se possível, testes de validação são realizados. Assim, são feitas inferências e interpretações para finalizar a etapa do tratamento.

### 1.7.3 Codificação (*Coding*)

Segundo Creswell (2015), *coding* é um processo de análise de dados qualitativos de texto que separa os dados para analisar o que eles representam antes de reunir os dados de forma significativa. Este tipo de ferramenta oferece facilidades excepcionais para o armazenamento, gerenciamento e manipulação de dados, permitindo trabalhar com grande volume de dados no processo manual, facilitando a organização dos dados e reduzindo a ambiguidade dos dados coletados (BLAIR, 2015). O tempo economizado com a automação de algumas funções pode ser dedicado às fases de reflexão e processos analíticos (ELLIOTT, 2018).

*Coding* é uma técnica muito utilizada na pesquisa qualitativa, é considerado fundamental no processo analítico e das maneiras pelas quais os pesquisadores dividem seus dados para fazer algo novo (ELLIOTT, 2018). Elliott (2018) e Blair (2015) também entendem que a utilização do *coding* economiza tempo e facilita o trabalho de análise, devido às características que aproximam o pesquisador dos dados originais e codificados simultaneamente.

Seaman (1999) afirma que para entender a transformação dos dados que ocorre durante o *coding*, é necessário entender um equívoco comum sobre a diferença entre dados quantitativos e qualitativos. Como apontado anteriormente, a objetividade ou subjetividade dos dados é completamente ortogonal aos métodos qualitativos ou quantitativos. O processo de *coding* transforma dados qualitativos em dados quantitativos, mas isso não afeta sua subjetividade ou objetividade (CARROLL; ROTHE, 2010).

As categorias que são criadas pelo pesquisador podem ser criadas antes e durante a análise. Categorias criadas antes, ou "a priori", estão relacionadas a hipóteses de pesquisa pré-definidas, ou seja, ideias pré-concebidas sobre os fenômenos que estão sendo estudados. As categorias que surgem durante a análise, ou "emergentes", são categorias que emergem dos dados, ou seja, representam as "novidades" criadas a partir da análise dos dados (MORAES; GALIAZZI, 2006). Vale ressaltar que mesmo a técnica sendo de caráter qualitativo, é possível o pesquisador identificar dados qualitativos que podem ser transformados em quantitativos por meio do *coding*. Por exemplo, ao analisar a frase "*José, Maria e Hugo foram os únicos alunos que compareceram à conferência*", um resultado dessa transformação poderia ser: *numberParticipants = 3*.

Independentemente de usar *coding*, as vantagens de se trabalhar com dados qualitativos e quantitativos devem ser destacadas. Combinar esses dados torna os resultados mais ricos em detalhes.

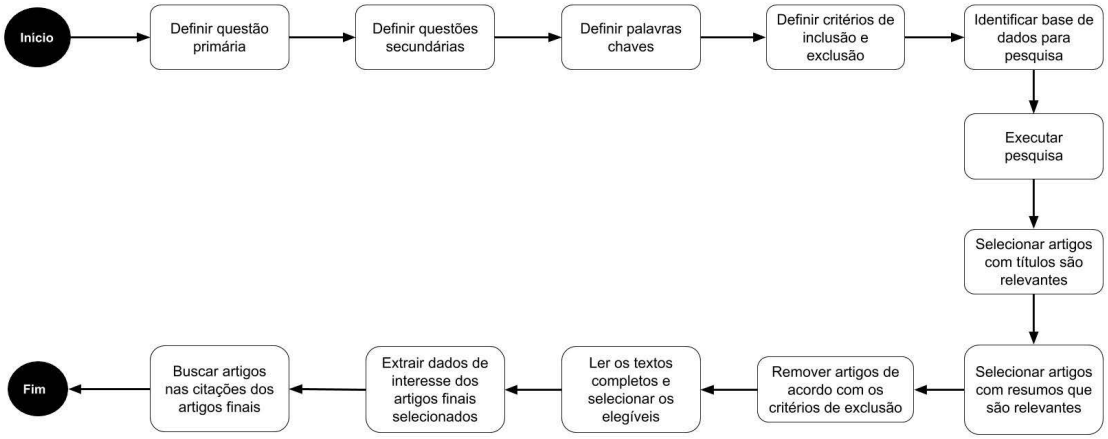
Quando a técnica de *coding* é realizada manualmente, exige bastante tempo do pesquisador. Para apoiar a análise qualitativa dos dados usando a técnica de *coding* (STRAUSS; CORBIN, 1998), uma ferramenta foi escolhida para ser utilizada, ela foi desenvolvida pela autora do texto em colaboração com alguns outros membros do time de pesquisa MARVEL-SE [6] (TEAM, 2018). Essa ferramenta tem por objetivo codificar trechos de textos durante a análise qualitativa.

[6] Mining, Analytics, Reuse, Visualization, Evolution, and Learning in *software* Engineering

# 2 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Durante a fase de planejamento, o protocolo da pesquisa foi produzido para realização do mapeamento sistemático e a partir do planejamento concluído, o processo da Figura 5 foi realizado passo a passo até a obtenção dos resultados. Este protocolo foi elaborado a partir do trabalho de (KITCHENHAM; BUDGEN; BRERETON, 2011).

Figura 5 - Processo do mapeamento sistemático




Legenda: Passo a passo do processo do mapeamento sistemático da literatura.
Fonte: A autora, 2020.

A equipe responsável pelo mapeado sistemático foi constituída por pessoas com diferentes experiências e conhecimento na área de agilidade e engenharia de *software*. Dentre elas, um professor doutor com conhecimento avançado em normas para desenvolvimento de sistemas e *software*, professora doutora com atuação em diversas áreas, como por exemplo, engenharia de requisitos, engenharia de *software* orientada a agente, métodos de desenvolvimento e qualidade de *software* e uma aluna de mestrado em ciências computacionais com experiência na indústria de desenvolvimento de *software* com atuação na área de coordenação de projetos e produtos digitais de times ágeis.

## 2.1 Especificação das questões de pesquisa

O principal foco do mapeamento sistemático da literatura é responder algumas questões estabelecidas previamente durante a fase de definição de questões de competência. A Tabela 2 apresenta a questão primária, que se refere ao estudo principal e ao parâmetro para as questões secundárias e também as questões secundárias e seus obje-

tivos. Primeiro, identificar como está sendo a transição do processo de desenvolvimento de *software* tradicional para o ágil (QS1), onde novos requisitos são incorporados ao *software*. As vantagens e desafios do uso da abordagem ágil são explorados em QS2 e QS3, respectivamente.

Tabela 2 - Questões da Pesquisa

| **Questão da pesquisa** | **Motivação** |
|---|---|
| **Questão Primária:** Como as metodologias ágeis têm sido utilizadas na manutenção evolutiva? | Identificar as principais abordagens, técnicas, práticas, métodos e metodologias ágeis utilizadas na manutenção evolutiva de *software*. |
| **QS1:** Como tem sido a transição da abordagem tradicional para a ágil? | Identificar como tem sido a mudança do modelo de desenvolvimento tradicional para o ágil. |
| **QS2:** Quais as vantagens do uso da abordagem ágil em manutenção? | Identificar as vantagens em utilizar práticas ágeis na manutenção de *software*. |
| **QS3:** Quais os desafios do uso da abordagem ágil em manutenção? | Identificar os desafios encontrados em utilizar práticas ágeis na manutenção de *software*. |

Fonte: A autora, 2020.

## 2.2 **Fonte de dados**

A pesquisa de artigos foi realizada nas bibliotecas mais relevantes da área da informática (ACM Digital library, IEEE) e no Google Acadêmico. Embora aconteçam muitas repetições entre as fontes, há publicações que aparecem exclusivamente em uma fonte. As publicações encontradas na biblioteca digital da ACM e IEEE, que já haviam sido extraídas do Google acadêmico, foram desconsideradas.

## 2.3 **Termos de Busca**

De acordo com as questões de pesquisa, os termos de busca incluem as palavras "ágil" (*"agile"*) e "manutenção evolutiva" (*"evolutionary maintenance"*) por se tratar da utilização de práticas ágeis na manutenção evolutiva. A string de busca utilizada na pesquisa foi (("EVOLUTIONARY MAINTENANCE" OR "MAINTENANCE") AND ("AGILE" OR "AGILITY")) OR (("MANUTENCÃO EVOLUTIVA" OR "MANUTENÇÃO") AND ("ÁGIL" OR "AGILIDADE")).

## 2.4 **Critérios de Busca**

A pesquisa considera apenas trabalhos que utilizam práticas ágeis e publicados a partir de Janeiro 2010. Essa opção foi motivada para focar nas práticas que estão sendo utilizados atualmente, evitando o risco de selecionar trabalhos considerados em desuso. Além disso, os seguintes critérios foram adotados: (i) Estudos que relatam problemas, desafios, abordagens, práticas, técnicas, métodos e metodologias ágeis em manutenção evolutiva de *software* utilizando metodologias ágeis; (ii) Estudos que descrevam contribuições de modelos de manutenção de *software* evolutiva utilizando práticas ágeis; (iii) Estudos publicados em revistas, eventos, livros na área de computação que estejam indexados nas bases de estudos; (iv) Estudos publicados até a presente data; e (v) Estudos escritos em inglês e português.

## 2.5 **Revisão e Seleção dos Artigos**

A busca por trabalhos considerou o período compreendido entre janeiro de 2010 e outubro de 2019. A partir dos resultados obtidos com as strings de buscas na fonte de dados, os seguintes passos foram executados: (i) Seleção dos trabalhos através da avaliação dos títulos; (ii) Seleção dos trabalhos através da avaliação dos resumos; (iii) Remoção dos trabalhos de acordo com os critérios de busca; (iv) Seleção através da leitura rápida do texto; e (v) Seleção através da leitura completa do texto. Considerando todas as fontes de busca e os passos (i) a (iii) executados, a pesquisa retornou 25.117 trabalhos, divididos conforme a Tabela 3.

Tabela 3 - Quantidade de publicações

| **Fonte de busca** | **Início** | **Após passo i** | **Após passo ii** | **Após passo iii** |
|---|---|---|---|---|
| ACM | 5245 | 60 | 21 | 20 |
| Google Acadêmico | 17200 | 25 | 10 | 8 |
| IEEE | 2672 | 34 | 14 | 13 |
| *Snowballing* | 0 | 0 | 0 | 5 |
| Total | 25117 | 119 | 45 | 46 |

Legenda: Quantidade de publicações depois dos passos definidos no processo do mapeamento sistemático.

Fonte: A autora, 2020.

Para ampliar as possibilidades de retornar mais artigos relevantes ao tópico de pesquisa, foi utilizado o método de *snowballing* durante a leitura dos artigos. Cinco artigos foram utilizados a partir do método, respeitando os critérios de busca previamente

estabelecidos. Este método consiste em procurar as referências de artigos incluídos no trabalho para identificar trabalhos que potencialmente sejam de interesse para a pesquisa.

## 2.6 **Ameaças à validade**

As ameaças à validade desse estudo incluem, dentre elas, que mais fontes de engenharia de *software* e computação poderiam ser pesquisadas na expectativa de encontrar trabalhos relevantes sobre manutenção ágil *software*. Além das fontes que não foram incluídas no mapeamento sistemático, podem existir relatos sobre manutenção ágil em fontes cinzentas como blogs e postagens em redes sociais que não foram coletados.

## 2.7 **Resultados**

Ao se realizar a busca foram encontradas 25.117 publicações. Na etapa de análise dos títulos foram eliminadas 24.998 publicações, restando 119 publicações. Ao se analisar os resumos das 119 publicações descartaram-se 74 publicações e restaram 45. Ao realizar uma leitura rápida dos artigos, eliminou-se 4 artigos que não atendiam aos critérios de busca, restando 41 artigos. Com método de *snowballing* incluiu-se mais 5 artigos. Esses artigos foram lidos na íntegra e apenas 37 foram aproveitados.

O resultado resumido dos artigos selecionados está exibido na tabela 15, onde foram destacados as práticas ágeis e quais as vantagens e desafios da utilização das práticas ágeis em manutenção evolutiva de *software*. A questão primária e questão secundária (QS1) da seção 2 são respondidas a partir da análise das três colunas da tabela 15 e na subseção 2.7.1. As questões (QS2) e (QS3) são respondidas a partir da análise das colunas 2 e 3 da tabela 15, respectivamente e seções 2.7.2 e 2.7.3.

### 2.7.1 Abordagem Ágil na Manutenção de *Software*

A manutenção de *software* é tratada separadamente do desenvolvimento de *software* pois as características de cada fase são diferentes. Muitos métodos, ferramentas, técnicas e modelos de processos de desenvolvimento não são utilizados para manutenção de *software* (PINO et al., 2012).

Como a manutenção é uma atividade tradicionalmente orientada por um plano, a ideia de ser ágil não parece funcionar (OMANOVIC; BUZA, 2013). Entretanto, a cada dia, as pessoas estão aderindo mais a ideia do ágil e manutenção evolutiva como algo que tem vantagens em relação a requisições de mudanças de tamanho médio. Caso existam

tarefas de tamanho médio, OMANOVIC; BUZA (2013) afirmam que um desenvolvedor pode realizar todas as atividades desta tarefa. Caso a tarefa seja de tamanho grande, pode-se colocar mais de um desenvolvedor para resolvê-la.

Na metodologia ágil, a manutenção pode ser vista como um projeto ágil sem previsão de término, onde o *backlog* do produto está em constante mudança com novas histórias de usuário sendo adicionadas à lista. As histórias de usuários são enviadas e priorizadas pelo cliente (OMANOVIC; BUZA, 2013). Omanovic e Buza (2013) afirmam que na manutenção, a implementação de cada história do usuário não é incondicional, o cliente precisa aprovar se o preço daquela implementação de mudança é viável. Histórias de usuário são implementadas e removidas da lista.

Pino et al. (2012) apresentam o Agile_MANTEMA, é uma proposta de metodologia para manutenção de *software* em pequenas organizações. Essa proposta é baseada no modelo do MANTEMA, que foi desenvolvido para projetos e serviços de manutenção de médio e grande porte. Dentre os quais se destacam os terceirizados, como, por exemplo, os relacionados a bancos ou administração pública (PINO et al., 2012). O MANTEMA segue um serviço de manutenção baseado na norma ISO 12207:2002 (www.iso.org), com uma definição muito elaborada que contém: fluxo de atividades e tarefas a serem executadas, modelo de funções e organizações participantes, entradas e saídas de atividades, e uma lista de técnicas recomendadas para cada atividade, juntamente com uma revisão dos tipos de manutenção (PINO et al., 2012).

O Agile_MANTEMA incorpora o gerenciamento ágil de projetos utilizando o método Scrum. A proposta define detalhadamente uma estratégia de manutenção que informa o que deve ser realizado, quando, como e por quem, procurando orientar passo a passo o processo de manutenção de *software* (PINO et al., 2012). O Agile_MANTEMA foi introduzido para ajudar pequenas organizações na manutenção de seus produtos e para fornecer serviços ao cliente (PINO et al., 2012). Os métodos ágeis tradicionalmente utilizados nas pequenas organizações, com o tempo, passaram a ser utilizados nas organizações de médio e grande porte (QURESHI, 2012).

Knippers (2011) afirma que utilizar práticas ágeis faz com que as fases de mudanças seja mais rápida, permitindo alterar os requisitos do sistema quando necessário. Embora um *design* simples e refatorações constantes visem garantir que a alteração do *design* do sistema possa ser feita rapidamente, não pode-se descartar que essa tarefa exige esforço (KNIPPERS, 2011). Em particular, a adaptação regular ao ambiente se traduz em trabalhos de manutenção na categoria adaptativa.

O trabalho Jr e Dantas (2019) descreve uma abordagem para lidar com projetos que são submetidos a mudanças e precisam de uma resposta rápida. A abordagem utilizada é baseada em princípios e conceitos da área de manutenção e evolução de *software*, considerando que tipos de demandas diferentes devem ser tratadas através de estratégias diferentes, essas demandas são classificadas em sustentação e evolução (JUNIOR; DAN-

TAS, 2019). A Sustentação fica responsável por acesso a dados de relatórios, exportação de dados e auditoria de eventos (JUNIOR; DANTAS, 2019). A Evolução fica responsável por novas funcionalidades, adaptações arquiteturais ou tecnológicas, pequenos aprimoramentos e correção de defeitos (JUNIOR; DANTAS, 2019). Eles se basearam nas práticas e conceitos dos métodos ágeis Scrum, Kanban e *Lean Software Development* com o objetivo de otimizar e aprimorar o processo de manutenção de sistemas (JUNIOR; DANTAS, 2019).

Rehman et al. (2018) utilizaram o Scrum para criar um modelo de manutenção de *software*. Neste modelo, requisições urgentes do cliente eram realizadas em paralelo com as atividades da *sprint* regular. O planejamento da manutenção envolve a execução de tarefas da *sprint* e requisições emergenciais. A participação ativa do cliente durante as *sprints*, a atualização da documentação por *sprint* e testar sempre antes de entregar são práticas que Rehman et al. (2018) enfatizam em seu modelo.

O trabalho de Ahmad et al. (2016) apresenta sobre a transição de times de manutenção de *software* do Scrum para o Kanban. O trabalho realizou um estudo de caso com dois times de manutenção de *software* e ambos perceberam que as *sprints* do Scrum não se encaixavam no processo de manutenção que eles tinham, pois existiam tarefas imediatas que não podiam esperar a data de lançamento (AHMAD et al., 2016). Além disso, o cliente não estava participando muito do processo. Como o Kanban promove visibilidade e aumenta a comunicação e colaboração do time durante o processo de desenvolvimento das tarefas, os times se adaptaram melhor a este método (AHMAD et al., 2016).

Para Tolfo et al. (2011) é necessário avaliar quais são as características do projeto, da equipe e do processo para escolher qual método ágil utilizar. Além disso, é possível adotar práticas de diferentes métodos e aos poucos ir formando um processo ágil de acordo com as necessidades de cada cenário.

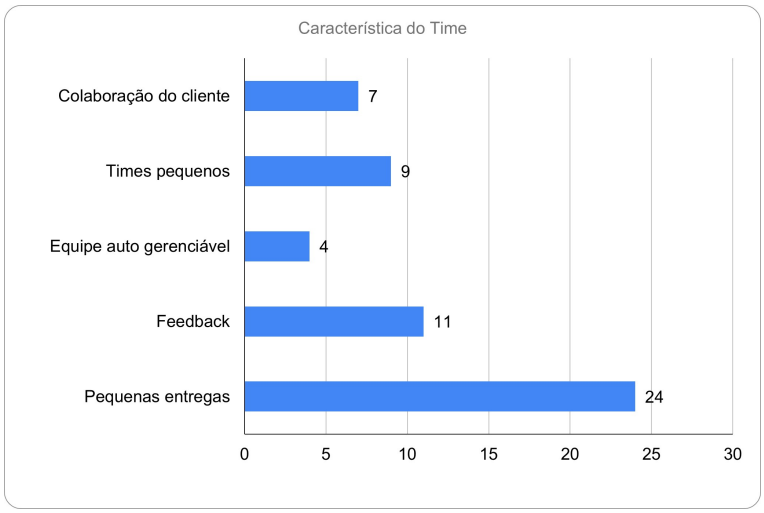
Durante a leitura dos artigos, pode-se perceber que diversas técnicas, práticas e metodologias foram utilizadas. Com o objetivo de ter uma melhor visualização dessas práticas, elas foram divididas em 6 categorias: (i) Gestão, (ii) Características do time ágil, (iii) Cerimônias, (iv) Organização de tarefas e *backlog* do produto, (v) Práticas de desenvolvimento e (vi) Testes.

Sobre a gestão de projetos de manutenção ágeis, dois itens foram selecionados para serem classificados como itens de gestão: (i) Realização de reuniões semanais com estratégias, estimativas e priorização e (ii) Monitoração contínua. As reuniões semanais com o cliente são importantes em projetos de manutenção evolutiva para manter as equipes alinhadas com as estratégias, estimativas e priorização das tarefas (RICO, 2008). A monitoração contínua, citada por Tarwani e Chug (2016), foca em ter radiadores de informação como quadro Kanban e gráficos do andamento da entrega da equipe.

No gráfico da Figura 6, o eixo x apresenta a quantidade de citações das práticas relacionadas à características gerais do time que estão listadas no eixo y. A realização

de pequenas entregas é a prática mais citada nos artigos que foram analisados. Os times ágeis possuem como objetivo entregar valor para o cliente com certa frequência (duas a quatro semanas). Além das pequenas entregas, *feedbacks* constantes, times pequenos e colaboração do cliente também são características muito citadas.

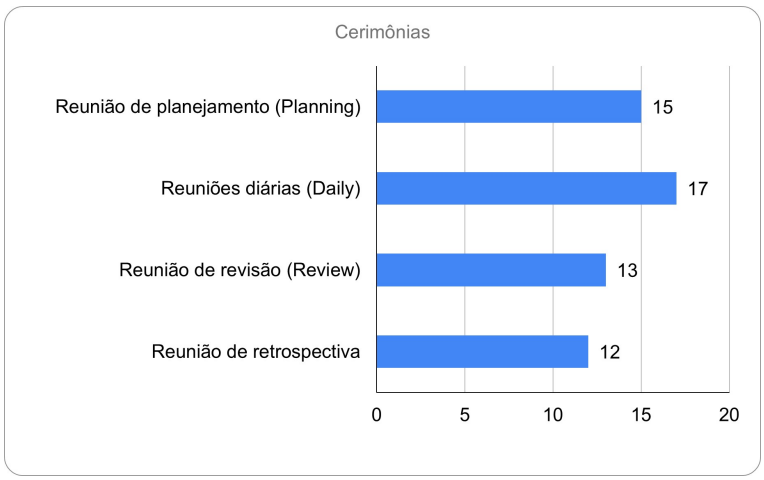
Figura 6 - Características do time




Legenda: Quantidade de artigos que cada característica foi encontrada.
Fonte: A autora, 2020.

No gráfico da Figura 7, é apresentado no eixo x a quantidade de vezes que cada cerimônia ágil que está no eixo y foi mencionada em manutenção evolutiva de *software* nos artigos analisados. As reuniões diárias, reunião de planejamento, reunião de revisão e retrospectiva foram as cerimônias ágeis mais citadas citadas.

Figura 7 - Cerimônias

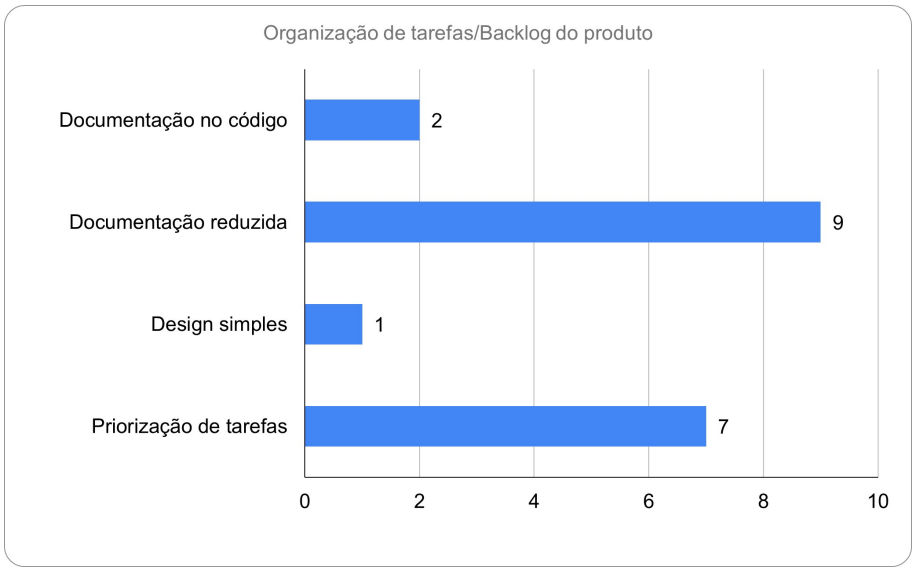



Legenda: Quantidade de artigos que cada cerimônia ágil foi encontrada.
Fonte: A autora, 2020.

Conforme visto no Capítulo 1, uma característica que se pode destacar nos modelos

de desenvolvimento de *software* tradicional é a documentação bastante detalhada e documentos extensos. Na metodologia ágil, prefere-se ter uma documentação mais reduzida e também existem projetos que realiza a documentação a partir de comentários no código, o que facilita bastante a fase de manutenção evolutiva. O gráfico da Figura 8 apresenta no eixo y todas as práticas que foram citadas nos artigos que possuem relação com as organizações de tarefas e com a descrição do *backlog* do produto e o eixo x a quantidade de citações encontradas nos artigos. A documentação reduzida foi o item mais utilizado pelos trabalhos de manutenção evolutiva de *software*, outras práticas como: documentação no código, *design* simples e priorização de tarefas também são destacadas pelo uso na manutenção evolutiva de *software*.

Figura 8 - Organização de tarefas/backlog do produto

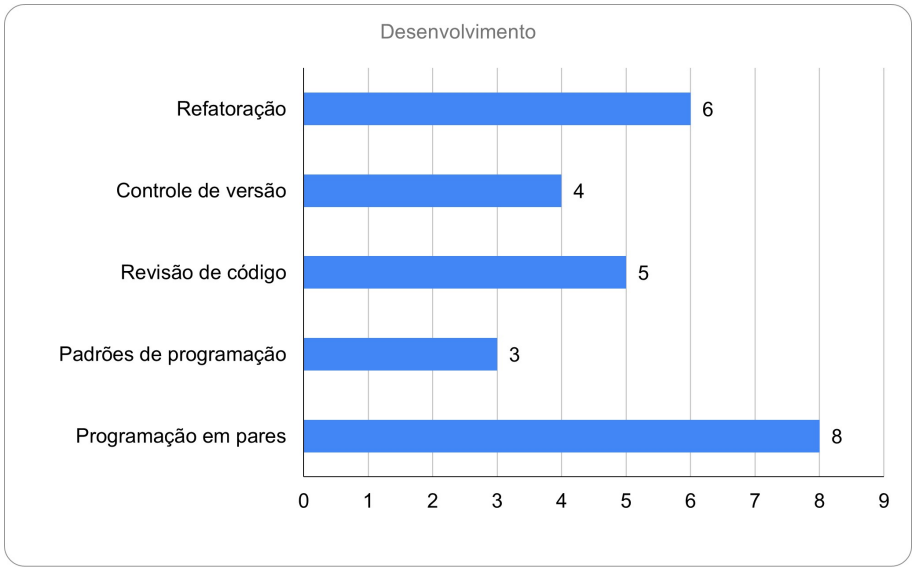



Legenda: Quantidade de artigos que cada característica de organização de tarefas ou sobre backlog do produto foi encontrada.
Fonte: A autora, 2020.

Além das características de organização das tarefas e do *backlog* do produto, foram encontradas as práticas de desenvolvimento representadas no gráfico da Figura 9. A programação em pares é a prática mais citada pelos artigos do mapeamento sistemático. Além dela, a refatoração do código, o controle de versão, a revisão de código e os padrões de programação também foram citados nos artigos.

Em projetos ágeis, os testes são realizados constantemente e não apenas na fase final do projeto. Por isso, as boas práticas de testes, como testes unitários e testes automatizados são muito citados, conforme exibido no gráfico da Figura 10. Os testes automatizados e testes de regressão são muito utilizados, principalmente por serem muito valiosos em fases de manutenção evolutiva de *software*. Isso ocorre pois a cada nova atualização que o time de desenvolvimento realiza, os testes de regressão (automatizados

Figura 9 - Desenvolvimento




Legenda: Quantidade de artigos que cada prática de desenvolvimento foi encontrada.
Fonte: A autora, 2020.

ou não) verificam se alguma outra funcionalidade deixou de funcionar devido a subida de código da nova funcionalidade. Além desses tipos de testes, são citados os testes funcionais, teste de aceitação, teste de integração e o desenvolvimento orientado a testes (TDD).

### 2.7.2 Vantagens da Abordagem Ágil

O uso de práticas ágeis oferecem uma solução ao desenvolvimento de *software* iterativo e incremental, onde requisitos e soluções evoluem através da colaboração entre o time de desenvolvimento e o cliente, permitindo uma flexibilidade na mudança de requisitos (MANUJA et al., 2014). A Tabela 4 apresenta uma síntese das vantagens do uso da abordagem ágil e em quais trabalhos estas foram identificadas.

Uma das maiores vantagens de utilizar o ágil na manutenção evolutiva é a habilidade de adaptação à mudanças, onde o escopo do projeto é aberto e o cliente pode solicitar alterações nos requisitos. Além disso, com as entregas frequentes, existe o *feedback* constante do cliente (vantagem 2) e a melhora na comunicação (vantagem 3).

Além das vantagens já identificadas, com a redução no número de erros sendo uma consequência natural do constante *feedback* do cliente e da melhora na comunicação (vantagem 3), existirá uma natural redução de custos na fase de manutenção (vantagem 4).

As práticas ágeis também permitem uma maior visibilidade e transparência das

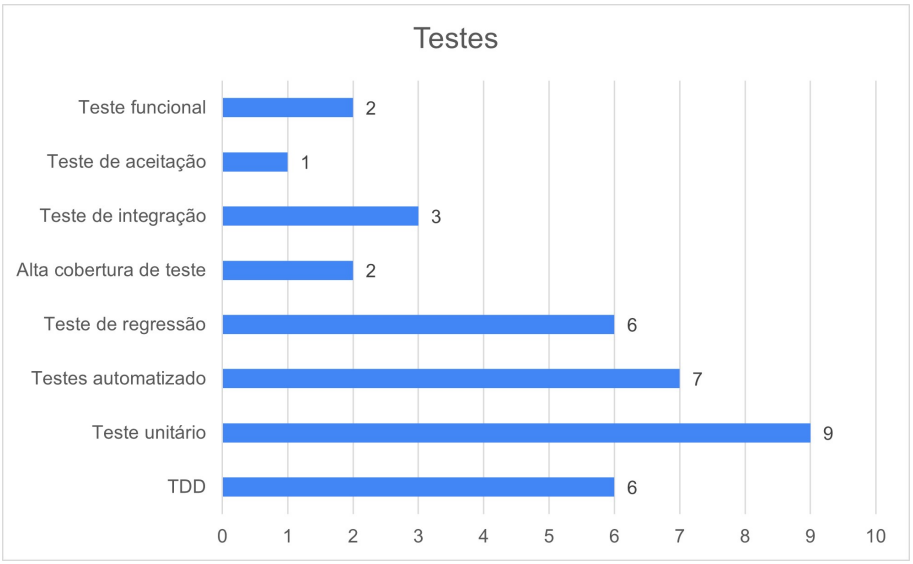
Tabela 4 - Quais as vantagens do uso da abordagem ágil em manutenção?

| # | **Descrição** | **Trabalhos** |
|---|---|---|
| 1 | Flexibilidade às mudanças | (RAO; NAIDU; CHAKKA, 2011), (MANUJA et al., 2014), (ABRAHAMSSON; OZA; SIPONEN, 2010), (KNIPPERS, 2011), (REN et al., 2011), (SANTOS, 2015), (SANTOS; OLIVEIRA, 2012), (STOICA; MIRCEA; GHILIC-MICU, 2013), (MARTINS, 2013), (JAVANMARD; ALIAN, 2015), (KILPALA; KÄRKKÄINEN, 2015), (KUHRMANN et al., 2016), (TARWANI; CHUG, 2016), (NAZ; KHAN; AAMIR, 2016), (ZANOTTI; KAYLOR; DAVIDSEN, 2017), (BOGOJEVIĆ et al., 2017) |
| 2 | *Feedback* constante | (MANUJA et al., 2014), (DUKA, 2012), (PINO et al., 2012), (TRIMBLE; WEBSTER, 2013), (RAMOS et al., 2013), (CHOUDHARI; SUMAN, 2014), (JAVANMARD; ALIAN, 2015), (KUHRMANN et al., 2016), (ABRAHAMSSON et al., 2017), (RAO; NAIDU; CHAKKA, 2011), (STOICA; MIRCEA; GHILIC-MICU, 2013), (COSTA, 2016) |
| 3 | Melhora na comunicação | (KNIPPERS, 2011), (SANTOS; OLIVEIRA, 2012), (CHOUDHARI; SUMAN, 2012), (TRIMBLE; WEBSTER, 2013), (STOICA; MIRCEA; GHILIC-MICU, 2013), (RAMOS et al., 2013), (JAVANMARD; ALIAN, 2015), (HEEAGER; ROSE, 2015), (SANTOS, 2015), (KILPALA; KÄRKKÄINEN, 2015), (KUHRMANN et al., 2016), (ZANOTTI; KAYLOR; DAVIDSEN, 2017), (NAZ; KHAN; AAMIR, 2016), (ZANOTTI; KAYLOR; DAVIDSEN, 2017), (LOUS et al., 2018), (COSTA, 2016) |
| 4 | Redução de custos | (SANTOS; OLIVEIRA, 2012), (DUKA, 2012), (STOICA; MIRCEA; GHILIC-MICU, 2013), (ABRAHAMSSON et al., 2017) |
| 5 | Visibilidade das atividades/Transparência | (THANGASAMY, 2012), (SILVA; SANTOS; NETO, 2012), (MARTINS, 2013) |
| 6 | Trabalho estimulante | (SANTOS; OLIVEIRA, 2012), (BOGOJEVIĆ et al., 2017), (CHOUDHARI; SUMAN, 2012) |
| 7 | Produtividade do time | (SANTOS; OLIVEIRA, 2012) |
| 8 | Satisfação do cliente | (CHOUDHARI; SUMAN, 2012), (TRIMBLE; WEBSTER, 2013) |
| 9 | Aumento da qualidade | (SANTOS; OLIVEIRA, 2012) |
| 10 | Equipe madura | (SILVA; SANTOS; NETO, 2012) |
| 11 | Antecipação de risco | (STOICA; MIRCEA; GHILIC-MICU, 2013) |
| 12 | Foco nas entregas mais importantes | (TRIMBLE; WEBSTER, 2013) |
| 13 | Métricas para gerência | (SILVA; SANTOS; NETO, 2012), (MARTINS, 2013) |

Legenda: As vantagens encontradas durante a leitura dos artigos encontrados no mapeamento sistemático.

Fonte: A autora, 2021.

Figura 10 - Testes




Fonte: A autora, 2021.

atividades (vantagem 5) que estão sendo realizadas pela equipe. Como consequência, é possível antecipar os riscos (vantagem 11) através dessa transparência.

Santos e Oliveira (2012), Bogojević et al. (2017) e Choudhari e Suman (2012) citaram que o uso da agilidade torna o trabalho mais estimulante (vantagem 6), consequentemente aumenta a produtividade do time (vantagem 7) e a equipe se torna madura (vantagem 10). Além do time, a satisfação do cliente (vantagem 8) também é uma vantagem citada pelos autores.

Com o uso das práticas ágeis durante o ciclo de desenvolvimento, a qualidade do *software* aumenta (vantagem 9) pois o foco do time é nas entregas mais importantes (vantagem 12) e também permite a equipe coletar métricas para gerência (vantagem 13).

### 2.7.3 Desafios da Abordagem Ágil

Kuhrmann et al. (2016) afirmam que equipes e projetos grandes são difíceis de gerenciar (desafio 1). Portanto, se o projeto for de grande escala, é necessário organizar as atividades e entregas por times menores. Rao *et al.* (2011) ressaltam que aplicar agilidade em organizações com projetos de grande escala é um desafio pois geralmente essas organizações não possuem uma visão geral e sim, múltiplas interdependências que não podem ser efetivamente monitoradas. Esse mesmo desafio é apontado em outros trabalhos (THANGASAMY, 2012), (STOICA; MIRCEA; GHILIC-MICU, 2013), (KAUR; JAJOO et al., 2015).

Um outro desafio em relação às equipes ágeis é a diversidade cultural (desafio 2).

A diversidade cultural envolve a comunicação entre equipes que não trabalham no mesmo ambiente, possuem diferenças comportamentais ou não falam o mesmo idioma (GHOSH, 2012). Duka (2012) afirma que muitas equipes de desenvolvimento de *software* que tentam aplicar abordagem ágil se encontram em ambientes tradicionais e, como resultado, são forçadas a seguir procedimentos, como documentação adicional ou revisões técnicas, que aumentam os custos gerais do projeto.

Rao *et al.* (2011) afirmam que podem ocorrer expectativas irreais (desafio 3) em relação ao ritmo de desenvolvimento do time, o que geralmente ocorre quando o time é novo e ainda não está integrado. Além disso, Tarwani e Chug (2016) e Manuja et al. (2014) afirmam que a mudança de *mindset* do time (desafio 7) também é um grande desafio em utilizar práticas ágeis na fase de manutenção.

Além dessas expectativas não realistas, as equipes de manutenção podem lidar com tarefas de características bem diferentes, imprevisíveis e críticas que exigem respostas rápidas (desafio 4). Essas tarefas são difíceis de realizar dentro de uma iteração ou *sprint*, sendo um desafio executar esse tipo de tarefas em abordagens ágeis que utilizam o conceito de iterações com tempo definido (AHMAD et al., 2016) (SILVA; SANTOS; NETO, 2012).

Também existe o cenário onde existe a dificuldade em estimar o esforço necessário (desafio 5) para algumas tarefas pois haviam outras dependências externas, consequentemente o tamanho da *sprint* pode acabar se tornando variável e proporcional à necessidade da entrega. Kaur *et al.* (2015) apresentam o caso de um projeto de manutenção de *software* de um banco estatal dos EUA que atende cerca de 1 milhão de clientes online e que deseja que os seus serviços de *internet banking* sejam mantidos por uma empresa X. Kaur *et al.* (2015) afirma que foi um desafio para a empresa X mapear o cenário para o ciclo de vida ágil para um projeto de manutenção.

Para que a abordagem ágil funcione com sucesso, é necessário que níveis gerenciais e o cliente do projeto estejam alinhados com a abordagem ágil (GHOSH, 2012). Caso contrário, não haverá sincronismo do trabalho e indicará que a coordenação entre as equipes de gerenciamento, desenvolvimento e manutenção não é frequente nem eficaz (desafio 6) (AHMAD et al., 2016), conforme exibido na Tabela 5.

Manter documentação atualizada (desafio 8) pode ser um grande desafio em manutenção de *software* pois tem algumas tarefas que são incidentes e precisam de alterações rápidas. Devido a restrições de tempo e orçamento, as mudanças são realizadas sem planejamento, design, análise de impacto ou teste de regressão adequados, o que resulta na degradação da qualidade da documentação (CHOUDHARI; SUMAN, 2014) (HEEAGER; ROSE, 2015).

Muitos autores citaram que identificar as melhores práticas ágeis (desafio 9) para cada cenário de projeto ou organização é um grande desafio. Existem cenários que podem ocorrer mudanças no meio de uma *sprint* e é necessário interrompê-la, atrapalhando o andamento das atividades (HEEAGER; ROSE, 2015) (BOGOJEVIĆ et al.,

2017) (ABRAHAMSSON; OZA; SIPONEN, 2010).

Tabela 5 - Quais os desafios do uso da abordagem ágil na fase de manutenção?

| # | **Descrição** | **Trabalhos** |
|---|---|---|
| 1 | Gerenciamento de equipes e projetos/produtos grandes | (RAO; NAIDU; CHAKKA, 2011), (STOICA; MIRCEA; GHILIC-MICU, 2013), (KAUR; JAJOO et al., 2015), (THANGASAMY, 2012), (KUHRMANN et al., 2016), (JAVANMARD; ALIAN, 2015), (COSTA, 2016), (SILVA, 2018), (ABRAHAMSSON et al., 2017), (TARWANI; CHUG, 2016) |
| 2 | Diversidade Cultural | (GHOSH, 2012), (DUKA, 2012), (LOUS et al., 2018), (MANUJA et al., 2014) |
| 3 | Expectativas não realistas | (RAO; NAIDU; CHAKKA, 2011) |
| 4 | Tarefas imprevisíveis frequentemente em andamento | (AHMAD et al., 2016), (SILVA; SANTOS; NETO, 2012), (KILPALA; KÄRKKÄINEN, 2015) |
| 5 | Dificuldade em estimar | (KAUR; JAJOO et al., 2015) |
| 6 | Práticas ágeis em diferentes níveis | (GHOSH, 2012), (AHMAD et al., 2016), (SILVA; SANTOS; NETO, 2012), (RAMOS et al., 2013), (MARTINS, 2013) |
| 7 | Mudança de *mindset* da equipe | (MANUJA et al., 2014), (TARWANI; CHUG, 2016) |
| 8 | Manter documentação atualizada | (CHOUDHARI; SUMAN, 2012), (HEEAGER; ROSE, 2015) |
| 9 | Identificar melhores práticas | (HEEAGER; ROSE, 2015), (BOGOJEVIĆ et al., 2017), (ABRAHAMSSON; OZA; SIPONEN, 2010) |

Legenda: Os desafios encontrados durante a leitura dos artigos encontrados no mapeamento sistemático.
Fonte: A autora, 2021.

# 3 SURVEY

O objetivo do *survey* foi analisar o uso de práticas ágeis em equipes de manutenção evolutiva de *software*, com o objetivo de coletar dados para auxiliar na elaboração do *framework* de manutenção ágil.

Para efetuar a seleção dos participantes, foi utilizado a lista de *e-mails* e contatos dos pesquisadores. A busca foi feita até 17/03/2021 e considerou participantes que trabalham na área de desenvolvimento de *software* e já atuou na fase de manutenção evolutiva de *software*.

## 3.1 Coleta de dados

O esforço de coleta de dados foi realizado através de um questionário com especialistas. O questionário utilizado na pesquisa foi elaborado pela autora a partir dos resultados do mapeamento sistemático da literatura apresentados na seção 2.7 do Capítulo 2. Por serem dados de caráter qualitativo, o projeto com a proposta do questionário foi enviado para avaliação do comitê de ética e aprovado com o número de Certificado de Apresentação de Apreciação Ética (CAAE) *40860520.9.0000.5282*.

O questionário apresenta uma estrutura com cinco partes: (i) termo de consentimento livre e esclarecido; (ii) identificação do perfil do respondente (empresa/instituição); (iii) perguntas específicas sobre documentação; (iv) perguntas específicas da codificação; e (v) perguntas específicas dos testes. A exibição de algumas perguntas depende de respostas anteriores, conforme exibido no Apêndice .

Os questionários foram difundidos nas listas de *e-mails* e contatos da universidade e grupos de trabalho dos pesquisadores envolvidos e ficaram disponíveis até 17/03/2021. As respostas individuais possibilitaram alcançar uma variedade de impressões e percepções de diferentes participantes.

Após a coleta de dados, utilizou-se a técnica de análise de conteúdo, a fim de analisá-los. A aplicação desta técnica está descrita na próxima seção.

## 3.2 Análise dos resultados

Foram obtidas 50 respostas válidas fornecidas pelos especialistas. Ao todo foram descartados uma resposta pois o participante não concordou com o termo de consentimento. Dos respondentes, 82,35% eram de empresas e 17,65% de instituições de ensino.

As respostas que foram utilizadas na análise dos dados são respostas de perguntas

abertas, ou seja, cada participante preencheu conforme sua experiência. Com o objetivo de agrupar os conceitos em um grau de abstração mais alto, foi utilizada a técnica de *coding* citada no Capítulo 1 (VOSGERAU; POCRIFKA; SIMONIAN, 2016).

### 3.2.1 Leitura flutuante

A análise de dados foi iniciada por meio de uma leitura flutuante, ou seja, todos os trechos que detinham relação com o que era buscado como resposta ou explicação para o questionamento foram selecionados. A Figura 11 exibe como esses trechos foram destacados no *software* escolhido para esta análise.

Figura 11 - Exemplo de trecho selecionado a partir da leitura flutuante


Fonte: A autora, 2021.

### 3.2.2 Categorias intermediárias

As primeiras categorias foram criadas em conformidade com os dados que as constituíram. Após a análise dessas categorias iniciais, foram criadas dez categorias intermediárias. A primeira categoria intermediária foi criada para unificar todas as práticas ágeis citadas pelos participantes e selecionadas na pergunta 6 do questionário, conforme exibido na Figura 12 e a categoria foi denominada de **Práticas Ágeis**.

A tabela 6 ilustra o processo de formação da segunda categoria intermediária, **Tipos/características da documentação**.

A terceira categoria intermediária, **Práticas da documentação**, apresentada na tabela 7, exibe todos os códigos que estão alinhados às práticas relacionadas com documentação de requisitos, como por exemplo, refinamento das tarefas com o time, versionamento de documentos, revisão da documentação e realização de reuniões de planejamento. Todas essas práticas serão mais detalhadas na Seção 3.2.4.

Tabela 6 - Categoria Intermediária - II. Tipos/Características da documentação

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Documentação extensa/bem detalhada | Evidencia o detalhamento da documentação |
| Documentação atualizada/"viva" | Indica a constante atualização da documentação |
| *Behavior Driven Development* (BDD)<br>Caso de Uso | Indica uma maneira de documentar<br>Indica um tipo de documentação |
| Especificação de requisitos / Especificação Funcional (EF) / regras de negócio | Indica um tipo de documentação |
| Documentação desatualizada / incompleta | Ponto de atenção com relação a documentação |
| Atas de reunião | Indica um tipo de documentação |
| Manual do usuário | Indica um tipo de documentação |
| Documentação de banco de dados / Modelos ER | Indica um tipo de documentação |
| História de usuário | Indica um tipo de documentação |
| Documentação simplificada | Evidencia uma característica da documentação |
| Composta de *wireframes* | Indica um tipo de documentação |
| Inexistente ou quase inexistente | Ponto de atenção com relação a documentação |
| Design de experiência com usuário<br>Testes como documentação do projeto | Indica um tipo de documentação<br>Tipo de documentação |
| Documentação na *wiki* | Localização da documentação no projeto |

Fonte: A autora, 2021.

Para facilitar a execução das práticas dentro de um projeto, utiliza-se ferramentas e com o avanço da tecnologia, está cada vez mais comum projetos que possuem ferramentas para auxiliar na gestão, principalmente no escopo, cronograma, custo e comunicação do time (KENCHICOSKI; CRUZ, 2020). A categoria IV, apresentada na tabela 8 foi criada para representar todas as **Ferramentas** citadas nas respostas dos participantes do *survey*.

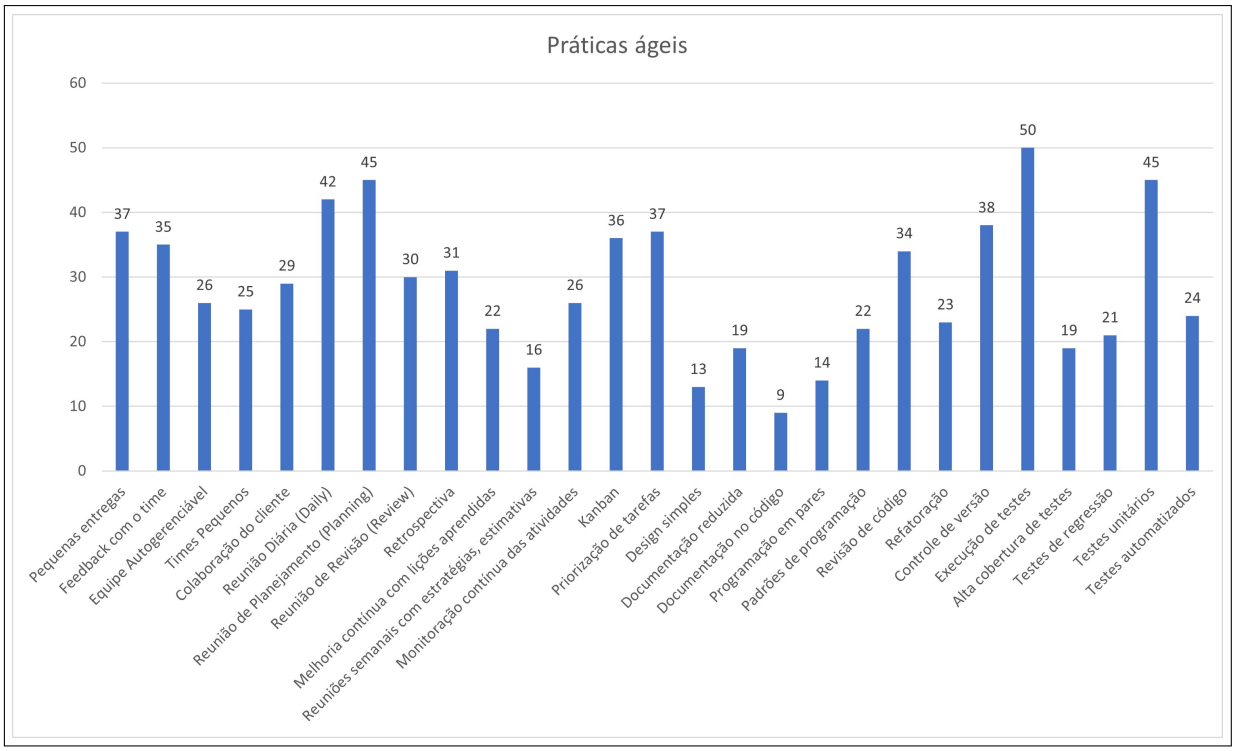
A quinta categoria, **Processo de codificação na manutenção**, analisa as percepções dos entrevistados acerca do processo de codificação na fase de manutenção de *software*. Na tabela 9, melhorias sendo tratadas como prioridade faz parte de definição do processo de manutenção, envolvendo o que o desenvolvedor vai priorizar no momento da entrega. Além disso, avaliar a tecnologia para o que foi solicitado, verificando se a tecnologia existente no projeto atende ou se vai ter que ser realizado alguma mudança arquitetural e outro ponto do processo que foi criticado é o acúmulo de tarefas.

Tabela 7 - Categoria Intermediária - III. Práticas de documentação

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Reuniões de detalhamento com cliente | Evidencia prática ágil relacionada a documentação |
| Refinamento com cliente | Evidencia prática ágil relacionada a comunicação e detalhamento das tarefas com o cliente |
| Refinamento com equipe | Evidencia prática ágil relacionada a comunicação e detalhamento das tarefas com o time |
| Documentos versionados | Boa prática de documentação |
| Documentação construída por *sprint* | Boa prática de documentação |
| Priorização de *backlog* | Indica a priorização de tarefas de *backlog* |
| Revisão da documentação | Boa prática de documentação |
| Aprovação do cliente | Evidencia a participação do cliente |
| Refinamento realizado de acordo com métricas do cliente | Evidencia a participação do cliente |
| Novas hipóteses eram testadas ainda na fase de *discovery* | Prática utilizada para testar hipóteses |
| Avaliação das histórias pós *planning* | Indica preocupação do time com as tarefas que serão realizadas |
| Priorização focada no *feedback* do usuário final/cliente | Evidencia a participação do cliente |
| conceitos de agilidade se perderam | Ponto de atenção |
| Novas necessidades por parte do cliente | Evidencia a participação do cliente |
| Refinamento de *backlog*/Novo *backlog* | Prática ágil com relação as histórias |
| Estimativa de esforço/*planning poker* | Boa prática para calcular performance do time durante as entregas |
| *Planning* | Indica a utilização de cerimônias ágeis |
| Documentação tratada por demanda | Boa prática de documentação |
| Tarefas incidentes (Kanban) x evolutivas (Scrum) | Apresenta dois tipos de metodologias utilizadas para separar tipo de tarefas |
| Reunião semanal para criação de *backlog* da semana | Evidencia processo para criação de *backlog* |
| Design simples/documentação reduzida | Evidencia a prática de documentação simples |
| Documentação no código | Boa prática de documentar através do código |

Fonte: A autora, 2021.

Figura 12 - Práticas ágeis




Legenda: Quantidade de vezes citadas nas respostas do *survey*.
Fonte: A autora, 2021.

Muitas boas práticas relacionadas ao desenvolvimento do código foram relatas pelos participantes e estão representadas na tabela 10 como categoria **Práticas de codificação**.

Diversos tipos de testes foram mencionados nas respostas e a sétima categoria intermediária criada foi **Tipo de testes** para identificar os tipos de testes e periodicidade destes testes. Todas as categorias iniciais relacionadas a teste está representado na tabela 11 e outras informações sobre cobertura de testes estão na categoria de **Cobertura de testes** e estão exibidas na tabela 12.

Muitos desafios e problemas são encontrados durante desenvolvimento de *software* e alguns desses problemas foram vistos como lições aprendidas para melhorar na fase de manutenção, essas informações são encontradas na tabela 13 e foram adicionadas na categoria **Pontos de atenção**.

Além desses problemas, algumas respostas apresentaram conteúdo relacionado com acompanhamento das atividades, portanto, foi criada a categoria **Acompanhamento**, representada na tabela 14 com as categorias iniciais que são relacionadas com gerenciamento e acompanhamento das atividades.

Tabela 8 - Categoria Intermediária - IV. Ferramentas

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Azure Devops | Ferramenta utilizada para acompanhamento de projetos e escrita de requisitos |
| Confluence | Ferramenta utilizada para acompanhamento de projetos e escrita de requisitos |
| Figma | Ferramenta utilizada para elaboração de protótipos |
| Zeplin | Ferramenta utilizada para elaboração de protótipos |
| Jira | Ferramenta utilizada para acompanhamento de projetos e escrita de requisitos |
| *Aplication Lifecycle Management* (ALM) | Ferramenta utilizada para acompanhamento de projetos e escrita de requisitos |
| Excel | Ferramenta utilizada para escrita de documentação |
| Trello | Ferramenta utilizada para acompanhamento de tarefas por quadro Kanban |

Fonte: A autora, 2021.

Tabela 9 - Categoria Intermediária - V. Processo de codificação na manutenção

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Melhorias eram prioridades | Indica a priorização de melhorias |
| Avaliação de tecnologias para o que foi solicitado | Indica a avaliação de tecnologias |
| Tarefas não concluídas durante a *sprint* e acumulavam no *backlog* | Evidencia ponto de atenção para tarefas acumuladas |

Fonte: A autora, 2021.

Tabela 10 - Categoria Intermediária - VI. Práticas de codificação

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Revisão por par | Evidencia Prática de *par programming* |
| Padronização de código/Código otimizado | Indica boa prática de código |
| Código sem revisão | Sinaliza ponto de atenção com relação ao código |
| Não ter controle de versão | Sinaliza ponto de atenção com relação ao código |
| Revisão de código | Indica boa prática de código |
| Testes unitários | Sinaliza a realização de testes de unidade |
| Reúso de código | Indica que existe o reuso do código |
| Pequenas tarefas | Evidencia que o desenvolvimento é realizado em cima de pequenas tarefas |
| Falta de padrão de código | Sinaliza ponto de atenção com relação ao código |
| Pontuação das histórias na reunião de planejamento | Boa prática para calcular performance do time durante as entregas |
| Controle de versão | Indica boa prática de código |
| Documentação no código | Boa prática de desenvolver com comentários e com clareza para o código servir de documentação |

Fonte: A autora, 2021.

Tabela 11 - Categoria Intermediária - VII. Tipo de testes

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Execução de testes manuais/funcionais | Indica tipo de teste realizado |
| Testes unitários | Indica tipo de teste realizado |
| Documentação de testes (casos de testes ou plano de testes) | Tipos de documentação de teste |
| Testes automatizados | Indica tipo de teste realizado |
| Teste baseado em BDD | Indica tipo de teste realizado |
| Testes exploratórios | Indica tipo de teste realizado |
| Testes de regressão | Indica tipo de teste realizado |
| Testes de integração | Indica tipo de teste realizado |
| Testes de retrocompatibilidade | Indica tipo de teste realizado |
| Testes de performance | Indica tipo de teste realizado |
| Testes de interface de programação de aplicações (API) | Indica tipo de teste realizado |

Fonte: A autora, 2021.

Tabela 12 - Categoria Intermediária - VIII. Cobertura de testes

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Maior cobertura de testes | Salienta a cobertura de testes |
| Testes ocorriam por *sprint* | Evidencia a periodicidade que o teste é realizado |
| Testes constantes | Evidencia a periodicidade que o teste é realizado |

Fonte: A autora, 2021.

Tabela 13 - Categoria Intermediária - IX. Pontos de atenção

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Código sem revisão | Indica falta de uma boa prática de codificação |
| Não ter controle de versão | Indica falta de uma boa prática de organização de código |
| Falta de padrão de código | Indica falta de uma boa prática de codificação |
| Documentação desatualizada/incompleta | Evidencia problema na documentação |
| Documentação inexistente ou quase inexistente | Evidencia problema na documentação |
| Conceitos da agilidade que se perderam | Destaca problema no *mindset* da equipe |
| Tarefas não concluídas durante a *sprint* e acumulavam no *backlog* | Evidencia ponto de atenção para tarefas acumuladas |
| Não tinha profissional dedicado aos testes | Indica falta de membro dedicado a testes |
| Entregas eram realizadas sem testes | Indica falta de preocupação com qualidade da entrega |

Fonte: A autora, 2021.

Tabela 14 - Categoria Intermediária - X. Acompanhamento

| **Categoria Inicial** | **Conceito norteador** |
|---|---|
| Reuniões semanais de requisitos | Evidencia periodicidade de reuniões para definição/refinamento de requisitos |
| Status do andamento | Elaboração de status do andamento das atividades |
| Métricas coletadas após a entrega | Indica a análise através de métricas coletadas após entregas das funcionalidades |
| Reuniões de melhoria continua | Evidencia o conceito de melhoria contínua através de reuniões com o time |

Fonte: A autora, 2021.

### 3.2.3 Categorias finais

As categorias iniciais e intermediárias, apresentadas nas seções anteriores, auxiliaram na construção das categoriais finais: (i) Documentação; (ii) Desenvolvimento; (iii) Testes; e (iv) Processo de manutenção. Elas foram construídas com o objetivo de auxiliar a interpretação e inferir os resultados para elaboração do *framework*. A Figura 13 apresenta as categorias intermediárias que constituem as categorias finais.

Com o objetivo de exibir a progressão das categorias, foram elaboradas as Figuras 14, 15, 16 e 17. Estas figuras foram utilizadas como base para construção da versão do *framework* ágil a partir de relatos de especialistas.

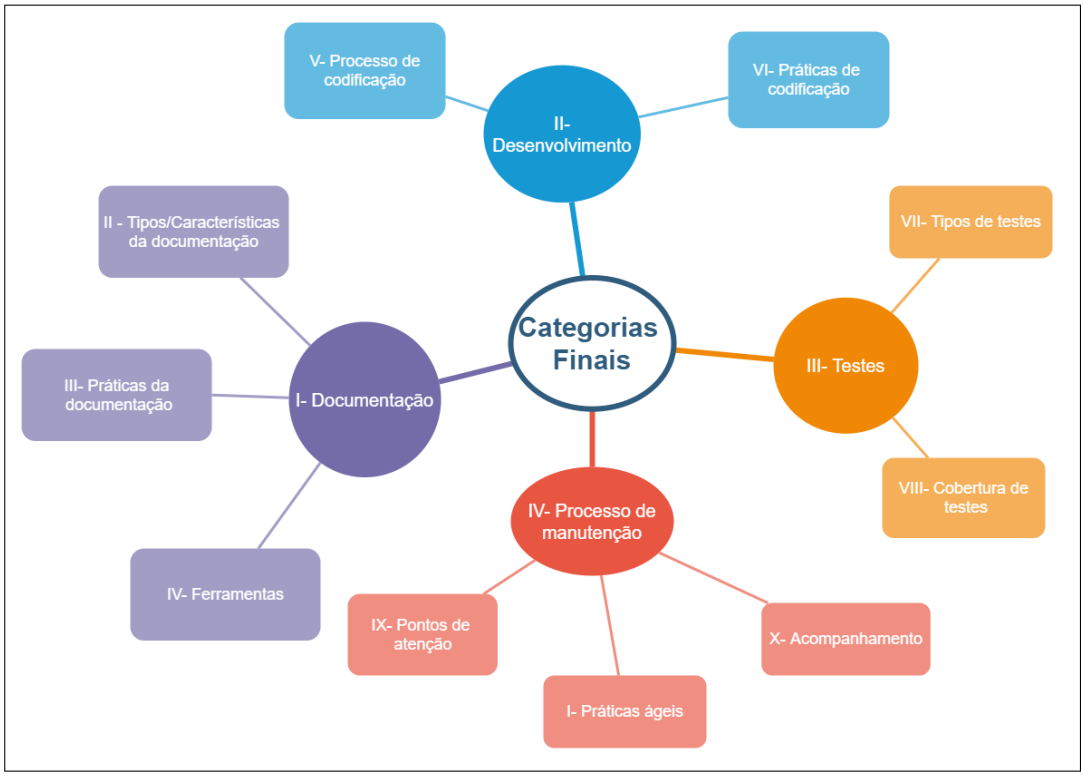
A figura 14 apresenta a categoria final **Documentação**, composta pelas categorias intermediárias **II- Tipos/Características da documentação, III- Práticas de documentação** e **IV- Ferramentas**. Todas as categorias intermediárias que tinham relação com documentação foram incluídas para compor a categoria final I.

A segunda categoria final **Codificação** representada na figura 15, engloba todas as categorias intermediárias relacionadas a práticas e processo de desenvolvimento de software, são elas **VI- Práticas de codificação** e **V- Processo de codificação na manutenção**.

A categoria final III exibida na figura 16 é relacionada a Testes. As categorias intermediárias que compõem a categoria III (Testes) são: **VII- Tipos de testes** e **VIII- Cobertura de testes**.

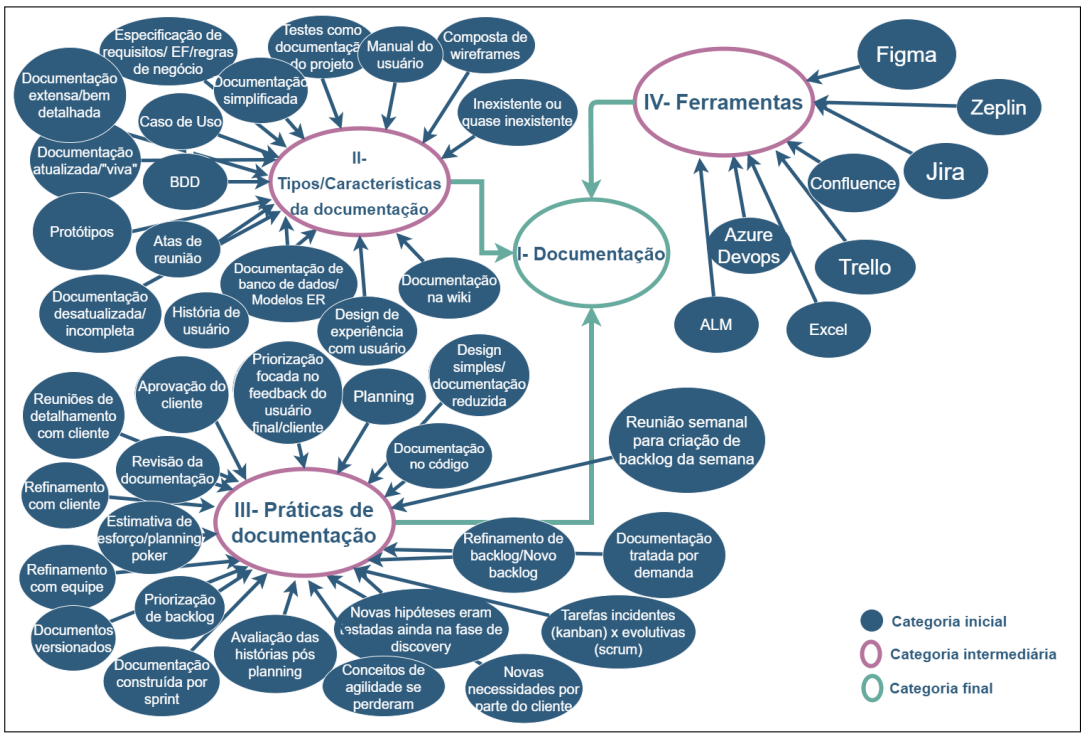
A última categoria final é a categoria IV- Processo de manutenção exibida na figura 17. Essa categoria reúne todas as intermediárias e iniciais que são relacionadas ao processo de manutenção. A categoria I- Práticas Ágeis é a primeira categoria intermediária que compõe a final de processo de manutenção, ela é composta por diversas práticas ágeis que são utilizadas na manutenção evolutiva de software. A segunda categoria intermediária é

Figura 13 - Categorias Finais



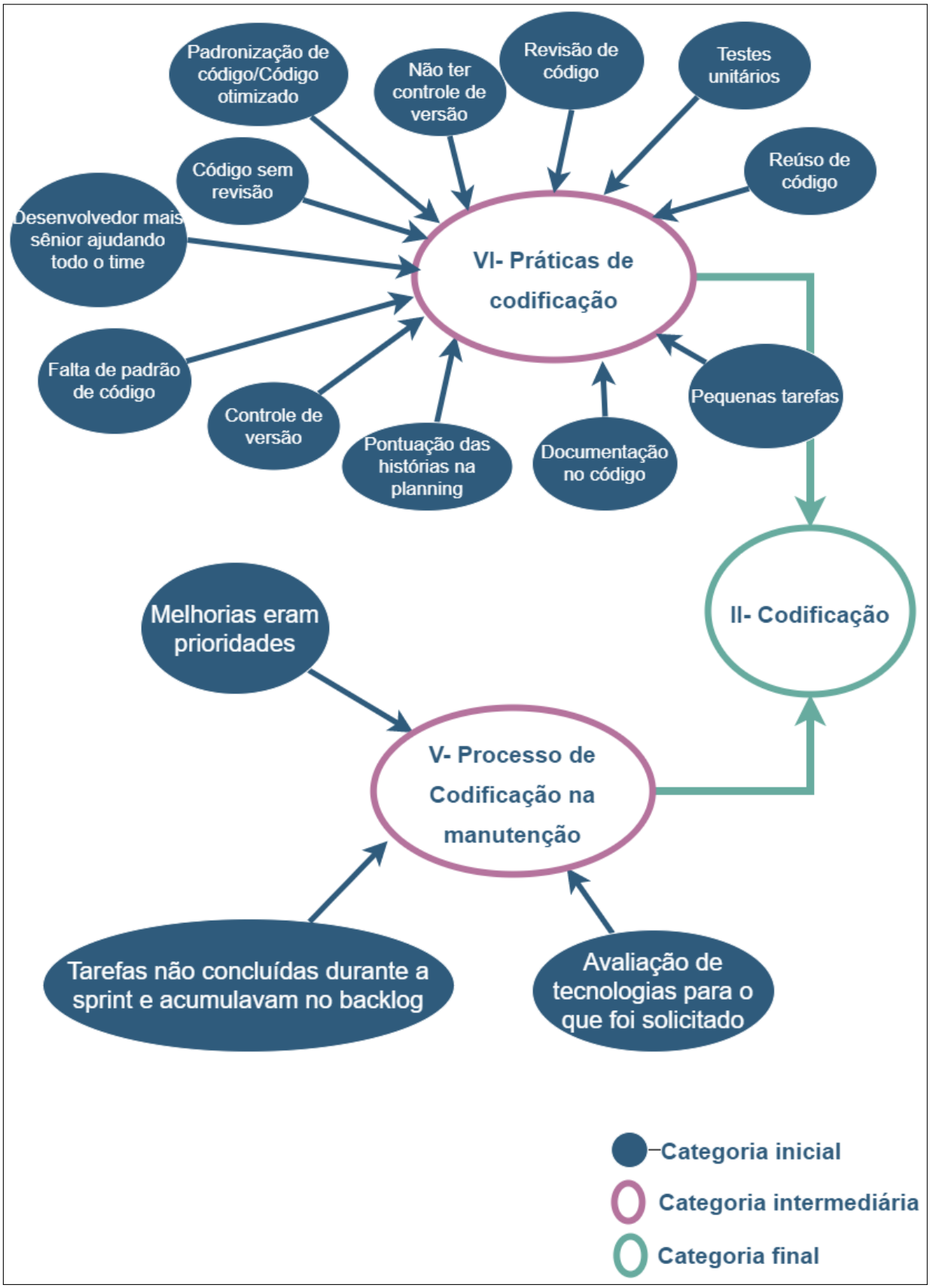

Fonte: A autora, 2021.

Figura 14 - Final I - Progressão das categorias



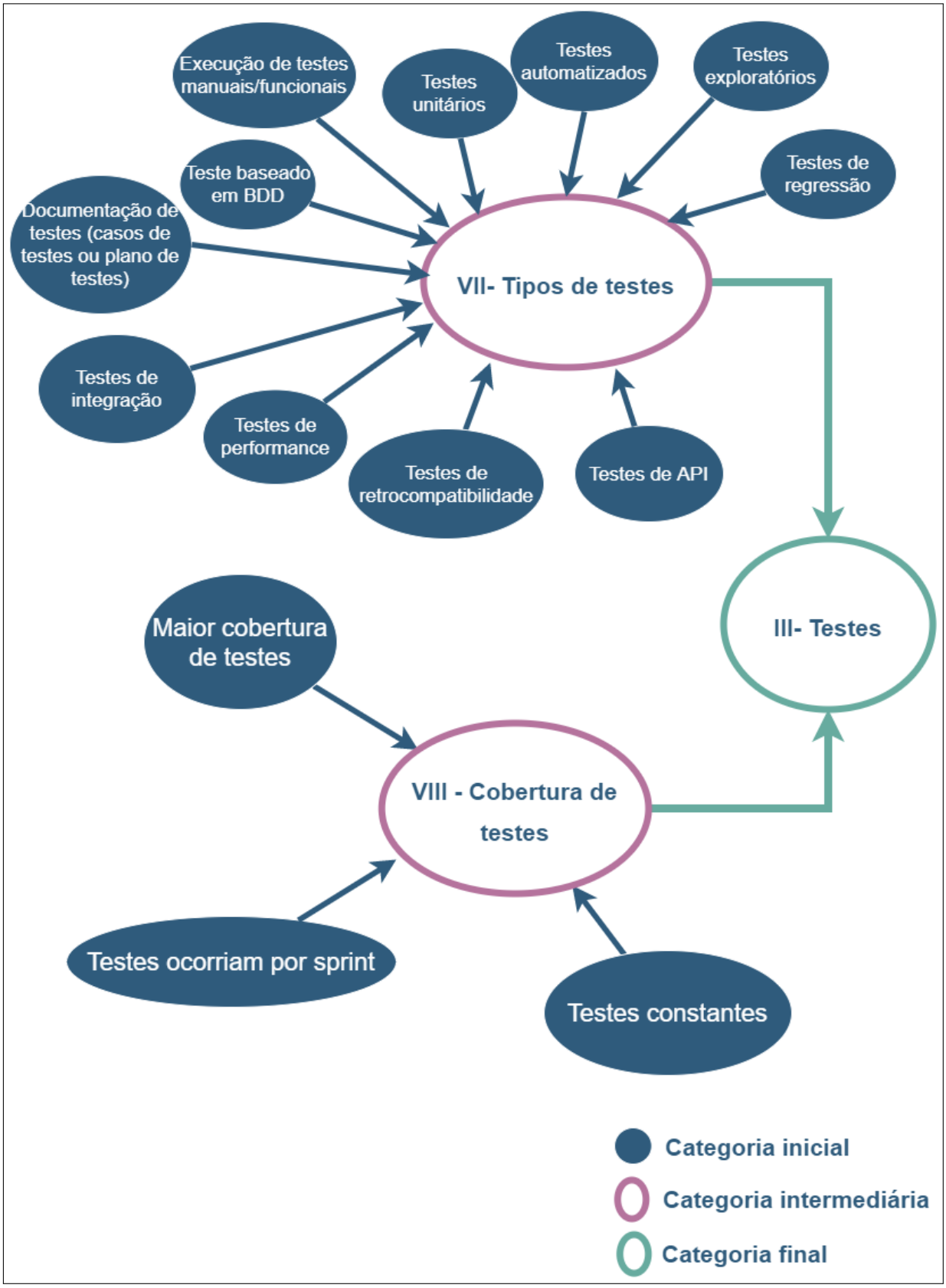

Fonte: A autora, 2021.

Figura 15 - Final II - Progressão das categorias




Fonte: A autora, 2021.

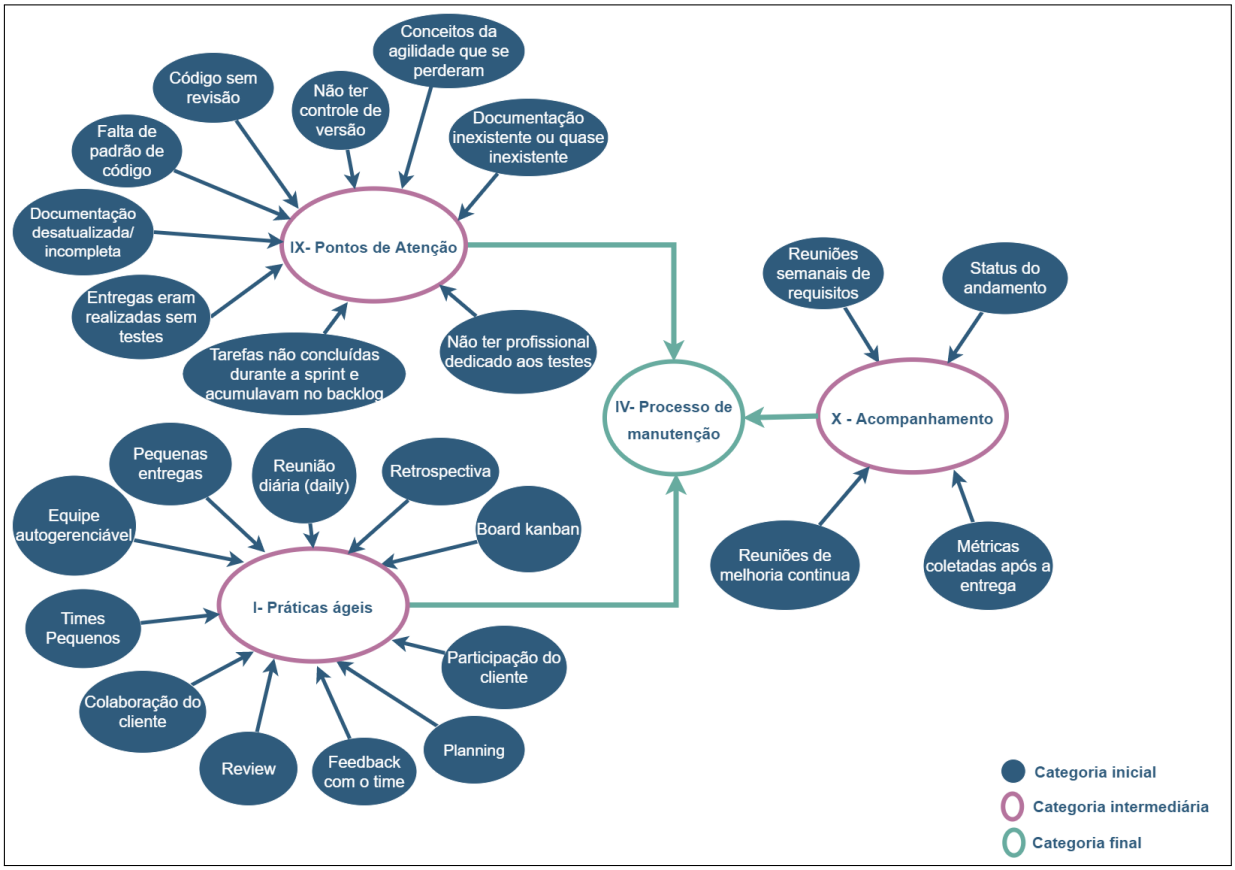
Figura 16 - Final III - Progressão das categorias




Fonte: A autora, 2021.

a categoria IX- Pontos de Atenção que reúne diversos pontos observados durante a leitura dos questionários, como por exemplo, códigos sem revisão, falta de padrão de código, etc. A terceira é a categoria X- Acompanhamento que tem categorias iniciais relacionadas ao acompanhamento do projeto com reuniões e métricas.

Figura 17 - Final IV - Progressão das categorias




Fonte: A autora, 2021.

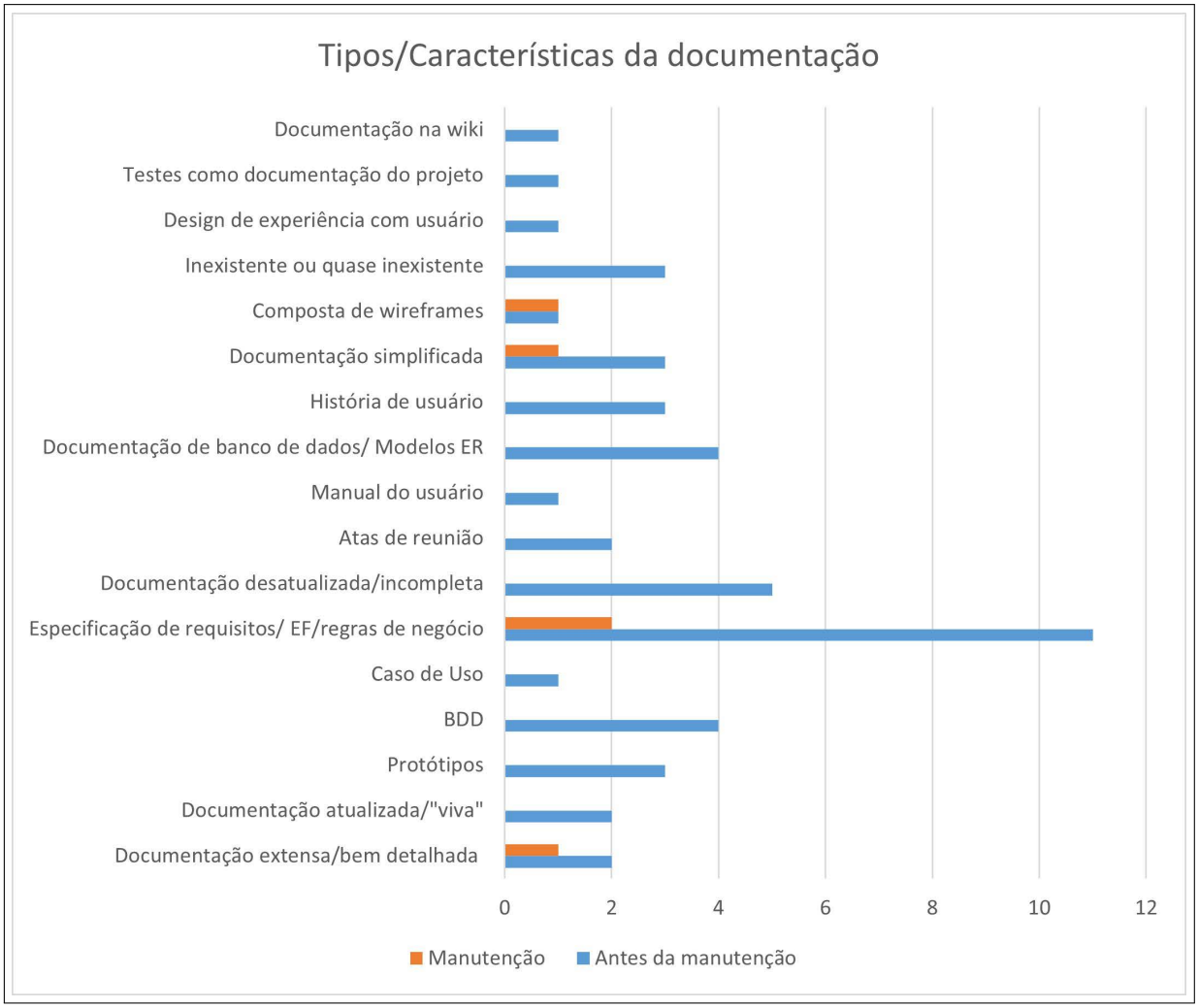
### 3.2.4 Análise detalhada dos dados

O questionário foi elaborado com o objetivo de coletar dados relacionados com fases de manutenção de *software* comparadas com fases anteriores a ela. Portanto, muitos participantes, nas perguntas abertas, relataram como era o desenvolvimento de *software* antes da fase de manutenção e durante a fase de manutenção. Com o objetivo de ter uma métrica quantitativa dos dados, foi contabilizada a quantidade de vezes que cada categoria inicial foi citada durante as perguntas relacionadas as fases anteriores à manutenção de *software* e durante a fase de manutenção de *software*.

- I- Documentação: A partir dos resultados exibidos na Figura 18 pode-se observar

que antes da fase de manutenção, alguns pontos negativos como inexistência da documentação ou documentação incompleta foram citados, mas essas características não foram encontradas na fase de manutenção ágil. Portanto, pode-se concluir que são itens que deixam de existir quando práticas ágeis são utilizadas.

Figura 18 - Intermediária I - Tipos/Características da documentação



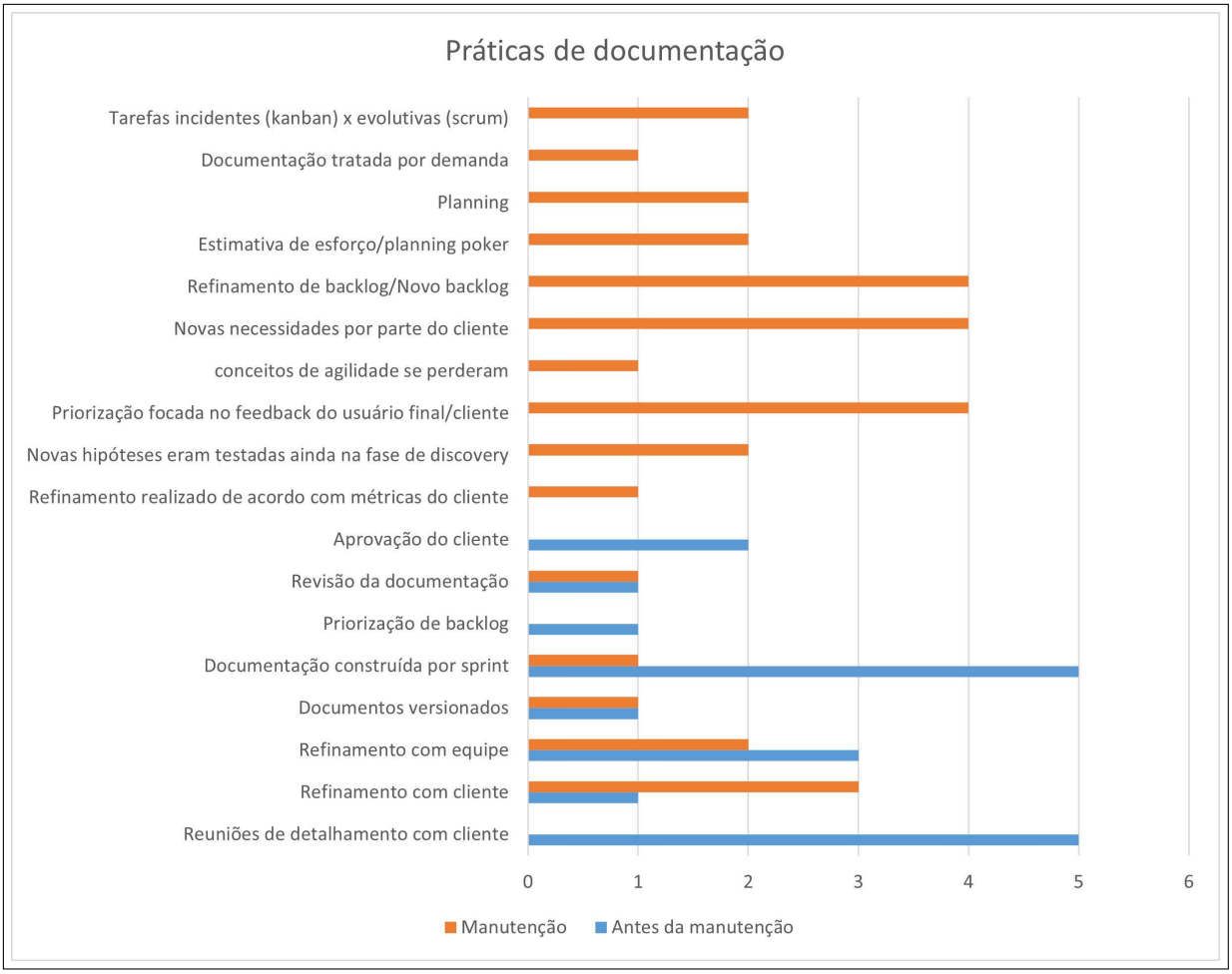

Fonte: A autora, 2021.

Analisando a Figura 19, muitas práticas de documentação foram citadas, destacando-se o **refinamento realizado de acordo com métricas do cliente**, **novas necessidades por parte do cliente** e **priorização focada no *feedback* do usuário final/cliente**. Estes itens são muito comuns na agilidade pois durante o desenvolvimento ágil, o cliente/*stakeholder* participa e acompanha mais de perto e com maior frequência o desenvolvimento do *software* e consequentemente, métricas e necessidades são identificadas.

Conforme visto na categoria intermediária III, algumas ferramentas de documentação foram citadas pelos participantes durante as respostas, são elas: Azure Devops,

Confluence, Figma, Zeplin, Jira, ALM, Excel, Git e Trello. Como esse dado foi citado por opção de alguns participantes, não foi contabilizado a quantidade de vezes pois essa informação não servirá de base de comparação já que foram respostas indiretas às perguntas realizadas.

Figura 19 - Intermediária II - Práticas de documentação



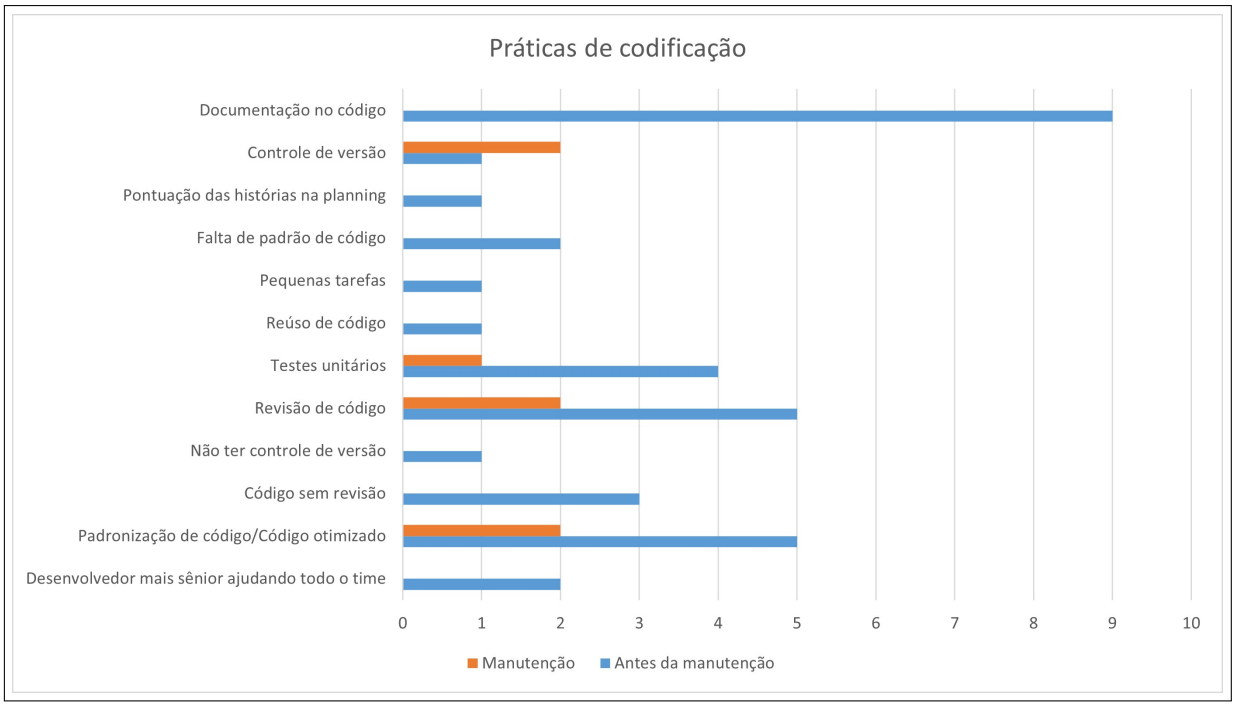

Fonte: A autora, 2021.

- II- **Desenvolvimento:** A categoria final Desenvolvimento engloba duas categorias intermediárias: (i) Processo de codificação e (ii) Práticas de codificação.

Durante as respostas, muitos participantes citaram alguns processos que ajudam no desenvolvimento, como: (i) melhorias como prioridades, o que é muito comum na fase de manutenção evolutiva de *software*, (ii) avaliação de tecnologias, e (iii) fase de manutenção para incorporar histórias que não foram entregues nas *sprints*.

**Código sem revisão, código sem controle de revisão e falta de padrão** são problemas levantados por alguns participantes na fase anterior à fase de manutenção, conforme exibido na Figura 20. Entretanto, outros participantes pontuaram **revisão de**

Figura 20 - Intermediária V - Práticas de codificação




Fonte: A autora, 2021.

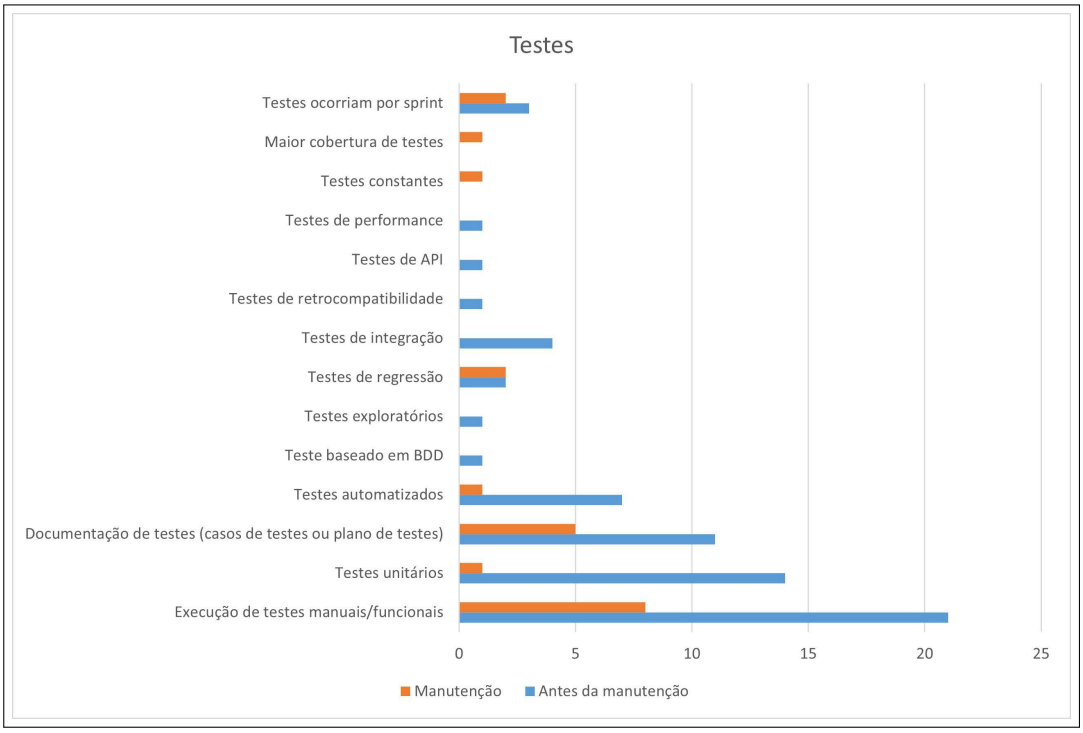
**código**, **padronização** e **controle de versão**, tanto na fase anterior à manutenção como na fase de manutenção. Essas três práticas são muito importantes para uma boa manutenção de código.

Outras práticas ágeis como **priorização de *backlog***, **testes unitários**, **reuso de código e cerimônias ágeis** foram citadas durante o desenvolvimento das tarefas, mas serão analisadas com mais detalhes na categoria IV de processo de manutenção.

- III- Testes: diferentes tipos de testes foram citados pelos participantes. O gráfico da Figura 21 apresenta que grande parte dos participantes informaram que executam testes manuais e testes unitários, tanto na fase anterior à manutenção como na fase de manutenção.

Testes de regressão são muito comuns em fase de manutenção pois eles são bastante utilizados quando existem mudanças em componentes, o que é muito comum ocorrer em fase de manutenção, mesmo evolutiva pois para criação de novas funcionalidades, pode ser necessário adaptar componentes já existentes no *software* (SOUZA; GUSMÃO; ROCHA, 2008). Entretanto, esses testes foram citados por apenas 4 participantes. Um fator que pode explicar esse número abaixo do esperado é que para utilizar testes de regressão, é necessário ter um cenário favorável porque ele pode ser muito custoso dependendo do tamanho do conjunto de casos de teste a serem executados (MALIMPENSA, 2018).

Figura 21 - Intermediárias VII e VIII - Tipos e Cobertura de Testes




Fonte: A autora, 2021.

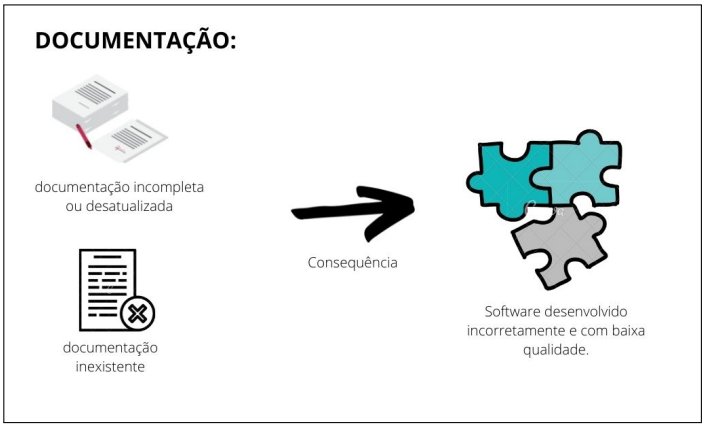
Outros tipos de testes foram informados pelos participantes, são eles, testes automatizados, exploratórios, integração, retro compatibilidade, API e performance. Além dos tipos de testes, participantes destacaram que durante a fase de manutenção, testes passaram a ser executados com mais frequência e maior cobertura, conforme apresentado na Figura 21.

- IV- Processo de manutenção: a qualidade é um fator importante a ser considerado em todo o processo de desenvolvimento de um *software*. Para manter essa qualidade, diferentes técnicas e práticas podem ser adotadas, proporcionando uma melhoria de processos (LIMA, 2017). Portanto, é necessário avaliar os problemas que foram destacados no desenvolvimento de *software*, as práticas mais utilizadas e como é realizado o acompanhamento da manutenção pelos especialistas que participaram da pesquisa.

A Figura 22 apresenta como os problemas de documentação levantados pelos participantes podem impactar na manutenção de *software*. Foram encontrados dois problemas comuns que são: (i) documentação incompleta ou desatualizada; e (ii) documentação inexistente. É muito comum que equipes que enfrentam esses problemas construam *softwares* que não atendem à expectativa do solicitante ou *softwares* de qualidade baixa. Portanto, é muito importante existir uma documentação atualizada e completa.

Além dos problemas de documentação, foram citados problemas de código, que

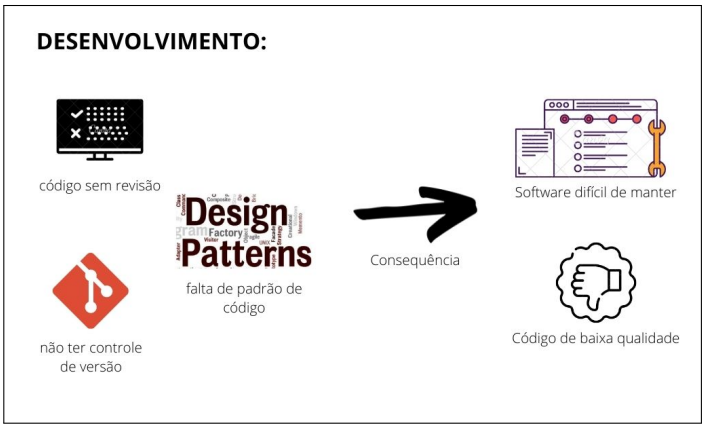
Figura 22 - Problemas de documentação




Fonte: A autora, 2021.

estão representados na Figura 23. Códigos sem revisão, falta de padrão de código e não ter controle de versionamento faz com que o código se torne cada vez mais difícil de manter e que ele possua baixa qualidade. Esses três itens levantados são muito importantes para *softwares* de grande escala e principalmente para fases de manutenção.

Figura 23 - Problemas de desenvolvimento

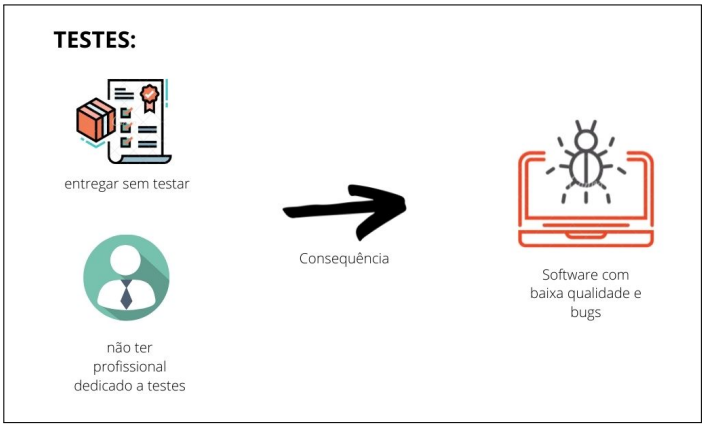



Fonte: A autora, 2021.

Entregar sem testes e não ter um profissional dedicado aos testes pode gerar um *software* com baixa qualidade e *bugs* que poderiam ser antecipados durante o desenvolvimento vão ser identificados após a entrega, conforme exibido na Figura 24.

Adotar práticas ágeis também auxiliam na boa manutenção de *software* e na qualidade. A partir da análise do gráfico da Figura 12, pode-se perceber que a execução de testes é uma prática muito utilizada, principalmente na fase de manutenção. Além da execução de testes, 45 participantes também citaram a reunião de planejamento (*planning*) e 42 a reunião diária (*daily*) como práticas utilizadas durante o processo de manutenção evolutiva de *software*. Conclui-se que mesmos equipes que não utilizam *frameworks* ágeis para todo o desenvolvimento, é comum adotar algumas práticas ágeis visando a melhoria

Figura 24 - Problemas de teste

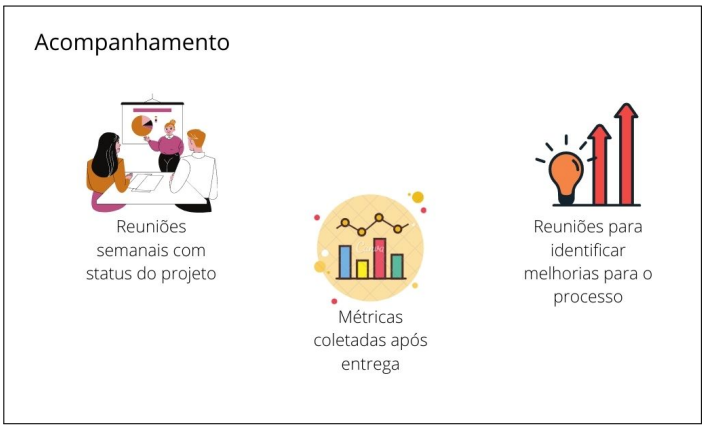



Fonte: A autora, 2021.

contínua durante a manutenção de *software*. Vale ressaltar que **todas** as práticas listadas no questionário foram votadas pelo menos 9 vezes.

Boas práticas unidas com um sólido processo de desenvolvimento ou manutenção de *software* tende a gerar *softwares* com sucesso. A Figura 25 exibe os três recursos mais citados pelos especialistas para acompanhamento de atividades de projeto e a busca pela melhoria contínua no processo. As reuniões semanais com status do projeto promove a visibilidade do andamento das atividades para todo o time e pessoas que atuam na gestão. Reuniões para identificar melhorias para o processo são importantes para análise do trabalho da equipe e construir planos de ação para promover melhoria contínua. As métricas são importantes para tomar decisões, é necessário identificar quais os critérios mais importantes para analisar e utilizar as métricas para refletir esses pontos.

Figura 25 - Acompanhamento




Fonte: A autora, 2021.

# 4 *FRAMEWORK* DE MANUTENÇÃO ÁGIL

O desenvolvimento do *framework* proposto foi elaborado em três etapas. Na primeira etapa, com base na literatura, foram definidos os passos para um *framework* de desenvolvimento de *software* de manutenção evolutiva utilizando as práticas ágeis mais vantajosas encontradas. Na segunda, um outro *framework* foi criado a partir da análise qualitativa das respostas do *survey* enviado para especialistas que atuam na área na indústria. E na última etapa, o *framework* final de manutenção evolutiva com práticas ágeis foi elaborado a partir dos modelos 1 e 2 criados anteriormente. A pesquisa foi de natureza quantitativa e qualitativa. Esta pesquisa foi útil para identificar as práticas mais utilizadas por profissionais na indústria de acordo com o que foi levantado na fase de elaboração do *framework* de desenvolvimento de *software* de manutenção evolutiva.

Após a validação das respostas do *survey*, foi realizada uma análise dos dados para verificar se o *framework* proposto a partir do mapeamento sistemático precisa de melhorias ou não. Para recrutamento dos profissionais da área de desenvolvimento de *software*, foi utilizado o método Bola de Neve (*snowballing*) que utiliza uma cadeia de referências para se obter uma amostra representativa (BALDIN; MUNHOZ, 2020). O questionário foi enviado para profissionais da área, que fazem parte da rede de conhecimentos dos pesquisadores, e cada profissional recomendou de 1 a 2 indivíduos, o que resultou em 50 participantes. Os dados foram analisados utilizando a técnica de *coding* citada na Seção 1.7.3 do Capítulo 1.

## 4.1 Modelo 1 - *Framework* a partir dos resultados do mapeamento sistemático

O modelo 1 do *framework* apresenta quais são as características do time responsável pela manutenção evolutiva do *software*, quais são as cerimônias ágeis mais recomendadas e as práticas para definição dos requisitos, desenvolvimento e testes mais utilizadas durante a fase de manutenção evolutiva de *software*.

Ao analisar os resultados do mapeamento sistemático da literatura apresentados na Seção 2.7 do Capítulo 2, pode-se observar a partir da Figura 6 que as práticas como "pequenas entregas", "equipe auto gerenciável" e "times pequenos" foram as mais citadas e portanto, são inseridas nas características da equipe responsável pela manutenção evolutiva, apresentadas na Figura 26. Para garantir a comunicação constante entre o time, recomenda-se equipes pequenas de 5 a 10 integrantes (JAVANMARD; ALIAN, 2015). Além disso, é importante que o time seja auto gerenciável para trabalharem em equipe e com autonomia suficiente para decidir a melhor forma de realizar o trabalho (BOGOJE-

VIĆ et al., 2017). As pequenas entregas são importantes para ter o constante *feedback* das partes interessadas e consequentemente melhorar qualidade do *software* que está sendo desenvolvido. A parte da definição dos requisitos, desenvolvimento e execução de testes está representada na Figura 27 e contém as seguintes partes principais: requisitos, desenvolvimento e testes.

Figura 26 - Modelo 1 - Processo de Manutenção ágil


Fonte: A autora, 2021.

As atividades de requisitos (ATR) estão representadas em azul na Figura 27 e dependendo de como foi realizada a fase anterior à fase de manutenção, a documentação pode seguir três opções diferentes: (ATR1) Fazer uma documentação simples e focada nas novas funcionalidades, (ATR2) Atualizar documentação antiga e detalhar as novas funcionalidades ou (ATR3) Atualizar documentação antiga e continuar conforme fase anterior. Após a documentação atualizada e refinada, os analistas de requisitos ou *product owners* apresentam as funcionalidades para os desenvolvedores durante a reunião de planejamento

(ATT1) da *sprint*. A reunião de planejamento é uma atividade de todo o time (ATT) e essas atividades estão destacadas em roxo na Figura 27.

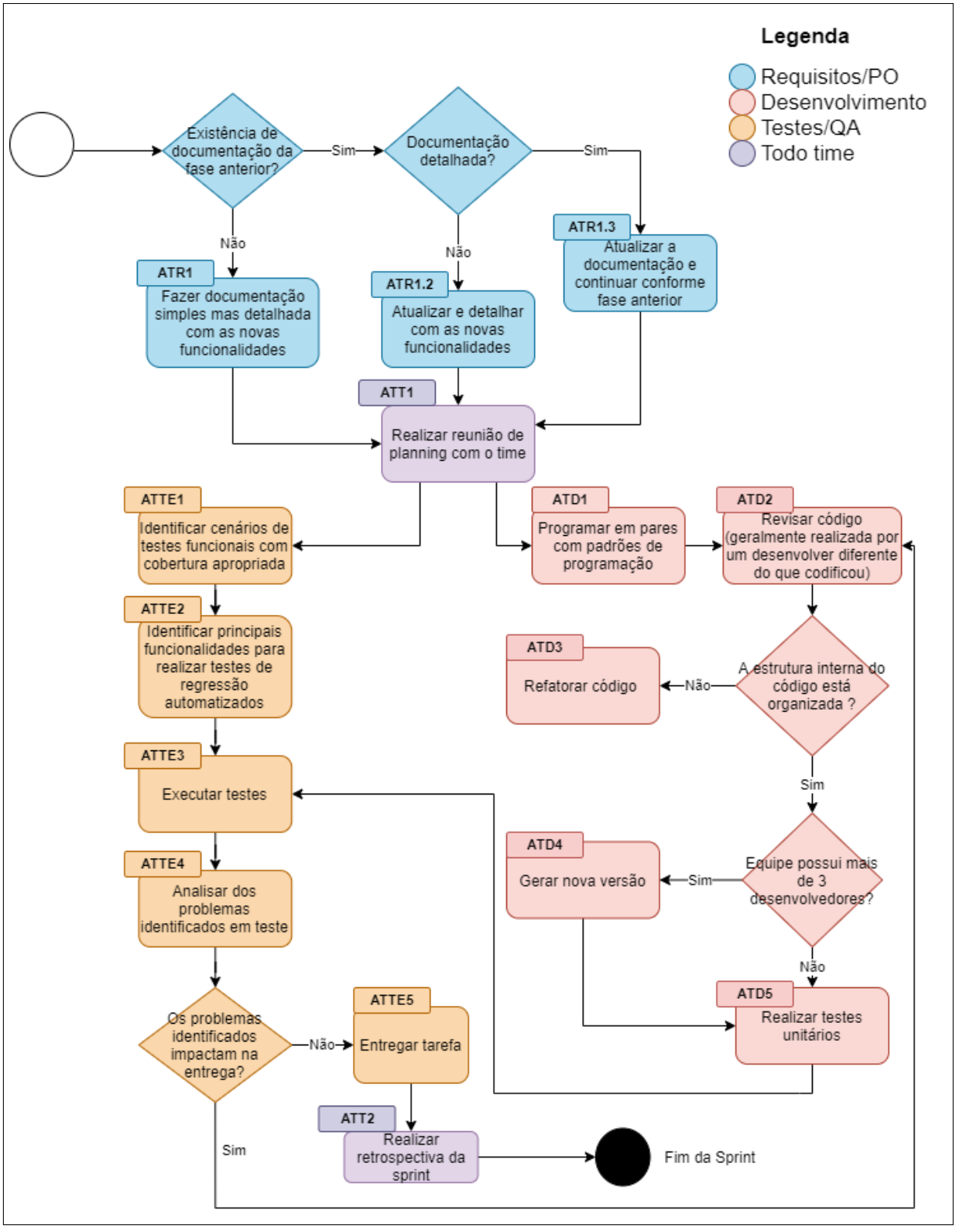
As atividades de desenvolvimento (ATD) estão representadas em rosa na Figura 27, foram utilizadas muitas práticas que foram analisadas no Capítulo 2. Após a reunião de planejamento, os desenvolvedores do time começam a programar em pares (ATD1) com o objetivo de manter um bom padrão do código e ao mesmo tempo que uma pessoa desenvolve, outra pessoa revisa (ATD2) e otimiza o código. Utilizar padrões de código e fazer revisão do mesmo são práticas importantes que devem ser realizadas por todos os desenvolvedores do time. Se a estrutura interna do código da fase anterior estiver organizada e dentro do padrão, o desenvolvimento segue sem alterações de código. Caso contrário, deve ser feito um refatoração (ATD3) no código afim de melhorá-lo, colocando dentro dos padrões estabelecidos pelo time de desenvolvimento.

Caso a equipe possua três ou mais desenvolvedores, deve existir um controle de versão (ATD4) de código para evitar que o trabalho de uma pessoa sobrescreva o que foi desenvolvido por outra. Antes de fazer a liberação para os testadores, são realizados testes unitários (ATD5) para assegurar que cada unidade está funcionando de acordo com o esperado. Após os testes unitários, as funcionalidades podem ser liberadas para testes funcionais e regressivos.

As atividades de teste (ATTE) estão representadas em laranja na Figura 27. A fase de teste se inicia junto com a fase de desenvolvimento. Enquanto os programadores estão desenvolvendo as funcionalidades da entrega, os testadores estão identificando os cenários de testes funcionais (ATTE1) e as principais funcionalidades que vão precisar de testes de regressão (ATTE2). Após os testes unitários realizados pelos programadores, a execução dos testes funcionais (ATTE3) se inicia. Depois que todos os cenários de testes forem executados, os testadores realizam uma análise dos problemas identificados (ATTE4) e caso esses problemas não tenham impacto crítico na entrega, a entrega (ATTE5) é efetuada. Caso contrário, o testador detalha os problemas e retorna para o programador analisar e corrigir. Após todos os problemas que foram identificados serem corrigidos e retestados (ATD2), a entrega especificada na reunião de planejamento pode ser realizada. Após a realização de todas as tarefas da iteração, o time realiza a reunião de retrospectiva (ATT2).

Se o *framework* proposto for utilizado dentro de um projeto de manutenção evolutiva de *software*, artefatos são gerados ao longo das entregas: (i) documentação de requisitos; (ii) casos de teste e (iii) documentação do código. O tipo de documentação de requisitos vai depender da fase anterior à fase da manutenção. O time junto com o *stakeholder* deve definir qual melhor tipo de documentação para o cenário do projeto.

Figura 27 - *Framework* de Manutenção Evolutiva - Modelo 1 (Mapeamento sistemático da literatura)



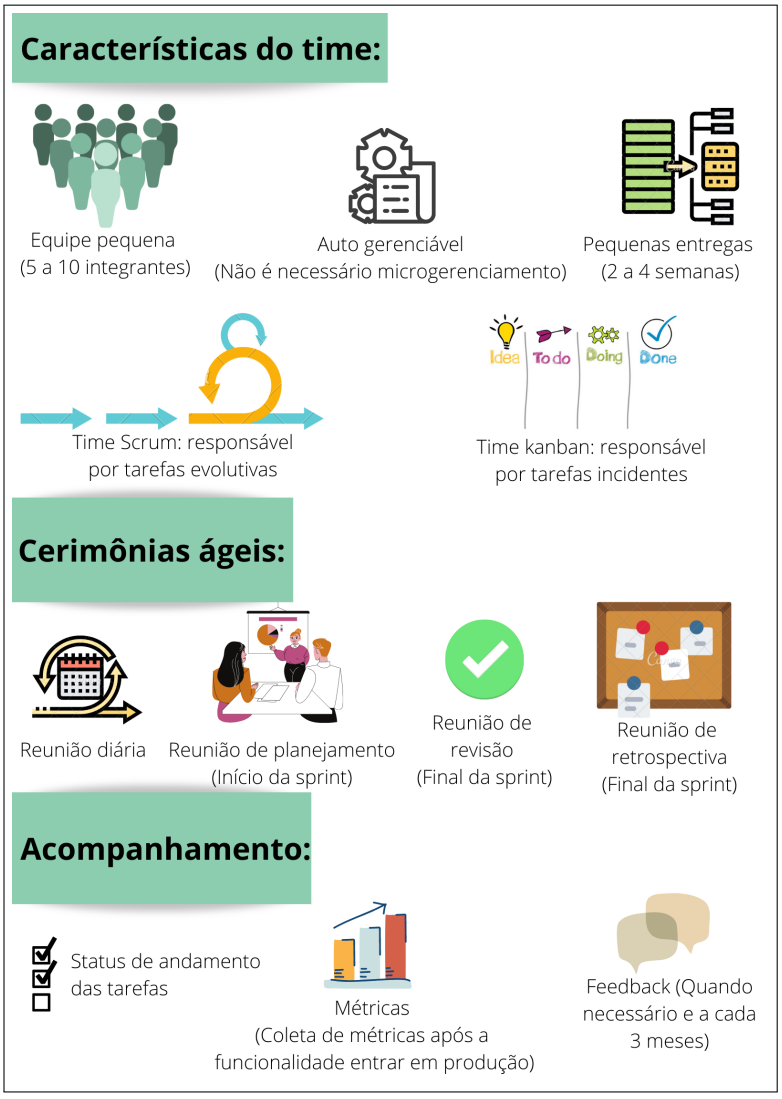

Fonte: A autora, 2021.

## 4.2 Modelo 2 - *Framework* a partir de *survey*

A elaboração do modelo de *framework* a partir dos resultados do *survey* foi realizada utilizando todos os pontos de destaque que foram levantados durante a análise detalhada dos dados coletados, descrita na seção 3.2.4 do Capítulo 3.

É importante estabelecer quais são as características do time, cerimônias ágeis recomendadas e atividades de acompanhamento da equipe de manutenção evolutiva de software. A Figura 28 apresenta a interpretação dos resultados da análise convertidos em informações gerais para definição do processo de manutenção evolutiva de *software*. A equipe deve ser pequena, composta por 5 a 10 integrantes, deve ser auto gerenciável, ou seja, capaz de priorizar suas tarefas e gerenciá-las, as entregas são feitas por iterações de no máximo quatro semanas e os times são divididos em Kanban ou Scrum. O time Kanban é responsável pelas tarefas incidentes e o time Scrum pelas tarefas evolutivas.

Figura 28 - Modelo 2 - Processo de manutenção ágil




Fonte: A autora, 2021.

Além das características do time, existem quatro cerimônias ágeis que foram muito citadas durante as respostas do *survey* e foram incorporadas no modelo 2 do *framework*, são elas, as reuniões diárias, reunião de planejamento, retrospectiva e *review*.

Existem práticas que são indispensáveis para fase de manutenção evolutiva, uma delas é a coleta de métricas após a funcionalidade entrar em produção. A partir dessas métricas, é possível fazer uma priorização de *backlog* mais assertiva e detalhamento das tarefas focado no usuário final.

Durante o andamento das *sprints*, podem surgir problemas que precisem de reuniões de *feedback* fora do planejado para reportar algum caso que ocorreu, tanto positivo como negativo, se aplica aos dois casos. Além dessas ocasiões não planejadas, é importante que o gestor realize *feedback* com uma periodicidade para apresentar ao membro do time seus pontos positivos e pontos a melhorar. Esses *feedbacks* são muito importantes para o crescimento do profissional.

Além disso, em alguns casos, *stakeholders*/cliente não estão acompanhando a rotina do time e portanto, a evolução não é transparente. Nesses casos, recomenda-se elaborar um *status* para refletir o andamento das tarefas.

O fluxo do modelo 2 está dividido em quatro cores diferentes que representa as atividades relacionadas a requisitos (ATR), desenvolvimento (ATD), testes (ATTE) e relacionadas à todo o time (ATT), conforme exibido na Figura 29.

Neste modelo, a manutenção evolutiva inicia com a realização de reunião com o cliente (ATR1) com objetivo de detalhar as necessidades e anotar os *feedbacks* em relação ao *software*. Em seguida, o PO do time vai realizar refinamento (ATR2) do *backlog* com o time, priorizar as tarefas (ATR3) e detalhá-las (ATR4).

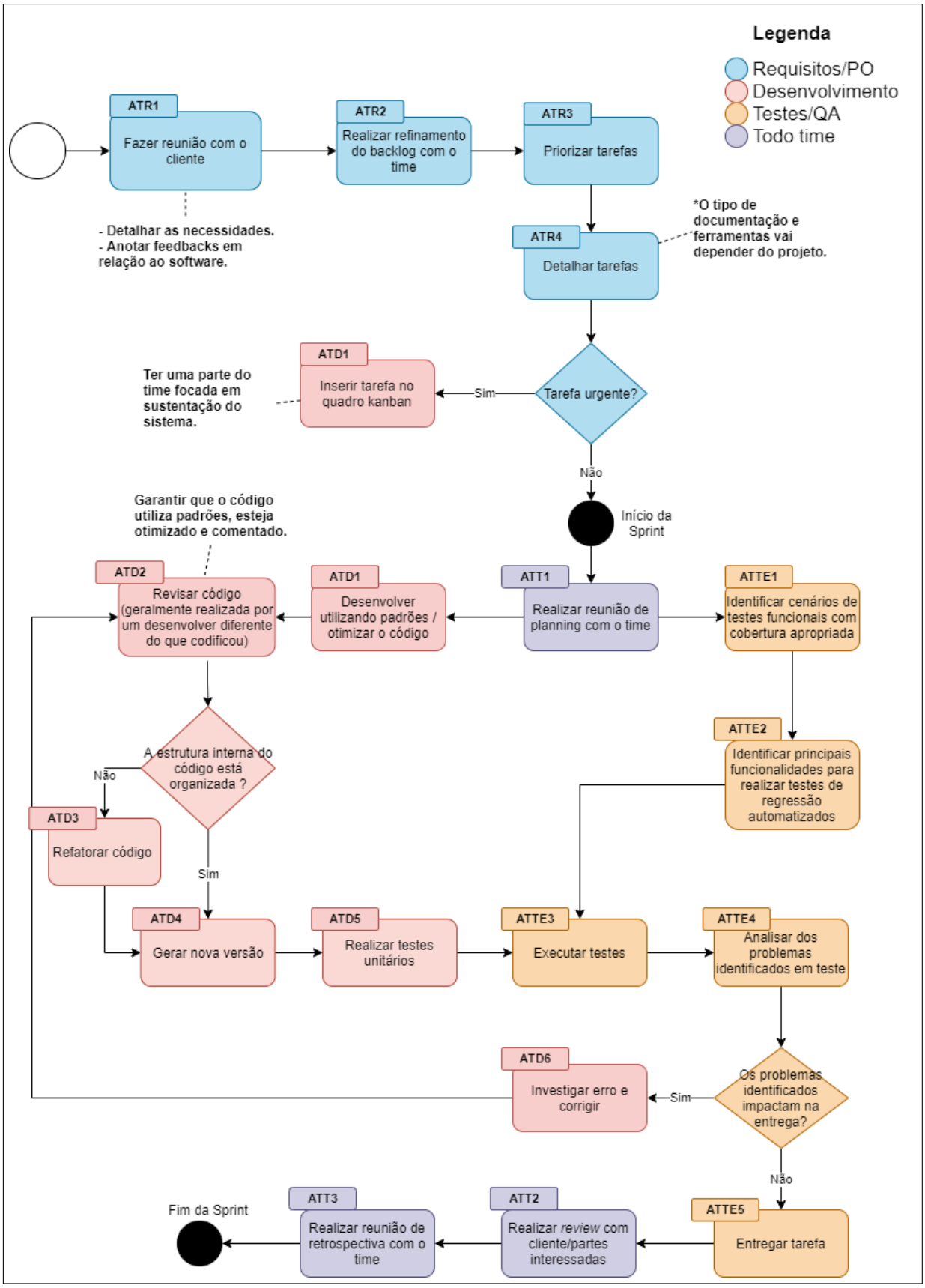
Durante a análise das respostas, participantes indicaram que dividem os times entre Scrum e Kanban (ATD1), time Scrum fica responsável pelas tarefas evolutivas e time Kanban pelas tarefas incidentes. Este fluxo foi representado no modelo 2 do *framework*.

Após as tarefas estarem detalhadas e classificadas, a *sprint* se inicia com a realização da reunião de planejamento com o time (ATT1). A reunião de planejamento foi citada 45 vezes nas respostas dos participantes. Após a reunião, os desenvolvedores vão começar a desenvolver as tarefas (ATD1) e os testadores iniciam a identificação dos cenários de testes funcionais (ATTE1) e quais funcionalidades serão realizados testes automatizados (ATTE2).

Durante o desenvolvimento das tarefas, muitas práticas citadas pelos participantes foram incorporadas no modelo: revisão de código (ATD2), utilização de padrões de projeto (ATD1), otimização de código (ATD1), refatoração (ATD3), versionamento do código (ATD4) e realização de testes unitários (ATD5).

Resumidamente, o desenvolvedor termina de realizar a tarefa, outro desenvolvedor do time vai revisar o código (ATD2) para garantir que o código está no padrão, otimizado e comentado. Caso a estrutura do código esteja organizada, uma nova versão é gerada,

Figura 29 - *Framework* de Manutenção Evolutiva - Modelo 2 (*Survey*)




Fonte: A autora, 2021.

o desenvolvedor executa testes unitários (ATD5) e libera para o time de qualidade testar (ATT3) a funcionalidade que foi desenvolvida.

O time de qualidade realiza a execução dos testes (ATTE3) de todos os cenários que foram identificados e analisa os erros encontrados (ATTE4). Se o erro não impactar a entrega, a tarefa é dada como pronta (ATTE5). Caso contrário, o time de desenvolvimento investiga o erro e realiza a correção (ATD6).

Ao final da *sprint*, é realiza a reunião de *review* (ATT2) onde o time apresenta todas as funcionalidades que foram desenvolvidas e testadas para os *stakeholders*. Antes de iniciar a próxima *sprint*, uma reunião de retrospectiva (ATT3) é realizada para identificar os pontos positivos e pontos de melhoria para as próximas *sprints*. As reuniões de *review* e retrospectiva foram citadas por aproximadamente 30 participantes.

## 4.3 Comparação do Modelo 1 com o Modelo 2

Iniciando a comparação pelas informações do processo de cada modelo, pode-se perceber que, através dos resultados do *survey*, elaborou-se um processo mais completo. Todas as características do time e cerimônias ágeis descritas no modelo 1 estão contempladas no modelo 2, portanto, o modelo 2 foi utilizado para gerar a versão final do processo de manutenção evolutiva na Seção 4.4.

Fazendo uma análise visual dos dois modelos de *frameworks* construídos, o modelo 1 possui menos etapas que o modelo 2. O modelo 1 começa com uma série de condições para definir qual melhor tipo de documentação a ser utilizado. Como os dados coletados no questionário não tiveram destaque apenas no tipo da documentação, mas no processo completo da construção da mesma, o modelo 1 e 2 se complementam.

Um ponto de destaque que não tem no modelo 1 e possui no modelo 2 é a definição da urgência das tarefas e dividir o time para diferentes tipos de tarefas. Time Scrum é o time responsável pela manutenção evolutiva e as tarefas evolutivas e o time Kanban é responsável pelas tarefas urgentes, focadas na sustentação do sistema. Esse modelo de times é muito utilizado principalmente para sistemas e produtos que já estão em produção. É importante separar os times para não impactar as entregas de evolução.

A reunião de planejamento realizada no início da *sprint* é prática comum entre os dois modelos. Programação em pares foi uma técnica bastante citada nos trabalhos encontrados, porém apenas aproximadamente 30% dos participantes do *survey* responderam utilizar a prática de programação em pares no dia a dia, portanto, esta prática não foi inserida no modelo 2.

As práticas de revisão de código, refatoração, versionamento e testes unitários foram muito destacadas em ambas as fontes e inseridas nos dois modelos. O modelo 1 apresenta a condição de existir mais de 3 desenvolvedores para realização da revisão de

código, mas durante a análise de dados do *survey*, nenhum participante indicou o número de desenvolvedores como premissa para realização da revisão de código.

O fluxo de testes é exatamente igual nos dois modelos, a diferença está na entrega. No modelo 1, as cerimônias finais não foram colocadas, a entrega é realizada assim que todos os defeitos que possuem impactos são corrigidos e retestados com sucesso. O modelo 2 apresenta duas práticas muito utilizadas no desenvolvimento de *software* que também são muito úteis e proveitosas na manutenção evolutiva, são elas, reunião de *review* com os *stakeholders* para apresentar as funcionalidades desenvolvidas e reunião de retrospectiva com o time para identificar os pontos de melhoria.

A partir dessa análise, pode-se verificar que o resultado encontrado no mapeamento sistemático está muito próximo com o que especialistas estão utilizando em projetos na indústria e na academia.

## 4.4 ***Framework* final**

A versão final do *framework* contempla o modelo 1 criado com base na teoria identificada nos resultados do mapeamento sistemático da literatura do Capítulo 2 em conjunto com o modelo 2 que foi construído de acordo com as práticas reportadas pelos participantes do *survey* apresentados no Capítulo 3. A comparação descrita na Seção 4.3 auxiliou no processo da elaboração final do *framework*.

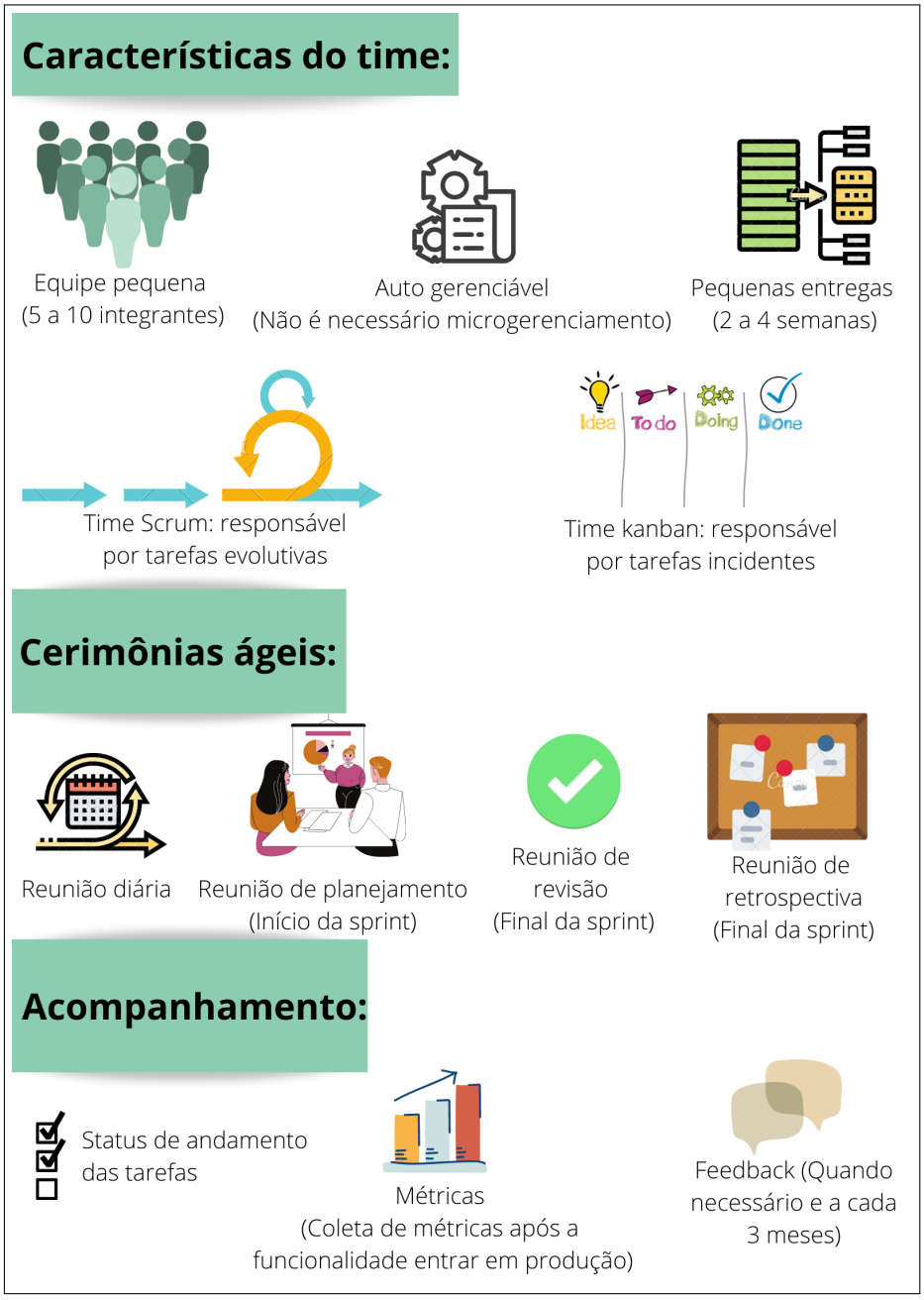
O processo da manutenção está exibido na Figura 30. Para a manutenção evolutiva de *software*, recomenda-se que a equipe seja pequena, com até dez integrantes. Também é importante que a equipe consiga auto gerenciar as tarefas necessárias. As entregas deverão ser realizadas entre duas a quatro semanas e se o cenário tiver tarefas que precisam de soluções imediatas, recomenda-se ter dois times. Um time dedicado a resolução de incidentes: time Kanban, e outro time dedicado as tarefas evolutivas: time Scrum.

O time Kanban vai realizar atualizações no *software* sempre que necessário para atender a demanda incidente. O time Scrum vai ter datas pré-estabelecidas para realizar as entregas do que foi desenvolvido.

As cerimônias ágeis essenciais para a manutenção evolutiva são: (i) reunião diária, (ii) reunião de planejamento, (iii) retrospectiva, e (iv) *review*. Além das cerimônias, é muito importante que as pessoas que exercem o papel de *scrum master* e PO se reúnam para apresentação do andamento das atividades para os *stakeholders*.

Recomenda-se que o PO utilize técnicas de coleta de dados para identificar métricas com objetivo de melhorar o produto, trazendo benefícios para os *stakeholderes* e o usuário final.

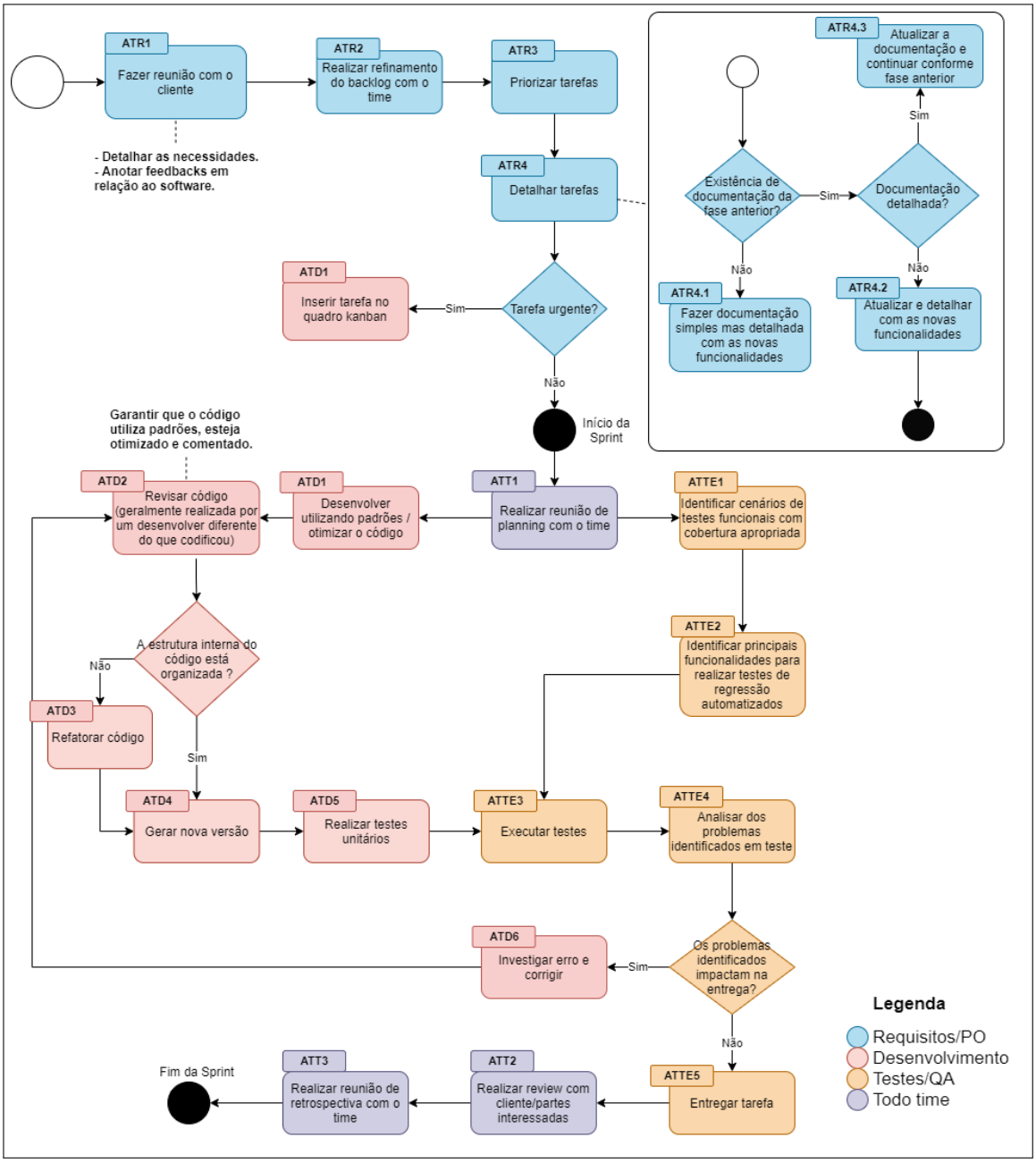
Figura 30 - Processo de manutenção ágil




Fonte: A autora, 2021.

A versão final do *framework* está exibida na Figura 31 com todas as fases que a equipe participa durante a manutenção evolutiva de *software*.

Figura 31 - *Framework* - Versão final




Fonte: A autora, 2021.

Vale ressaltar que, artefatos serão gerados ao longo das entregas: (i) documentação de requisitos; (ii) casos de teste; (iii) documentação do código e (iv) métricas.

# 5 PROVA DE CONCEITO

De acordo com Guimarães et al. (2008), define-se *prova de conceito (usa-se no texto a sigla POC, do inglês Proof of Concept) como uma técnica que permite demonstrar que uma determinada ideia é tecnicamente possível, ou seja, pode ajudar a verificar se uma arquitetura é 'construível'".*

Este capítulo tem como objetivo realizar uma simples POC para apresentar como o *framework* é utilizado. Um cenário hipotético foi utilizado apenas para demonstração da utilização do *framework* em um cenário de manutenção de *software*.

## 5.1 Contexto inicial

A empresa "Limpa Show" é uma distribuidora de produtos de limpeza de diversas marcas para pequenas e médias empresas que desejam comprar no atacado. Os vendedores realizam as vendas dos produtos para clientes cadastrados através de chamada telefônica ou mensagem por aplicativo. A empresa está crescendo e começando a atender clientes internacionais, exportando os produtos para outros países.

Com essa expansão para clientes internacionais, os vendedores estão tendo dificuldade para realizar o cálculo do preço dos produtos em outra moeda e também calcular o frete para outros países, já que o sistema atual só possui as localidades dentro do Brasil.

Além dessas funcionalidades não existentes no sistema, os usuários também identificaram alguns defeitos e melhorias que devem ser realizadas. A partir desses *feedbacks* dos vendedores, a gestão decidiu contratar a empresa de Tecnologia da informação (TI) que desenvolveu o sistema para implementar novas funcionalidades e corrigir defeitos.

A vendedora Olívia é a funcionária mais antiga na empresa e tem bastante conhecimento do sistema. Ela realizou uma pesquisa com todos os vendedores para coletar as deficiências do sistema e fazer essa interface com a empresa de TI. Ela vai ser a responsável pelas reuniões com o PO da empresa contratada.

## 5.2 Processo de manutenção ágil

De acordo com o cenário compartilhado pela empresa "Limpa Show", a equipe terceirizada de TI fez uma análise e foi definido:

- Características do time: A equipe contratada para essa fase de manutenção evolutiva é composta por um PO, três desenvolvedores, um *Scrum Master* e dois testadores. O time escolhido é formado por pessoas que já atuam juntos em outros projetos

e possuem perfil auto gerenciável e uma excelente comunicação. As entregas serão realizadas a cada três semanas para o cliente “Limpa Show”. Como a empresa apresentou necessidade de correção de defeitos, desenvolvimento de melhorias e novas funcionalidades. Terão dois times para atender as diferentes demandas. Time Scrum: Responsável pelas tarefas evolutivas com entregas a cada 3 semanas. Time Kanban: Responsável pela correção de tarefas incidentes com entregas de acordo com a necessidade.

- Cerimônias ágeis: Todas as cerimônias ágeis recomendadas no *framework* na Seção 4.4 do Capítulo 4 serão realizadas pelo time Scrum. O time kanban realizará a reunião diária e a cada 4 semanas, reunião de retrospectiva para levantar pontos de destaque e melhorias.
- Acompanhamento: Foi decidido entre contratante e contratada que uma reunião semanal será realizada para contratada apresentar o andamento das atividades. Métricas serão coletadas depois que as novas funcionalidades forem para produção. O *feedback* da empresa contratante com a contratada será realizado a cada três meses, a partir de pontos que forem sendo identificados ao longo das *sprints*.

### 5.2.1 Equipe completa

- Olívia - Responsável do lado do cliente (usuária)
- Paula - *Product Owner*
- **Time Scrum**
  1. Bruna - *Scrum Master*
  2. Daniel - Desenvolvedor 1
  3. João - Desenvolvedor 2
  4. Lívia - Testadora 1
- **Time Kanban**
  1. Carolina - Desenvolvedora 3
  2. Andreza - Testadora 2

## 5.3 **Reunião com o cliente - ATR1**

Olívia se reúne com Paula (PO), Daniel (Desenvolvedor 1) e Bruna (*Scrum Master*) para descrever as necessidades e apresentar os resultados da sua pesquisa com os outros usuários do sistema.

Anotações da Paula (PO) da primeira reunião com o usuário (ATR1) final:

- Funcionalidades, defeitos e melhorias:
  - Funcionalidade para converter valor do produto de acordo com a cotação diária da moeda escolhida pelo vendedor no momento que estiver realizando a venda (Os vendedores estão realizando o cálculo manualmente no momento da compra, pesquisando o valor da cotação e realizando a conversão, gerando uma demora para informar para o cliente o valor do produto);
  - Cálculo de frete para outros países (os vendedores não conseguem realizar o cálculo no momento da venda, é necessário entrar em outro sistema ou ligar para transportadora para verificar o preço para outro país: Vendas já foram perdidas por esse problema);
  - Envio de *e-mail* de propaganda de novo produto no portfólio para clientes cadastrados;
  - Lentidão no momento do envio de *e-mail* com confirmação de compra para cliente;
  - Não está sendo possível adicionar mais de mil itens dentro de uma venda. O vendedor quando faz uma venda maior que mil itens precisa separar em duas ou mais vendas;
  - Mensagem de alerta quando usuário não preenche o valor do produto na tela de edição de produto está sendo exibida com o texto "O valor do produto está incorreto", mas o texto esperado é "O campo **valor do produto** é obrigatório";
  - O botão de "Salvar" no cadastro de produto só é habilitado quando todas as informações obrigatórias são preenchidas. Na tela de edição, o botão de "Salvar" deve ter o mesmo comportamento, mas está sempre habilitado, mesmo se o usuário apagar algum campo de preenchimento obrigatório.
- Detalhes da funcionalidades
  - Países para conversão de valores e cálculo de frete: Estados Unidos, Canadá, Argentina, Chile, México, Portugal e Espanha;
  - Texto padrão para envio de *e-mail*: Olivia ficou de definir e enviar.
- Prioridade das tarefas evolutivas

– Conversão de valor do produto para moedas estrangeiras;

– Cálculo de frete para outros países;

– Envio de *e-mail* de propaganda.

## 5.4 **Refinamento do *backlog* - ATR2**

Após a reunião com a usuária Olívia, Paula (PO) se reuniu com o time para refinar e começar a especificar as tarefas que serão desenvolvidas pelo time. O time desenhou um protótipo de baixa fidelidade para validar com a usuária o comportamento das funcionalidades. Os protótipos de baixa fidelidade estão exibidos nas figuras 32 e 33.

Figura 32 - Protótipo de baixa fidelidade - História 1


Fonte: A autora, 2021.

Além do protótipo, Paula realizou anotações para detalhar as histórias, essas anotações foram estão detalhadas na Seção 5.6 deste capítulo.

Figura 33 - Protótipo de baixa fidelidade - História 2


Fonte: A autora, 2021.

## 5.5 **Priorização de tarefas - ATR3**

A priorização de tarefas foi realizada pela usuária durante a reunião de detalhamento das necessidades.

## 5.6 **Detalhamento das tarefas - ATR4**

Antes de realizar o detalhamento das tarefas, Paula (PO) teve que definir qual seria o formato de documentação utilizado.

### 5.6.1 Definição da documentação

Paula (PO) identificou que existia documentação da fase de desenvolvimento do *software* e a documentação estava detalhada e atualizada, portanto, não foi necessário atualizá-la. Olívia informou que não tiveram alterações no sistema desde a última atualização na documentação. A documentação da fase anterior à de manutenção era Caso de Uso, mas a Olívia pediu uma documentação mais simples (ATR4.1) e Paula sugeriu história de usuário. Foi definido que as novas funcionalidades da fase de manutenção seriam documentadas em formato de História de usuário. E para o cadastro de defeitos e melhorias, o *card* no quadro de tarefas seria criado com descrição do que deve ser

realizado.

### 5.6.2 Histórias de usuário

1. **História 1:** Converter valor do produto para outras moedas **Como** vendedor, eu **quero** converter o valor de um produto, **para** realizar uma venda para clientes internacionais

   **Critérios de aceite:**

   (a) A *combobox* "Moeda" deve ter "real (R$)" como valor padrão;

   (b) A *combobox* "moeda" deve exibir as opções: "dólar americano (US$)", "dólar canadense (C$)", "peso argentino ($)", "peso chileno (CLP$)", "peso mexicano (MXN$)", "euro (€)";

   (c) A *combobox* "moeda" é de seleção única;

   (d) A conversão deve ser exibida com o acionamento do botão "Converter valor";

   (e) Uma modal de mensagem de confirmação "Você tem certeza que deseja converter o valor do produto para <moeda selecionada>?" deve ser exibida para usuário no momento de fechar a venda;

   (f) As opções "Sim" e "Não" devem ser exibidas na modal de confirmação;

   (g) O valor da cotação diária deve ser exibido ao lado da *combobox* ex: <símbolo moeda selecionada> 1,00 = R$<valor cotação diária>.

   **Tarefas:**

   (a) Tarefa 1: Criar *combobox* para seleção da moeda;

   (b) Tarefa 2: Criar lógica para buscar cotação do dia;

   (c) Tarefa 3: Criar lógica do cálculo de conversão;

   (d) Tarefa 4: Implementar mensagem de aviso;

   (e) Tarefa 5: Desenvolver *front-end*;

   (f) Tarefa 6: Criar teste unitário;

   (g) Tarefa 7: Elaborar Casos de teste funcionais e de regressão;

   (h) Tarefa 8: Executar testes funcionais e de regressão.

2. **História 2:** Calcular frete para outros países **Como** vendedor, eu **quero** calcular frete para outros países, **para** realizar uma venda para clientes internacionais

   **Critérios de aceite:**

(a) Um novo campo “País” deve ser adicionado;

(b) O campo “Estado” deve ser atualizado de acordo com o país selecionado;

(c) No campo “País” deve ser exibido a lista dos países: “Estados Unidos”, “Canadá”, “Argentina”, “Chile”, “México”, “Portugal” e “Espanha”;

(d) O valor do frete é exibido na moeda selecionada no campo “Moeda”.

**Tarefas:**

(a) Tarefa 1: Aumentar o raio do mapa;

(b) Tarefa 2: Criar campo “País”;

(c) Tarefa 3: Implementar lógica dos campos “País” e “Estado”;

(d) Tarefa 4: Atualizar banco de dados com países e estados;

(e) Tarefa 5: Desenvolver *front-end*;

(f) Tarefa 6: Criar teste unitário;

(g) Tarefa 7: Elaborar Casos de teste funcionais e de regressão;

(h) Tarefa 8: Executar testes funcionais e de regressão.

3. **História 3:** Enviar *e-mail* de propaganda de novo produto **Como** administrador, eu **quero** ter a opção de enviar *e-mail* de propaganda de novos produtos do portfólio toda vez que um produto novo for cadastrado, **para** divulgar para os clientes

   **Critérios de aceite:**

   (a) Modal “Novo Produto” deve ser exibida no final do cadastro de um produto;

   (b) A modal “Novo Produto” deve conter os *radio buttons* “Enviar *e-mail* sobre o produto”, “Não Enviar”, o campo “Enviar para” e o botão “OK”.

   (c) A opção “Enviar *e-mail* sobre o produto” é selecionada por padrão.

   (d) O campo *combobox* de seleção múltipla “Enviar Para” deve exibir a lista de clientes cadastrados;

   (e) Mensagem de sucesso “*E-mail* enviado com sucesso.” é exibida após selecionar o botão “OK”.

   (f) O *e-mail* deve seguir o *template*: “*Assunto: [Limpa Show] Novidade: Produto “<nome do produto” Corpo de texto: Conheça o nosso novo produto <nome do produto> que possui as funções de <descrição das funções do produto> por apenas <valor do produto> ! Entre em contato com o seu vendedor para mais informações. Estaremos enviando uma amostra do produto no seu próximo pedido com a loja!*”

**Tarefas:**

(a) Tarefa 1: Criar modal “Novo Produto”;

(b) Tarefa 2: Implementar lógica do campo “Enviar Para”;

(c) Tarefa 3: Implementar mensagem de sucesso;

(d) Tarefa 4: Implementar *e-mail* de novo produto;

(e) Tarefa 5: Criar serviço de envio de *e-mail*;

(f) Tarefa 6: Desenvolver *front-end*;

(g) Tarefa 7: Criar teste unitário;

(h) Tarefa 8: Elaborar Casos de teste funcionais;

(i) Tarefa 9: Executar testes funcionais.

*Não é necessário teste de regressão para essa história.

### 5.6.3 Defeitos e melhorias

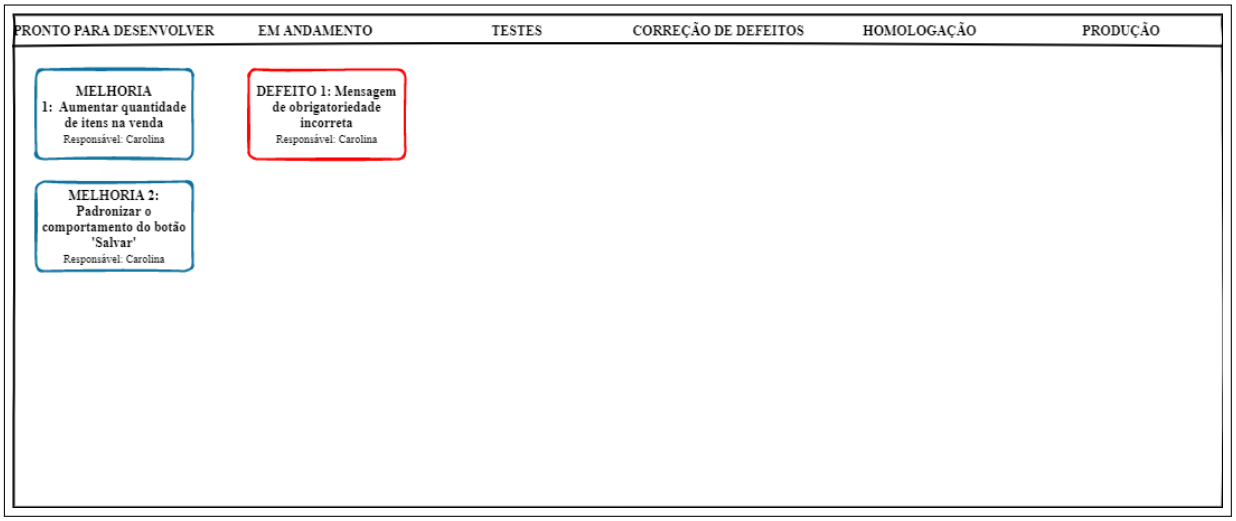
- **Melhoria 1:** Aumentar quantidade de itens na venda **Descrição:** É necessário aumentar a quantidade de itens para “100.000” no carrinho para realização de uma venda.
- **Defeito 1:** Mensagem de obrigatoriedade incorreta **Descrição:** Quando usuário não preenche o campo “Valor” na tela de edição de produto, o sistema está exibindo a mensagem “O valor do produto está incorreto”, a mensagem esperada é “O campo **valor do produto** é obrigatório”.
- **Melhoria 2:** Padronizar o comportamento do botão “Salvar” no cadastro de produto **Descrição:** O botão de “Salvar” no cadastro de produto só é habilitado quando todas as informações obrigatórias são preenchidas. Na tela de edição, o botão de “Salvar” deve ter o mesmo comportamento.

## 5.7 **Quadro Kanban - ATD1**

Todas as melhorias e os defeitos que são informados pela Olívia, a Bruna cria no quadro de Kanban. O quadro está exibido na figura 34. Defeitos são representados em vermelho e melhorias em azul. O quadro possui seis colunas que representa cada fase que aquele defeito ou melhoria pode passar, são elas (i) Pronto para desenvolver, ou seja, a tarefa está com todos os requisitos necessários descritos para ser desenvolvida; (ii)

Desenvolvendo, quando está em fase de desenvolvimento; (iii) Pronto para teste, está no ambiente de teste para o testador executar o teste; (iv) Correção de defeitos, a tarefa é incluída nesta coluna caso o testador encontre algum defeito para ser corrigido que tenha sido gerado pela tarefa; (v) Homologação, quando a tarefa está em homologação com a Olívia; (vi) Produção, tarefa validada pela Olívia e já está no sistema em produção. Essas mesmas colunas são exibidas para as tarefas do time de Scrum.

Figura 34 - Quadro Kanban




Fonte: A autora, 2021.

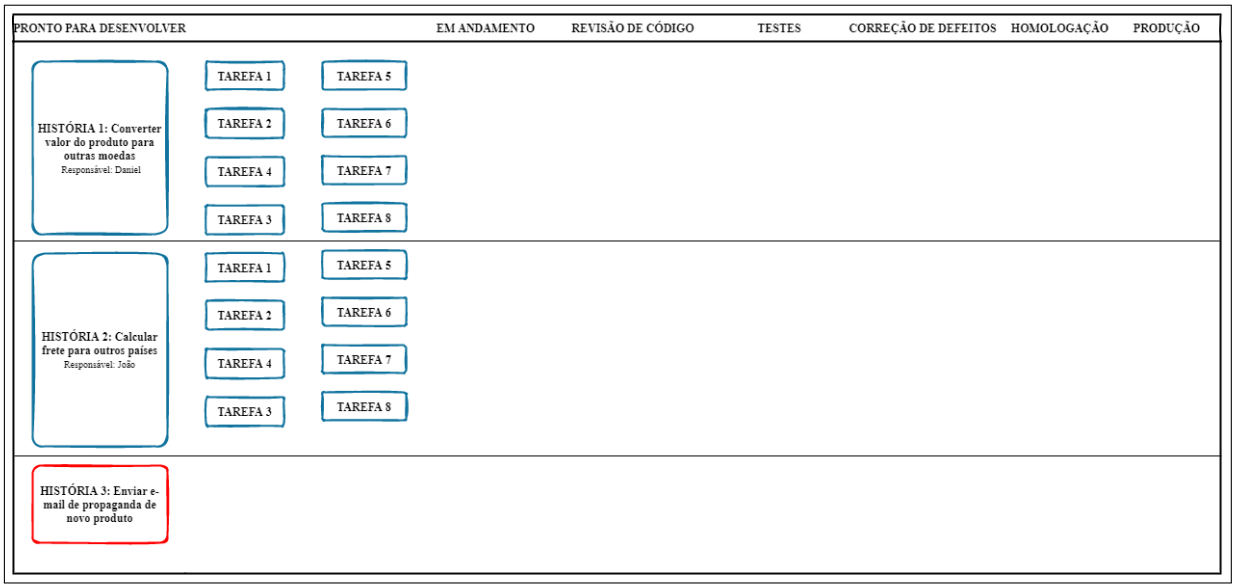
## 5.8 **Andamento da *Sprint* 1**

### 5.8.1 DIA 1: Reunião de planejamento - ATT1

- Time Scrum: No primeiro dia da *sprint* do time Scrum, Paula (PO) apresenta as histórias 1, 2 e 3 para o time durante a reunião de planejamento. O time de desenvolvimento avaliou que dentro de três semanas, apenas as histórias 1 (Converter valor do produto para outras moedas) e 2 (Calcular frete para outros países) conseguiriam ser entregues completamente. Portanto, a história 3 ficará como extra. Foi definido que cada desenvolvedor ficaria responsável por uma história e o quadro de histórias com as tarefas foi criado e está exibido na figura 35. Bruna (*Scrum Master*) alertou o time Scrum que, o time Kanban possui 2 melhorias e 1 defeito para desenvolver, se essas melhorias e o defeito forem corrigidos e não surgirem novos, a Carolina (desenvolvedora) pode ajudar o time Scrum nas entregas.
- Time Kanban: Carolina (desenvolvedora) inicia a correção do defeito (Mensagem

de obrigatoriedade incorreta), conforme exibido na figura 34.

Figura 35 - Quadro Time Scrum detalhado com tarefas



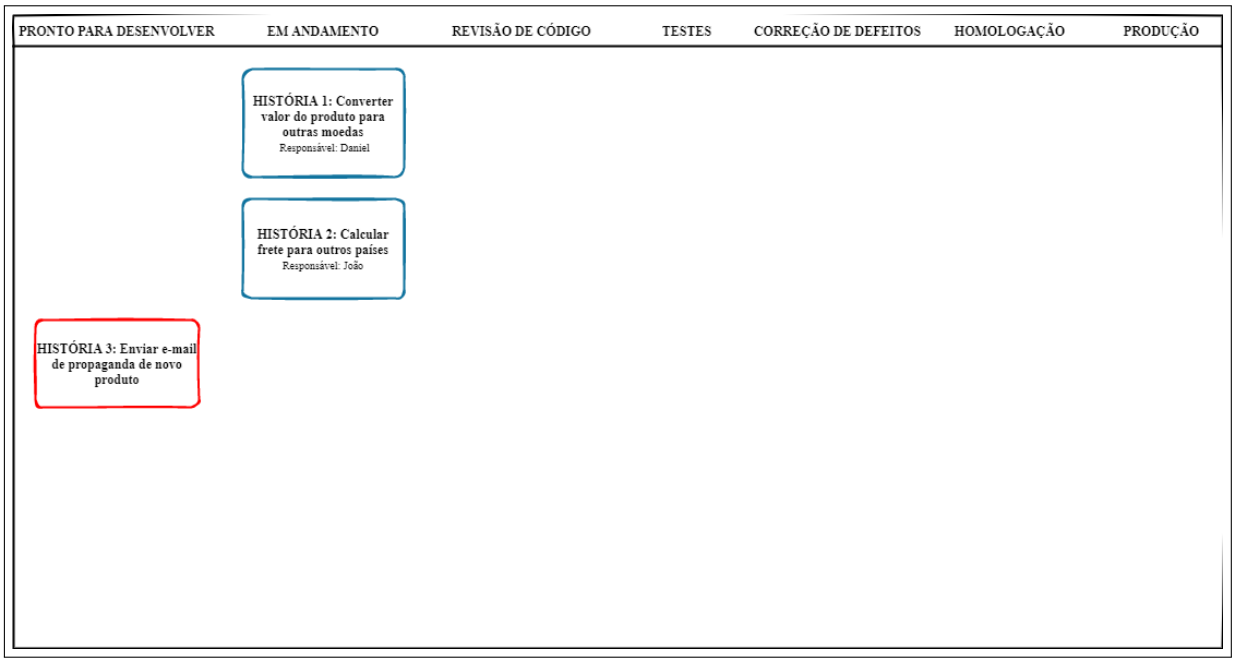

Fonte: A autora, 2021.

Apesar do quadro ter sido detalhado com as tarefas, como este capítulo trata de uma POC, o andamento das tarefas é refletido com o andamento das histórias.

### 5.8.2 DIA 2 - ATD1, ATTE1 e ATTE2:

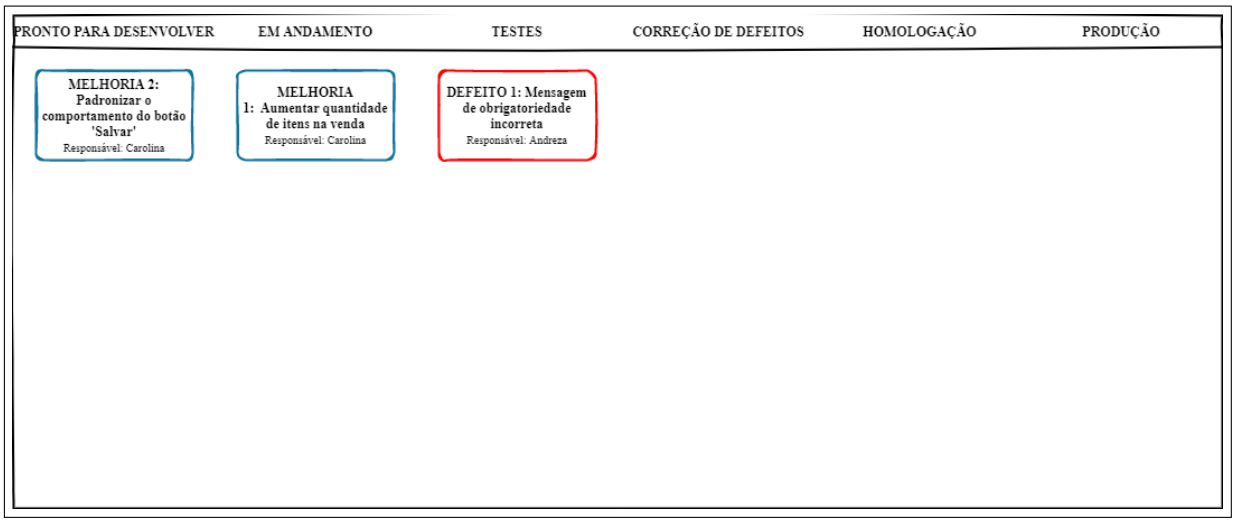
- Time Scrum: Desenvolvedor Daniel começa a desenvolver a tarefa 1 (Criar *combobox* para seleção da moeda) da história de conversão de valor do produto para outras moedas e o desenvolvedor João a tarefa 1 (aumentar o raio do mapa) da história de cálculo do frete para outros países, conforme exibido na figura 36. Lívia começa a tarefa 7 (elaborar casos de teste funcionais e de regressão) da história 1.
- Time Kanban: Carolina termina de desenvolver o defeito 1 (mensagem de obrigatoriedade incorreta) e move o *card* para a coluna “Pronto para teste” e inicia o desenvolvimento da melhoria 1 (aumentar quantidade de itens na venda), conforme exibido na figura 37.

Figura 36 - Quadro Time Scrum - Dia 2




Fonte: A autora, 2021.

Figura 37 - Quadro Time Kanban - Dia 2

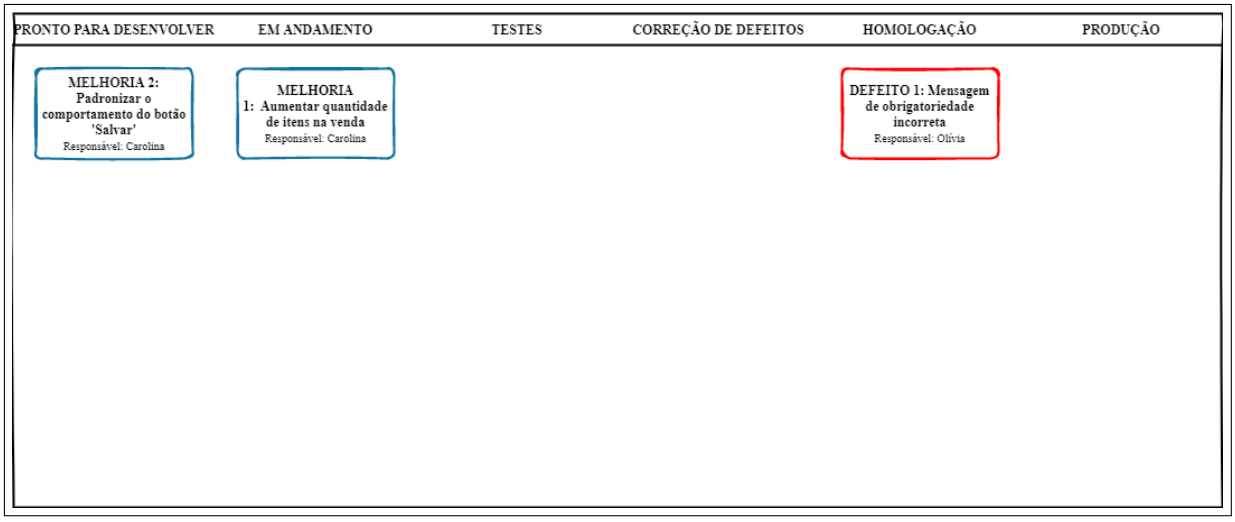



Fonte: A autora, 2021.

### 5.8.3 DIA 3 - ATD1, ATTE1, ATTE2 e ATD5:

- Time Scrum: Desenvolvedor Daniel termina a tarefa 1 (criar *combobox* para seleção da moeda) e João termina tarefa 1 (aumentar o raio do mapa) de suas respectivas histórias. As histórias continuam em andamento pois tem outras tarefas para concluírem, inclusive os testes unitários. Lívia continua na tarefa 7 (elaborar casos de teste funcionais e de regressão) da história 1. O quadro do time Scrum não altera.
- Time Kanban: Carolina continua no desenvolvimento da melhoria 1 de aumentar a quantidade de itens na venda, conforme exibido na figura 38. O teste do defeito 1 foi realizado pela Andreza e enviado para homologação.

Figura 38 - Quadro Time Kanban - Dia 3




Fonte: A autora, 2021.

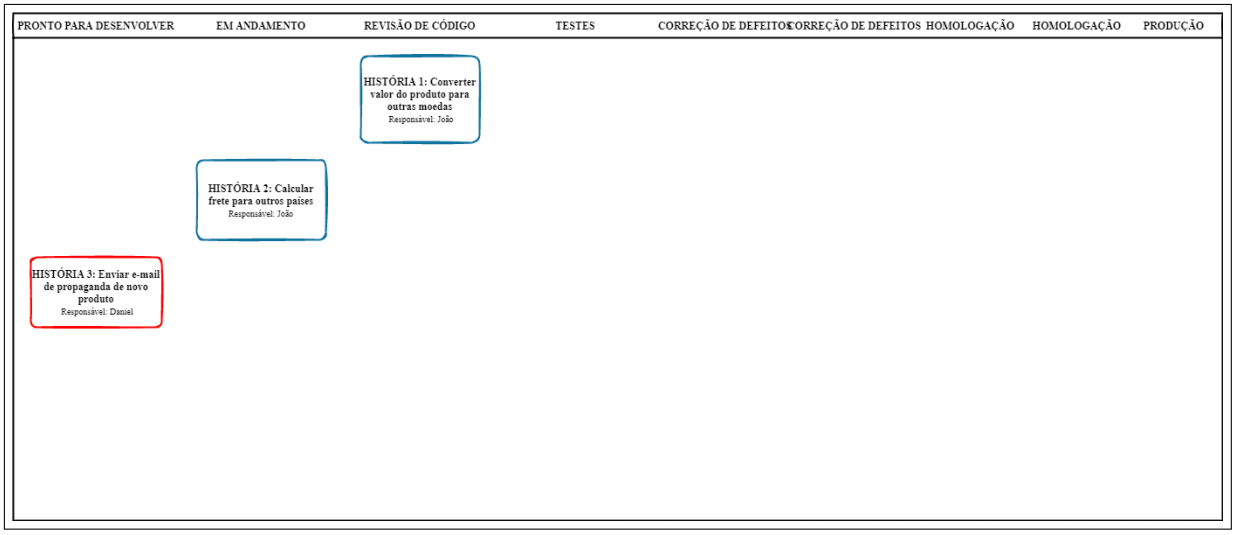
### 5.8.4 DIA 4 e 7 - ATD1, ATD2, ATTE1 e ATTE2:

- Time Scrum: No dia 4, Lívia finalizou a tarefa 7 (elaborar casos de teste funcionais e de regressão) da história 1 (Converter valor do produto para outras moedas) e começou a tarefa 7 (elaborar casos de teste funcionais e de regressão) da história 2 (calcular frete para outros países). No dia 7, desenvolvedor Daniel finaliza a última tarefa da história 1 e libera a história 1 para revisão de código. Como Daniel estava aguardando a revisão de código da história 1 , iniciou o desenvolvimento da história 3 (enviar *e-mail* de propaganda de novo produto). O desenvolvedor João está na tarefa 5 (desenvolver *front-end*) da história 2 e em paralelo está realizando a revisão

de código da história liberada por Daniel. O quadro do sétimo dia de *sprint* está exibido na figura 39.

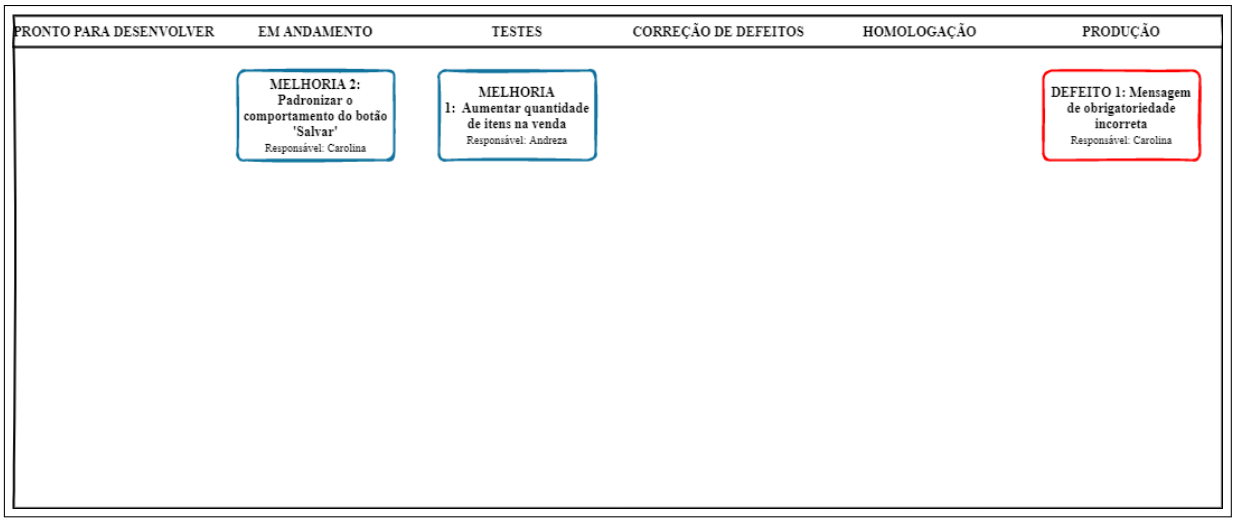
- Time Kanban: Carolina libera a melhoria 1 (aumentar a quantidade de itens na venda) para Andreza testar no dia 7 e inicia o desenvolvimento da melhoria 2 (padronizar o comportamento do botão 'Salvar'). O defeito 1 (mensagem de obrigatoriedade incorreta) já foi corrigido em produção. O quadro Kanban do dia 7 está exibido na figura 40.

Figura 39 - Quadro Time Scrum - Dia 7




Fonte: A autora, 2021.

Figura 40 - Quadro Time Kanban - Dia 7

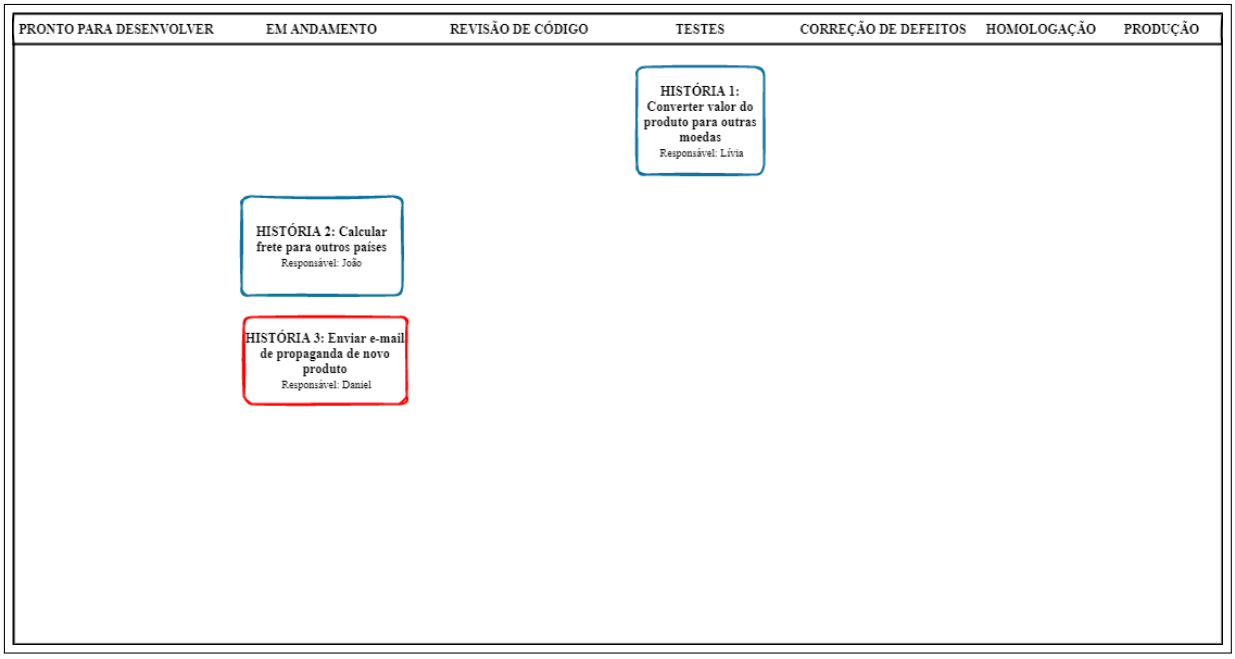



Fonte: A autora, 2021.

### 5.8.5 DIA 8 - ATD4, ATD1 e ATTE3:

- Time Scrum: Uma nova versão é gerada e a história 1 (Converter valor do produto para outras moedas) é liberada para teste, história 2 (calcular frete para outros países) e 3 (enviar *e-mail* de propaganda de novo produto) continuam em andamento, conforme exibido no quadro da figura 41.
- Time Kanban: Andreza libera a melhoria 1 (aumentar quantidade de itens para venda) para Olívia iniciar a homologação e Carolina continua no desenvolvimento da melhoria 2 (padronizar o comportamento do botão 'Salvar'), conforme exibido na figura 42.

Figura 41 - Quadro Time Scrum - Dia 8



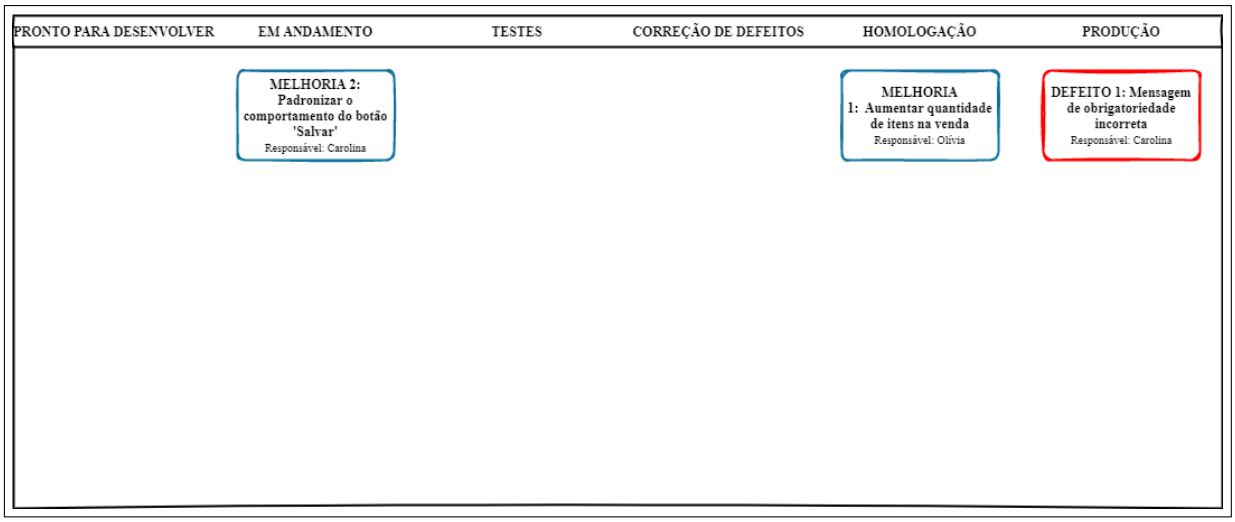

Fonte: A autora, 2021.

### 5.8.6 DIA 11: Reunião com cliente e continuação *Sprint* 1

ATR1, ATR2, ATR3, ATD1 e ATTE3

- Documentação | *Sprint* 2: Paula (PO) agenda uma reunião com Olívia para detalhar as necessidades, realizar refinamento das histórias e priorizar as tarefas para próxima *sprint*.

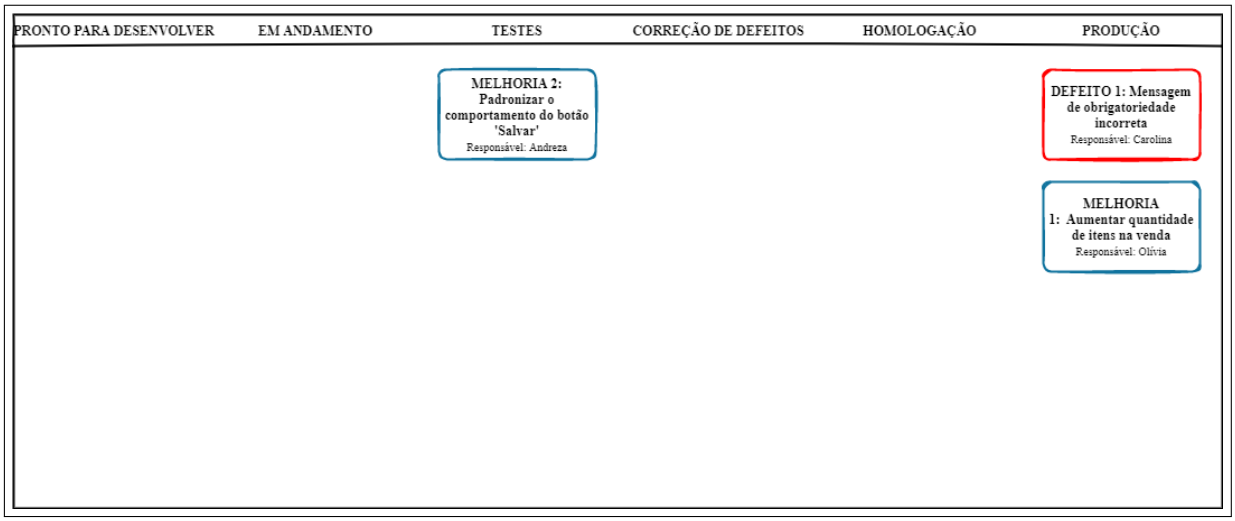
Figura 42 - Quadro Time Kanban - Dia 8




Fonte: A autora, 2021.

- Time Scrum: O quadro da figura 41 exibe o andamento do time Scrum, as histórias 2 e 3 continuam em desenvolvimento e a história 1 está em testes.
- Time Kanban: Carolina libera a melhoria 2 (padronizar o comportamento do botão 'Salvar') para Andreza testar e Olívia aprova a subida em produção da melhoria 1 (aumentar quantidade de itens para venda), conforme exibido na figura 43. Como Carolina terminou o desenvolvimento de todas as melhorias e defeitos do quadro de Kanban, ela se disponibiliza para ajudar Daniel com o desenvolvimento da história 3 (enviar *e-mail* de propaganda de novo produto) do time Scrum.

Figura 43 - Quadro Time Kanban - Dia 11

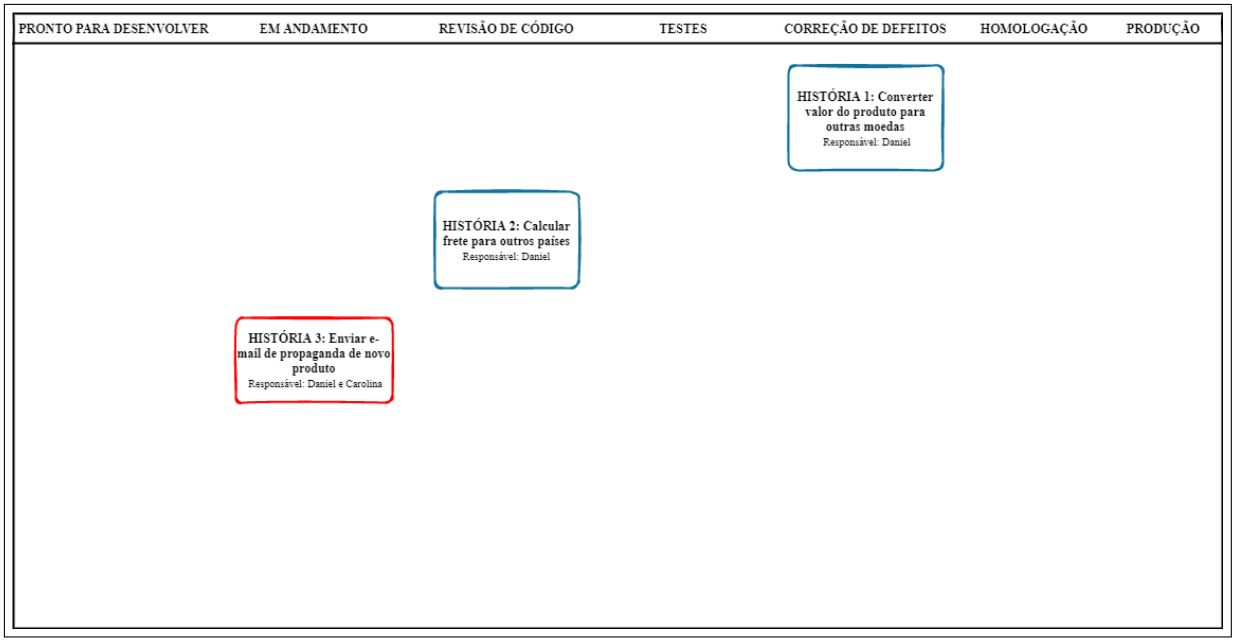



Fonte: A autora, 2021.

### 5.8.7 DIA 15 - ATTE4, ATD6, ATD2, ATD1:

- Time Scrum: Andreza finaliza os testes da história 1 (converter valor do produto para outras moedas) e abre defeitos para o desenvolvedor Daniel corrigir. A história 2 (calcular frete para outros países) foi finalizada pelo João e está em fase de revisão de código, com Daniel como responsável. Enquanto Daniel corrige defeitos, Carolina continua no desenvolvimento da história 3 (Enviar *e-mail* de propaganda de novo produto). O quadro atualizado do dia 15 está exibido na figura 44.
- Time Kanban: Andreza finaliza os testes da melhoria 2 (padronizar o comportamento do botão 'Salvar') e libera para homologação da Olívia, quadro atualizado na figura 45.

Figura 44 - Quadro Time Scrum - Dia 15

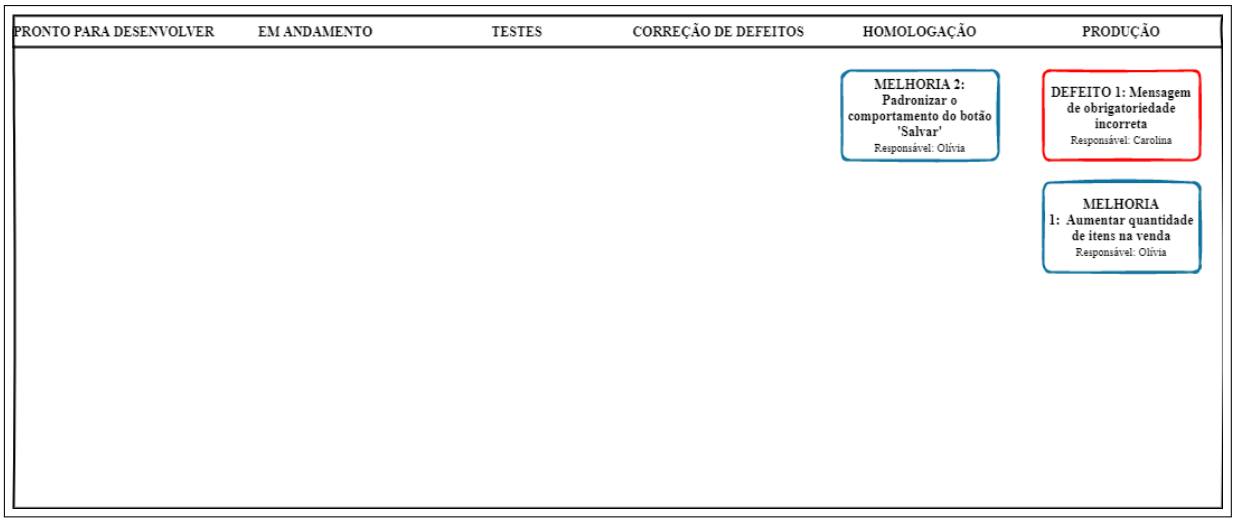



Fonte: A autora, 2021.

### 5.8.8 DIA 16 - ATTE3 e ATD1:

- Time Scrum: A correção de defeitos da história 1 (converter valor do produto para outras moedas) é finalizada e a história é movida para testes, para Lívia retestar. Como todos defeitos e melhorias finalizaram, Andreza ajuda o time de Scrum e começa os testes da história 2 (calcular frete para outros países). Os desenvolvedores continuam no desenvolvimento da história 3 (enviar *e-mail* de propaganda de novo

Figura 45 - Quadro Time Kanban - Dia 15




Fonte: A autora, 2021.

produto), João está responsável pela tarefa 3 (implementar mensagem de sucesso), Daniel pela tarefa 4 (implementar *e-mail* de novo produto) e Carolina está desenvolvendo a tarefa 5 (criar serviço de envio de *e-mail*). O quadro atualizado do dia 16 está exibido na figura 46.

- Time Kanban: A melhoria 2 (padronizar o comportamento do botão 'Salvar') foi liberada para subida em produção. Deixando o quadro Kanban sem novos defeitos e melhorias.

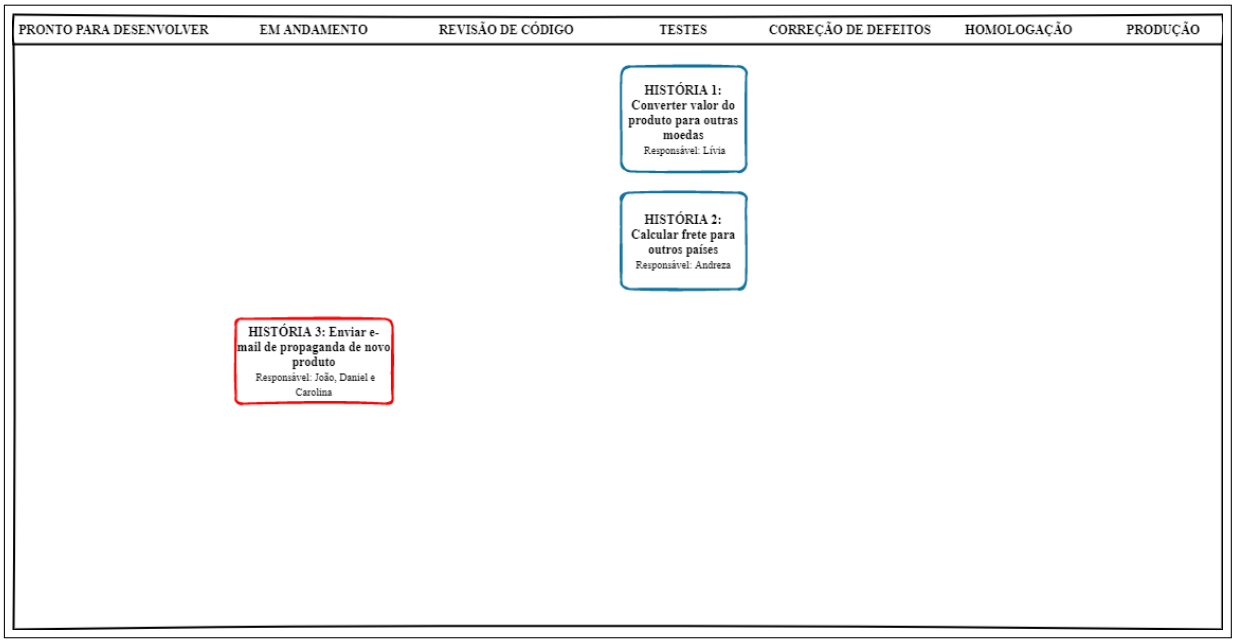
### 5.8.9 DIA 18: Detalhamento das histórias - ATR4, ATTE5, ATTE1 e ATD2

- Documentação | *Sprint* 2: Paula (PO) termina o detalhamento das histórias da *Sprint* 2 e envia para Olívia revisar se existe a necessidade de adicionar mais algum item.

- Time Scrum: Lívia terminou os retestes da história 1 (converter valor do produto para outras moedas) no dia 17 e finalizou a tarefa 8 (elaboração casos de teste funcionais) da história 3. Andreza continua os testes da história 2 (calcular frete para outros países). João inicia a revisão de código da história 3 (Enviar *e-mail* de propaganda de novo produto). O quadro atualizado do dia 18 está exibido na figura 47.

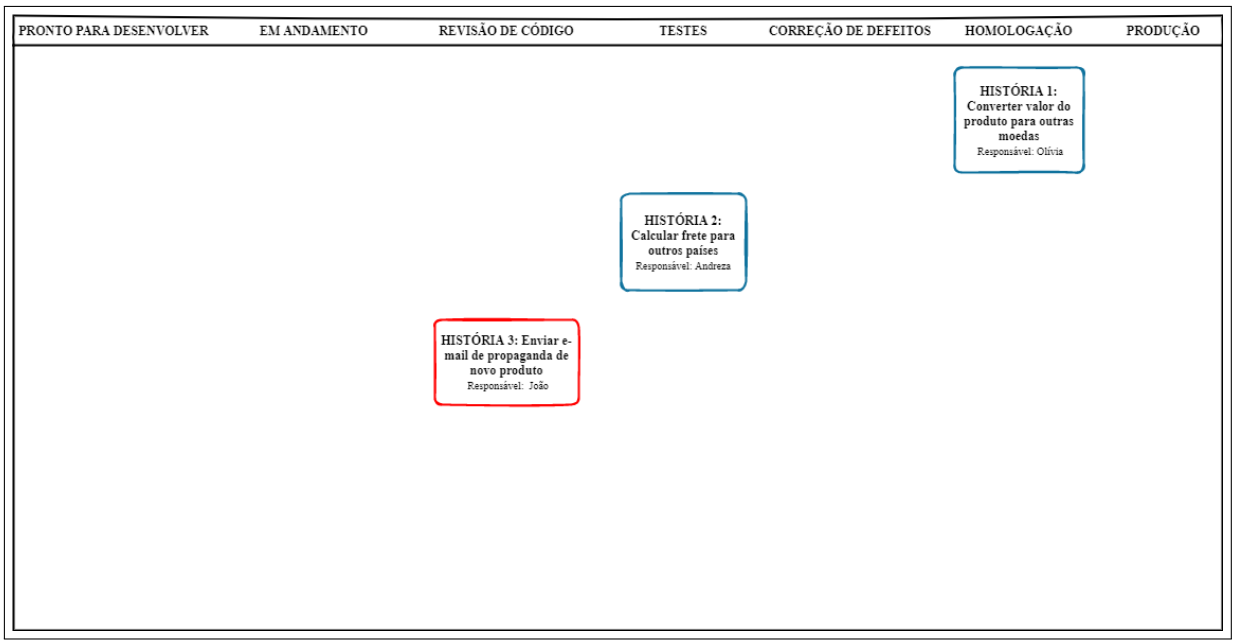
- Time Kanban: Sem novos defeitos ou melhorias.

Figura 46 - Quadro Time Scrum - Dia 16




Fonte: A autora, 2021.

Figura 47 - Quadro Time Scrum - Dia 18

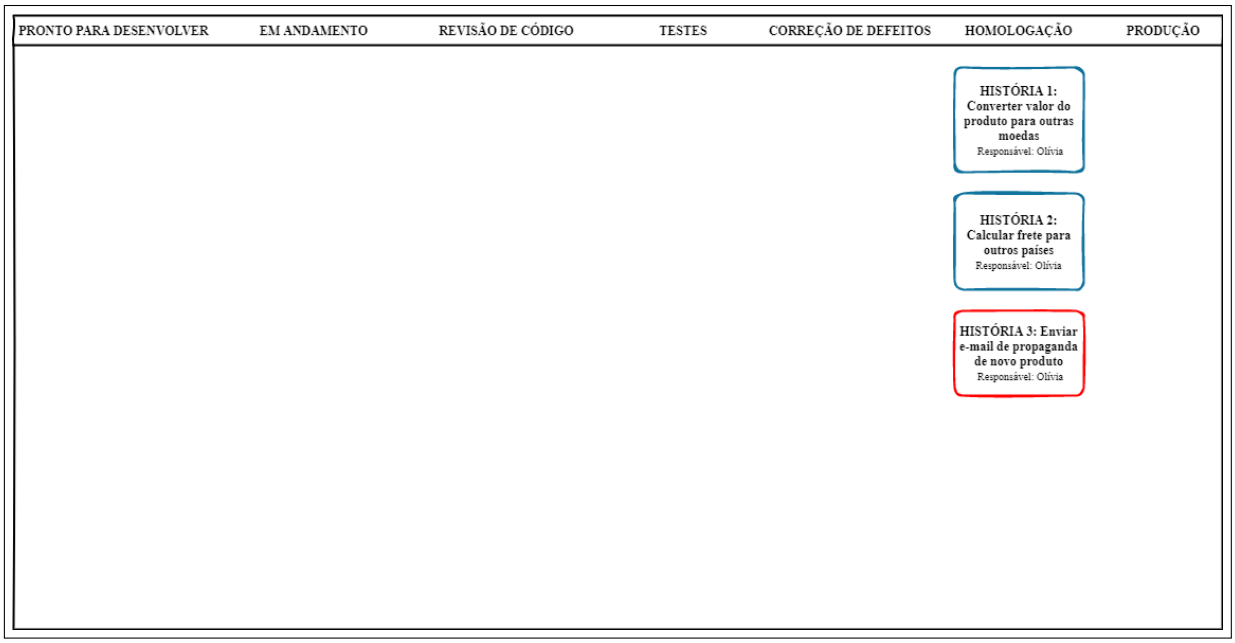



Fonte: A autora, 2021.

### 5.8.10 DIA 21 - ATTE5, ATTE3 e ATT2:

- Time Scrum: A história 2 (calcular frete para outros países) é liberada para homologação e a história 3 (enviar *e-mail* de propaganda de novo produto) está em fase de testes pela Lívia e Andreza. Lívia e Andreza finalizaram os testes da história 3 antes do horário de almoço e como não foram encontrados defeitos dessa história para corrigir, a história 3 também foi entregue durante a reunião de *review*. O quadro atualizado do dia 21 está exibido na figura 48.
- Time Kanban: Sem novos defeitos ou melhorias.
- Reunião de *Review*: Todas as histórias foram apresentadas para a Olívia durante a reunião e o time ficou em homologação junto com ela, todas as histórias foram aprovadas. Olívia solicitou a criação de duas melhorias referentes a história 3 que foram cadastradas no quadro de Kanban, representadas na figura 49.

Figura 48 - Quadro Time Scrum - Dia 21

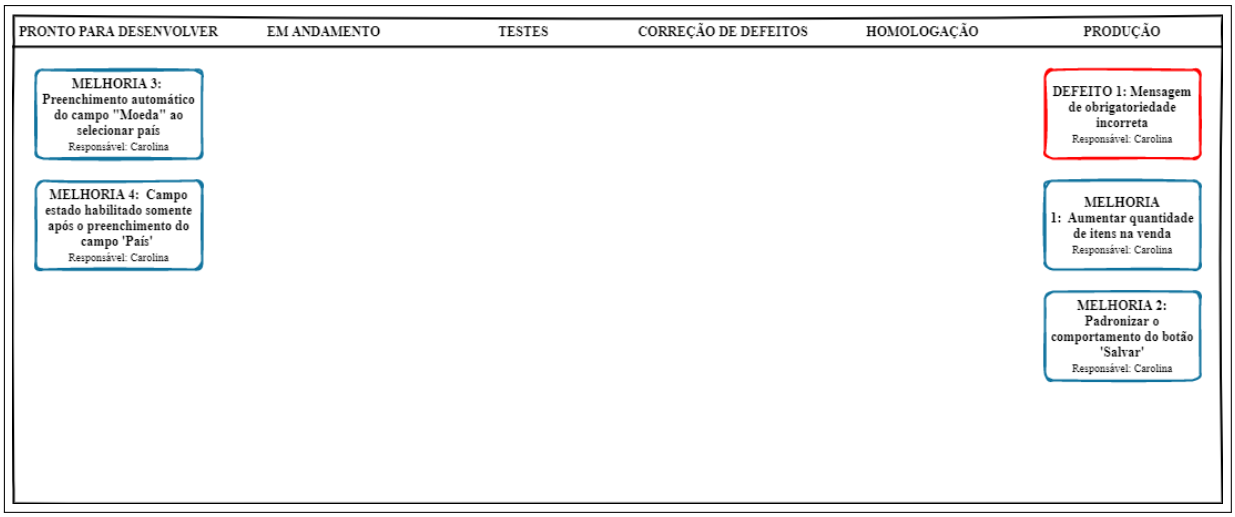



Fonte: A autora, 2021.

### 5.8.11 DIA 22: Reunião de retrospectiva ATT3 e planning ATT1 - *Sprint* 2

- Retrospectiva | Time Scrum: Bruna (*Scrum Master*) preparou um quadro virtual com 3 espaços para o time escrever anonimamente o que foi de positivo durante

Figura 49 - Quadro Time Kanban - Dia 21




Fonte: A autora, 2021.

a *sprint*, o que foi de negativo e sugestões de melhorias para as próximas *sprints*. O quadro gerado durante a reunião está exibido na figura 50. Os pontos positivos estão relacionados com a ajuda que a desenvolvedora Carolina e testadora Andreza puderam realizar no final da *sprint*, além disso, as histórias 2 e 3 tiveram poucos ou nenhum defeito encontrado e a Olívia elogiou a primeira entrega do time durante a reunião de *review*. Esses foram os 4 pontos que o time escreveu como pontos positivos. Como ponto negativo, o time destacou muitos defeitos encontrados na história 1 e melhorias que Olívia identificou que poderiam ter sido identificadas pelo time. Como sugestão, realizar conversas com a Paula (PO) quando tiver sugestões de melhorias durante o desenvolvimento do código das funcionalidades.

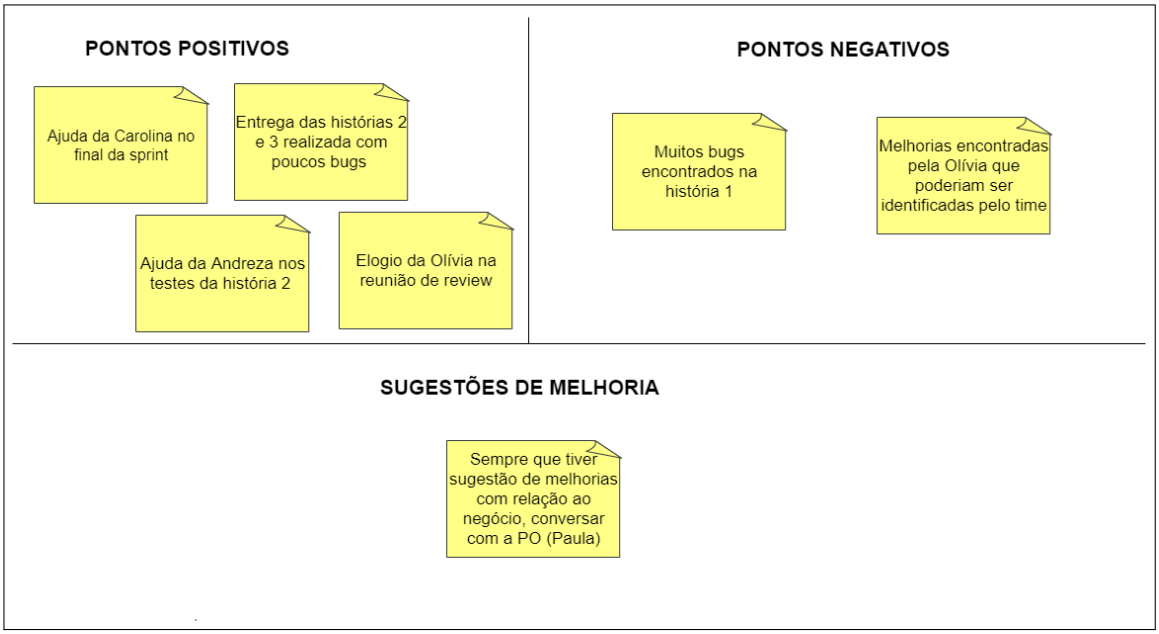
- Time Kanban: Carolina iniciou o desenvolvimento da melhoria 3 (Preenchimento automático do campo “Moeda” ao selecionar país).

A reunião de planejamento da *sprint* 2 é iniciada logo após a reunião de retrospectiva.

### 5.8.12 Observações

Como se trata de uma POC, as atividades de reunião com cliente, refinamento do *backlog*, priorização de tarefas e detalhamento de tarefas da *sprint* 2 que são realizados pelo PO do time não foram detalhadas pois já foram detalhadas antes da execução da *sprint* 1. Foi apenas sinalizado durante o texto quando essas atividades foram realizadas.

Figura 50 - Retrospectiva - *Sprint* 1




Fonte: A autora, 2021.

### 5.8.13 Conclusão

Uma deficiência do *framework* que foi identificada durante a POC está relacionada ao time Kanban. O *framework* não destaca quais práticas ágeis o time Kanban utiliza, apenas informa que o time é responsável por tarefas incidentes. Para o time Scrum, todo o processo definido no *framework* foi bem utilizado durante a POC.

# CONSIDERAÇÕES FINAIS

O desenvolvimento deste trabalho possibilitou o conhecimento de diversas práticas ágeis que estão sendo utilizados em artigos acadêmicos e especialistas na indústria. Um *framework* de manutenção ágil foi desenvolvido com as práticas ágeis mais utilizadas identificadas durante todo o processo de pesquisa com o objetivo de melhorar a fase de manutenção de *software*.

Primeiramente, foi realizado um Mapeamento Sistemático da Literatura que forneceu quais abordagens, técnicas, práticas, métodos e metodologias ágeis estão sendo utilizados em manutenção evolutiva de *software*. Foi possível compreender como a agilidade está sendo utilizada na fase de manutenção e quais as vantagens e desafios do uso dela. A partir do resultado da análise dos dados deste mapeamento, foi desenvolvido o modelo 1 de *framework* de manutenção ágil.

Além do mapeamento sistemático da literatura, foi realizado um *survey* com especialistas que atuam na área de manutenção de *software* para verificar quais as principais práticas, vantagens e desafios da manutenção na indústria. O modelo 2 do *framework* foi elaborado a partir da análise dos resultados coletados do *survey*.

O desenvolvimento do *framework* final de manutenção ágil apresentado no Capítulo 4 foi norteado pela comparação dos modelos 1 e 2 elaborados a partir do mapeamento sistemático e do *survey* com especialistas. O *framework* desenvolvido é baseado em equipes pequenas (5 a 10 integrantes), auto gerenciáveis, com pequenas entregas (2 a 4 semanas) e times separados para tarefas evolutivas e tarefas incidentes. As reuniões diárias, reunião de planejamento, reunião de revisão e reunião de retrospectiva são as cerimônias recomendadas para o time. Além disso, é necessário ter um acompanhamento do andamento das tarefas, *feedback* constante com o time e coleta de métricas das funcionalidades após entrar em ambiente de produção.

As atividades do *framework* estão divididas em ATR, ATD, ATTE e ATT conforme exibido na Figura 31. Cada sigla representa um tipo de atividade, sendo ATR identificadas por atividade de requisitos, ATD por atividade de desenvolvimento, ATTE por atividade de teste e ATT por atividade do time.

Com a realização de uma POC com um cenário hipotético para apresentar como o *framework* é utilizado, foi possível avaliar que as práticas ágeis do *framework* são bem aplicadas na fase de manutenção evolutiva, inclusive para times que também possuem tarefas corretivas e incidentes.

### Contribuições

Como principais contribuições deste trabalho podem-se citar:

- Modelo de *framework* a partir de resultados de um Mapeamento Sistemático da Literatura;
- Modelo de *framework* a partir de resultados de uma pesquisa qualitativa através de *survey* com especialistas de manutenção de *software* da indústria;
- *Framework* de manutenção ágil com as melhores práticas utilizadas nos artigos e no mercado;
- Realização de uma POC simulando a utilização do *framework* em uma manutenção de *software*.

### Trabalhos Futuros

Como trabalhos futuros, pretende-se identificar quais práticas ágeis são utilizadas em organizações e projetos que utilizam o Modelo Integrado de Maturidade em Capacitação (CMMI) como forma de melhoria do *framework*.

Além disso, com o objetivo de incrementar o *framework*, pode ser realizado um estudo das melhores métricas de equipe utilizadas em manutenção de *software* e incorporá-las no *framework*. Deste modo, além de definir as características do time, cerimônias e atividades, o *framework* pode sugerir quais métricas devem ser utilizadas durante a fase de manutenção evolutiva de *software*.

# REFERÊNCIAS


ABRAHAMSSON, Pekka; OZA, Nilay; SIPONEN, Mikko T. Agile software development methods: a comparative review. In: *Agile software development*. [S.l.]: Springer, 2010. p. 31–59.

ABRAHAMSSON, Pekka et al. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

AHMAD, Muhammad Ovais et al. Transition of software maintenance teams from scrum to kanban. In: IEEE. *2016 49th Hawaii International Conference on System Sciences (HICSS)*. [S.l.], 2016. p. 5427–5436.

AITKEN, Ashley; ILANGO, Vishnu. A comparative analysis of traditional software engineering and agile software development. In: IEEE. *2013 46th Hawaii International Conference on System Sciences*. [S.l.], 2013. p. 4751–4760.

ALVES, Raphaelly Antunes; MARTINS, Romário Carlos; PAULISTA, Paulo Henrique. Operação e manutenção de software: Uma abordagem teórica. *Revista Univap*, v. 22, n. 40, p. 766, 2017.

ARRUDA, Ludmila Varela. Desenvolvimento ágil de software: uma análise sintética a partir da metodologia kanban. In: *VII CONNEPI-Congresso Norte Nordeste de Pesquisa e Inovação*. [S.l.: s.n.], 2012.

AUDY, Jorge. *Scrum 360: Um guia completo e prático de agilidade*. [S.l.]: Editora Casa do Código, 2015.

BALDIN, Nelma; MUNHOZ, Elzira M Bagatin. *Snowball (bola de neve): uma técnica metodológica para pesquisa em educação ambiental comunitária. 2011*. 2020.

BARDIN, Laurence. *L'analyse de contenu*. [S.l.]: Presses universitaires de France Paris, 1977. v. 69.

BECK, Kent. *Extreme programming explained: embrace change*. [S.l.]: addison-wesley professional, 2000.

BLAIR, Erik. A reflexive exploration of two qualitative data coding techniques. *Journal of Methods and Measurement in the Social Sciences*, v. 6, n. 1, p. 14–29, 2015.

BOGOJEVIĆ, Petar et al. Comparative analysis of agile methods for managing software projects. *European Project Management Journal*, Udruženje za upravljanje projektima-IPMA Srbija, v. 7, n. 1, p. 58–74, 2017.

CARDANO, Mario. Manual de pesquisa qualitativa. *A contribuição da teoria da argumentação. Tradução: Elisabeth da Rosa Conill. Petrópolis, Rio de Janeiro: Vozes*, 2017.

CARROLL, Linda J; ROTHE, J Peter. Levels of reconstruction as complementarity in mixed methods research: A social theory-based conceptual framework for integrating qualitative and quantitative research. *International journal of environmental research and public health*, Molecular Diversity Preservation International, v. 7, n. 9, p. 3478–3488, 2010.

CHARMAZ, Kathy. *Constructing grounded theory: A practical guide through qualitative analysis*. [S.l.]: sage, 2006.

CHOUDHARI, Jitender; SUMAN, Ugrasen. Story points based effort estimation model for software maintenance. *Procedia Technology*, Elsevier, v. 4, p. 761–765, 2012.

_____. Extended iterative maintenance life cycle using extreme programming. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 39, n. 1, p. 1–12, 2014.

COSTA, Daiane Morandi da. Customização de processo de desenvolvimento de software baseada na modelagem ágil. 2016.

CRESWELL, John W. *30 Essential Skills for the Qualitative Researcher*. [S.l.]: Department of Family Medicine, University of Michigan, 2015.

DUKA, Denis. Agile experiences in software development. In: IEEE. *2012 Proceedings of the 35th International Convention MIPRO*. [S.l.], 2012. p. 692–697.

ELLIOTT, Victoria. Thinking about the coding process in qualitative data analysis. *The Qualitative Report*, The Qualitative Report, v. 23, n. 11, p. 2850–2861, 2018.

FILHO, Wilson de Pádua Paula. *Engenharia de software*. [S.l.]: LTC, 2003. v. 2.

FOWLER, Martin. The new methodology. *Wuhan University Journal of Natural Sciences*, Springer, v. 6, n. 1-2, p. 12–24, 2001.

GHOSH, Gopal K. Challenges in distributed scrum. In: IEEE. *2012 IEEE Seventh International Conference on Global Software Engineering*. [S.l.], 2012. p. 200–200.

GUIDE, SCRUM. Scrum. *Org, disponível em< http://www. scrum. org/>. Acessado em*, v. 13, 2020.

GUIMARÃES, Júlio Henrique dos Nogueira et al. *Método para manutenção de sistema de software utilizando técnicas arquiteturais.* Tese (Doutorado) — Universidade de São Paulo, 2008.

HEEAGER, Lise Tordrup; ROSE, Jeremy. Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, Springer, v. 20, n. 6, p. 1762–1784, 2015.

HUO, Ming et al. Software quality and agile methods. In: IEEE. *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.* [S.l.], 2004. p. 520–525.

JAVANMARD, Mahdi; ALIAN, Maryam. Comparison between agile and traditional software development methodologies. *Cumhuriyet Üniversitesi Fen-Edebiyat Fakültesi Fen Bilimleri Dergisi*, v. 36, n. 3, p. 1386–1394, 2015.

JR, Gibeon Soares de Aquino; DANTAS, André Medeiros. Uma abordagem ágil aplicada a projetos de manutenção intensa. In: SBC. *Anais do XV Simpósio Brasileiro de Sistemas de Informação*. [S.l.], 2019. p. 351–358.

JUNIOR, Gibeon Soares de Aquino; DANTAS, André Medeiros. An agile approach applied to intense maintenance projects. In: ACM. *Proceedings of the XV Brazilian Symposium on Information Systems*. [S.l.], 2019. p. 45.

K21. 2020. Disponível em: <https://k21.global/blog/product-backlog-epico-historia-tarefas>. Acesso em: Agosto 2021.

KAUR, Kamaljeet; JAJOO, Anuj et al. Applying agile methodologies in industry projects: Benefits and challenges. In: IEEE. *2015 International Conference on Computing Communication Control and Automation*. [S.l.], 2015. p. 832–836.

KENCHICOSKI, André Luiz Antunes; CRUZ, Felipe Lopes da. Gestão de projetos e inovação na administração pública brasileira. *Caderno Virtual*, v. 3, n. 48, 2020.

KILPALA, Minna; KÄRKKÄINEN, Tommi. Distributed scrum when turning into maintenance: A single case study. In: *Second International Conference on Computer Science and Information Technology (COSIT 2015)[Internet].[cited 2016 Feb 3]*. [S.l.: s.n.], 2015. p. 55–67.

KITCHENHAM, Barbara A; BUDGEN, David; BRERETON, O Pearl. Using mapping studies as the basis for further research–a participant-observer case study. *Information and Software Technology*, Elsevier, v. 53, n. 6, p. 638–651, 2011.

KLOTINS, Eriks; UNTERKALMSTEINER, Michael; GORSCHEK, Tony. Software engineering antipatterns in start-ups. *IEEE Software*, IEEE, v. 36, n. 2, p. 118–126, 2018.

KNIPPERS, Daniël. Agile software sevelopment and maintainability. In: *15th Twente Student Conf.* [S.l.: s.n.], 2011.

KUHRMANN, Marco et al. *Managing software process evolution: traditional, agile and beyond–how to handle process change*. [S.l.]: Springer, 2016.

LIMA, ALINE FRANCIELLE DOS ANJOS. *Processo de desenvolvimento de requisitos do CMMI-DEV e MR-MPS-SW para projetos de manutenção evolutiva: um estudo de caso de implementação*. Tese (Doutorado) — Dissertação (Mestrado)-Universidade Federal do Pará, Instituto de Ciências . . . , 2017.

LOUS, Pernille et al. From scrum to agile: a journey to tackle the challenges of distributed development in an agile team. In: *Proceedings of the 2018 International Conference on Software and System Process*. [S.l.: s.n.], 2018. p. 11–20.

MALIMPENSA, Gabriel Gioannini. Uma abordagem para a priorização de casos de teste de regressão baseada em rastreabilidade. Universidade Federal de São Carlos, 2018.

MANIFESTO, Agile. *Manifesto for Agile Software Development. Retrieved March 12, 2014*. 2004.

MANUJA, Manoj et al. Moving agile based projects on cloud. In: IEEE. *2014 IEEE International Advance Computing Conference (IACC)*. [S.l.], 2014. p. 1392–1397.

MARTINS, Fernanda Lanção Morais. Métodos ágeis na concepção e manutenção de sistemas ligados a infraestrutura. 2013.

MORAES, Roque; GALIAZZI, Maria do Carmo. Análise textual discursiva: processo reconstrutivo de múltiplas faces. *Ciência & Educação (Bauru)*, SciELO Brasil, v. 12, n. 1, 2006.

MOURA, Reinaldo A. Kanban: a simplicidade do controle da produção 5. ed. *São Paulo: IMAM*, 1999.

NAZ, Riffat; KHAN, MNA; AAMIR, Muhammad. Scrum-based methodology for product maintenance and support. *International Journal of Engineering and Manufacturing (IJEM)*, v. 6, n. 1, p. 10, 2016.

OMANOVIC, Samir; BUZA, Emir. Importance of stable velocity in agile maintenance. In: IEEE. *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*. [S.l.], 2013. p. 1–8.

PINO, Francisco J et al. A software maintenance methodology for small organizations: Agile_mantema. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 24, n. 8, p. 851–876, 2012.

PRIKLADNICKI, Rafael; WILLI, Renato; MILANI, Fabiano. *Métodos ágeis para desenvolvimento de software*. [S.l.]: Bookman Editora, 2014.

QURESHI, M Rizwan Jameel. Agile software development methodology for medium and large projects. *IET software*, IET, v. 6, n. 4, p. 358–363, 2012.

RAMOS, André LBM et al. Práticas ágeis aplicadas a um processo de manutenção de software: Um relato de experiência. In: SBC. *Anais do XII Simpósio Brasileiro de Qualidade de Software*. [S.l.], 2013. p. 342–349.

RAO, Kuda Nageswara; NAIDU, G Kavita; CHAKKA, Praneeth. A study of the agile software development methods, applicability and implications in industry. *International Journal of Software Engineering and its applications*, v. 5, n. 2, p. 35–45, 2011.

REHMAN, Fateh ur et al. Scrum software maintenance model: Efficient software maintenance in agile methodology. In: IEEE. *2018 21st Saudi Computer Society National Computer Conference (NCC)*. [S.l.], 2018. p. 1–5.

REN, Jimmy et al. Optimal refactoring policy for agile information systems maintenance: A control theoretic approach. 2011.

RICO, DF. *Agile Methods and Software Maintenance*. 2008.

SANTOS, Jads Victor Paiva dos. Uso do kanban em um processo de gestão de demandas de manutenção de software por terceiros para um órgão público federal brasileiro. 2015.

SANTOS, Marlu da Silva. Análise classificatória de sistemas colaborativos aplicada ao gerenciamento de projetos ágeis: uma abordagem scrum. *Anais da Escola Regional de Informática da Sociedade Brasileira de Computação (SBC)–Regional de Mato Grosso*, v. 7, 2016.

SANTOS, Saulo Eduardo Galilleo Souza dos; OLIVEIRA, Adicinéia Aparecida de. Modelos de processo de evolução e manutenção de software no contexto dos métodos ágeis. 2012.

SCHWABER, Ken. Scrum development process. In: *Business object design and implementation*. [S.l.]: Springer, 1997. p. 117–134.

SCHWABER, Ken; BEEDLE, Mike. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1.

SEAMAN, Carolyn B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, IEEE, v. 25, n. 4, p. 557–572, 1999.

SILVA, Alessandro et al. Método de pesquisa survey – estudo do método e aplicações na engenharia de produção. *XIV WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA DO CENTRO PAULA SOUZA*, 2019.

SILVA, Andressa Hennig; FOSSÁ, Maria Ivete Trevisan. Análise de conteúdo: exemplo de aplicação da técnica para análise de dados qualitativos. *Qualitas Revista Eletrônica*, v. 16, n. 1, 2015.

SILVA, Diogo Vinícius de S; SANTOS, F Alan de O; NETO, Pedro Santos. Os benefícios do uso de kanban na gerência de projetos de manutenção de software. In: SBC. *Anais do VIII Simpósio Brasileiro de Sistemas de Informação*. [S.l.], 2012. p. 715–725.

SILVA, JESSICA BELEM DA. Metodo kanban como ferramenta de controle de gestão. 2018.

SILVA, Rosa Carolina Sampaio; STRAUSS, Edilberto; MELLO, Flávio Luis de. Uso do mindset ágil aplicado na garantia da qualidade do processo e do produto de software. 2017.

SOARES, Michel dos Santos. Comparação entre metodologias ágeis e tradicionais para o desenvolvimento de software. *INFOCOMP Journal of Computer Science*, v. 3, n. 2, p. 8–13, 2004.

SOMMERVILLE, Ian. *Software Engineering, International Edition*. [S.l.]: Boston, Published by Addison-Wesley, 2010.

SOUZA, Ellen; GUSMÃO, Cristine; ROCHA, Humberto. rbtprocess-proposta de modelo de processo de teste de software baseado em riscos. *III EBTS–Encontro Brasileiro de Teste de Software*, 2008.

SOUZA, I.; SCHOTS, M. *Characterizing the Usage of Visualizations in Software Development: an Interview-Based Study*. 2018. Monografia (Bacharel em Ciência da Computação), UERJ (Universidade do Estado do Rio de Janeiro), RJ, Brazil.

STAKE, Robert E. *Pesquisa qualitativa: estudando como as coisas funcionam*. [S.l.]: Penso Editora, 2016.

STERN, Phyllis Noerager; PORR, Caroline Jane. *Essentials of accessible grounded theory*. [S.l.]: Routledge, 2017.

STOICA, Marian; MIRCEA, Marinela; GHILIC-MICU, Bogdan. Software development: Agile vs. traditional. *Informatica Economica*, v. 17, n. 4, 2013.

STRAUSS, Anselm; CORBIN, Juliet. Basics of qualitative research: Procedures and techniques for developing grounded theory. Thousand Oaks, CA: Sage, 1998.

SURESHCHANDRA, Kalpana; SHRINIVASAVADHANI, Jagadish. Moving from waterfall to agile. In: IEEE. *Agile 2008 conference*. [S.l.], 2008. p. 97–101.

TARWANI, Sandhya; CHUG, Anuradha. Agile methodologies in software maintenance: A systematic review. *Informatica*, v. 40, n. 4, 2016.

TEAM, MARVEL-SE Research. *Coding: Ferramenta de Análise Qualitativa*. 2018. Disponível em: <https://gitlab.com/marvel-se/qualitative-analysis>.

TELES, Vinicius Manhães. Um estudo de caso da adoção das práticas e valores do extreme programming. *UFRJ–Universidade Federal do Rio de Janeiro*, 2005.

TELES, Vinícius Manhães. *Extreme Programming-2ª Edição: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*. [S.l.]: Novatec Editora, 2014.

THANGASAMY, S. Lessons learned in transforming from traditional to agile development. Citeseer, 2012.

TOLFO, Cristiano et al. Agile methods and organizational culture: Reflections about cultural levels. *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley Online Library, v. 23, n. 6, p. 423–441, 2011.

TOLLEY, Elizabeth E et al. *Qualitative methods in public health: a field guide for applied research*. [S.l.]: John Wiley & Sons, 2016.

TRIMBLE, Jay; WEBSTER, Christopher. From traditional, to lean, to agile development: Finding the optimal software engineering cycle. In: IEEE. *2013 46th Hawaii International Conference on System Sciences*. [S.l.], 2013. p. 4826–4833.

VALLON, Raoul et al. Systematic literature review on agile practices in global software development. *Information and Software Technology*, Elsevier, v. 96, p. 161–180, 2018.

VOSGERAU, Dilmeire SantAnna Ramos; POCRIFKA, Dagmar Heil; SIMONIAN, Michele. Etapas da análise de conteúdo complementadas por ciclos de codificação: possibilidades a partir do uso de software de análise qualitativa de dados. *CIAIQ2016*, v. 1, 2016.

WAZLAWICK, Raul. *Engenharia de software: conceitos e práticas*. [S.l.]: Elsevier Editora Ltda., 2019.

_____. _____. [S.l.]: Elsevier Editora Ltda., 2019.

WWW.ISO.ORG. *Information technology—Software life cycle processes, 2002 - ISO/IEC 12207*. Acesso em: Abril 2011.

ZANOTTI, Christopher C; KAYLOR, Andrew J; DAVIDSEN, Kristine L. Multi-discipline agile development and reliability and maintainability. In: IEEE. *2017 Annual Reliability and Maintainability Symposium (RAMS)*. [S.l.], 2017. p. 1–3.

# GLOSSÁRIO

| | |
|---|---|
| Agilistas | Profissionais que trabalham com agilidade |
| *Backlog* | É uma lista que contém breves descrições de todas as funcionalidades desejadas para um produto |
| *Brainstorming* | É uma tempestade de ideias |
| *Combobox* | É um elemento de interface gráfica |
| *Crystal* | Uma família de metodologias Agile, que inclui diversas variantes, ex: Crystal Clear. |
| *Daily* | Um evento de 15 minutos para os desenvolvedores do time |
| *Discovery* | É a primeira fase no ciclo de um produto digital |
| *Feedback* | Dar resposta a uma atitude ou comportamento |
| *Front-end* | É a parte responsável pela interação com o usuário, ou seja, todo conteúdo que fica visível, como por exemplo: o layout da página, os efeitos visuais e até mesmo a validação de formulários |
| Git | É um sistema de controle de versão de arquivos |
| *Internet Banking* | Banco online onde o usuário consegue fazer diversas transações |
| Kanban | Método de gestão de mudanças que auxilia a visualizar o trabalho em andamento através de um quadro |
| *Lean* | Uma filosofia de gestão inspirada em práticas e resultados do Sistema Toyota |
| *Mindset* | A maneira como as pessoas do time agem frente aos desafios que surgem. |
| *Parprogramming* | Um método de programação no qual duas pessoas trabalham juntas em um único programa |
| *Radio button* | Um elemento de interface gráfica com dois estados: selecionado e não-selecionado |
| *Release* | É uma versão de um produto |
| Sashimi | É uma variação do modelo de desenvolvimento de software Cascata (FILHO, 2003) |
| Scrum | Framework para desenvolver e manter produtos complexos |
| *Sprint* | São eventos de duração fixa de um mês ou menos para criar consistência. Uma nova Sprint começa imediatamente após a conclusão da Sprint anterior. |
| *Stakeholders* | Qualquer indivíduo ou organização que, de alguma forma, é impactado pelas ações de uma determinada empresa |
| *Snowballing* | utiliza uma cadeia de referências para se obter uma amostra representativa |
| Template | É um documento de conteúdo, com apenas a apresentação visual e instruções sobre onde e qual tipo de conteúdo deve entrar |
| Time | Equipe ágil |
| *Wireframes* | Um protótipo ou uma versão bastante primitiva do visual de um projeto |

| | |
|---|---|
| Wiki | É uma ferramenta de edição colaborativa, uma ferramenta rápida para criação e edição de páginas online |

# **APÊNDICE** – Questionário

## A.1 **Termo de consentimento livre e esclarecido**

Você está sendo convidado (a) a participar, como voluntário (a), da pesquisa intitulada Modelo de manutenção de software evolutiva utilizando práticas ágeis conduzida por Isabelle Barbalho Oliveira de Souza, Johnny Cardoso Marques e Vera Maria Benjamin Werneck. Este estudo tem por objetivo explorar práticas ágeis utilizadas durante a manutenção de software voltado para projetos da área acadêmica e da indústria e a partir dos dados coletados do mapeamento sistemático da literatura e respostas de questionários, elaborar um modelo de manutenção evolutiva utilizando práticas ágeis. Você foi selecionado (a) por ser profissional da área de informática que possui experiências com projetos de manutenção evolutiva de software. Sua participação não é obrigatória. A qualquer momento, você poderá desistir de participar e retirar seu consentimento, abandonando o preenchimento do questionário. Sua recusa, desistência, ou retirada de consentimento não acarretará prejuízo. A pesquisa oferece riscos mínimos para os participantes, envolvendo possíveis constrangimentos em opinar sobre a qualidade do desenvolvimento de software durante a fase de manutenção evolutiva. Todavia, como benefícios, esta pesquisa gerará um modelo que poderá abrir novas perspectivas para os profissionais de desenvolvimento de software. Sua participação não é remunerada e nem implicará em gastos para você. Caso se sinta prejudicado, você tem direito de ser indenizado por danos decorrentes da pesquisa, nos termos da lei e de ser ressarcido de despesas decorrentes da participação na pesquisa, de acordo com os itens VI e VII do artigo 9o, do Cap. III, da Resolução 510/2016. Os dados obtidos por meio desta pesquisa serão confidenciais e não serão divulgados em nível individual, visando assegurar o sigilo de sua participação. Neste caso, os pesquisadores se comprometem a tornar públicos nos meios acadêmicos e científicos os resultados obtidos de forma consolidada, sem qualquer identificação de indivíduos, ou instituições participantes. Caso você concorde em participar desta pesquisa, assinale ao final deste documento, que possui duas vias, sendo uma delas sua, e a outra, do pesquisador responsável/coordenador da pesquisa. Seguem os telefones e o endereço institucional do pesquisador responsável e do Comitê de Ética em Pesquisa – CEP, onde você poderá tirar suas dúvidas sobre o projeto e sua participação nele, agora ou a qualquer momento.

Contatos dos pesquisadores responsáveis: Isabelle Barbalho Oliveira de Souza (21) 99903-8253, e-mail: ibarbalho26@gmail.com, Johnny Cardoso Marques, e-mail: johnny@ita.br, Vera M. B. Werneck, e-mail: vera@ime.uerj.br. Caso você tenha dificuldade em entrar em contato com os pesquisadores responsáveis, comunique o fato à Comissão de Ética em Pesquisa da UERJ: Rua São Francisco Xavier, 524, sala 3018, bloco E, 3º andar, - Maracanã - Rio de Janeiro, RJ, e-mail: eticauerj.br - Telefone: (021) 2334-2180. O CEP

COEP é responsável por garantir a proteção dos participantes de pesquisa e funciona às segundas, quartas e sextas-feiras, de 10h às 12h e 14h às 16h. Declaro que entendi os objetivos, riscos e benefícios de minha participação na pesquisa, e que concordo em participar.

## B.2 **Perguntas**

1- Deseja receber o resultado da pesquisa por e-mail? ( ) Sim ( ) Não

2- E-mail:

3- Empresa/Instituição de ensino:

4- A(s) fase(s) anterior(es) à manutenção do projeto utilizou metodologia ágil?
( ) Sim ( ) Não

5- Qual metodologia tradicional era utilizada? (*pergunta 5 é exibida se participante responder 'Não' na pergunta 4.*)
( ) Cascata
( ) Espiral
( ) Prototipação
( ) Outra:

6- Quais práticas ágeis eram utilizadas? (*pergunta 6 é exibida se participante responder 'Sim' na pergunta 4.*)
( ) Pequenas entregas
( ) Feedback com o time
( ) Equipe Auto gerenciável
( ) Times Pequenos
( ) Colaboração do cliente
( ) Reuniões diárias (Daily Meeting)
( ) Reunião de Planejamento (Planning Meeting)
( ) Sprint Review
( ) Reunião de retrospectiva
( ) Melhoria contínua com lições aprendidas
( ) Reuniões semanais com estratégias, estimativas
( ) Monitoração contínua das atividades
( ) Kanban
( ) Priorização de tarefas

( ) Design simples
( ) Documentação reduzida
( ) Documentação no código
( ) Programação em pares
( ) Padrões de programação
( ) Revisão de código
( ) Controle de versão
( ) Refatoração de código
( ) Execução de testes
( ) Alta cobertura de testes (>80%)
( ) Testes de regressão
( ) Testes unitários
( ) Testes automatizados

7- Dê detalhes sobre a documentação das fases anteriores à fase de manutenção evolutiva. (Caso não tenha tido documentação, responder com "-")

8- O processo de especificação de requisitos/refinamento do *backlog* da fase de manutenção foi alterada?
( ) Sim ( ) Não

9- Dê detalhes sobre a construção e refinamento do *backlog* na fase de manutenção evolutiva. (*pergunta 9 é exibida se participante responder 'Sim' na pergunta 8.*)

10- Dê detalhes sobre a construção e refinamento do *backlog* na fase de manutenção evolutiva. (*pergunta 10 é exibida se participante responder 'Não' na pergunta 8.*)

11- O processo de codificação da fase de manutenção foi alterada?
( ) Sim ( ) Não

12- Dê detalhes sobre a codificação na fase de manutenção evolutiva. (*pergunta 12 é exibida se participante responder 'Sim' na pergunta 10.*)

13- Dê detalhes sobre a execução de testes da fase anterior à fase de manutenção evolutiva. (*pergunta 13 é exibida se participante responder 'Não' na pergunta 10.*)

14- O processo de teste foi alterado na fase de manutenção evolutiva?
( ) Sim ( ) Não

15- Dê detalhes sobre o processo de testes na fase de manutenção evolutiva. (*pergunta 15 é exibida se participante responder 'Sim' na pergunta 14.*)

**ANEXO** – Tabela de resultado do mapeamento sistemático da literatura

Tabela 15 - Resultado resumido do mapeamento sistemático da literatura

| **Autores** | **Práticas ágeis** | **Vantagens** | **Desafios** |
|---|---|---|---|
| (ABRAHAMSSON; OZA; SIPONEN, 2010) | Pequenas entregas; melhoria contínua; teste de integração; teste de aceitação; teste unitário | Flexibilidade à mudanças | Identificar quais melhores práticas |
| (RAO; NAIDU; CHAKKA, 2011) | Pequenas entregas; Times pequenos | Satisfação do cliente; Flexibilidade à mudanças; Comunicação com o cliente; Qualidade no *software* (mais testes, menos *bugs*) | Não encontrado |
| (TOLFO et al., 2011) | Refatoração; Programação em pares; Integração contínua; Reunião diária; Testes automatizados; Padrões de código; Melhoria contínua | Não encontrado | Não encontrado |
| (KNIPPERS, 2011) | Pequenas entregas; Documentação no código; Times pequenos | Flexibilidade à mudanças; Comunicação com o cliente | Não encontrado |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (REN et al., 2011) | Reunião de planejamento; Programação em pares; Integração contínua; Pequenas entregas; Refatoração | Flexibilidade à mudanças | Não encontrado |
| (THANGASAMY, 2012) | Times pequenos; Reunião diária; Pequenas entregas; Testes; Reunião de retrospectiva; Equipe auto gerenciável; Lições aprendidas (melhoria contínua); TDD ; Programação por par; Testes automatizados | Boa visibilidade das atividades | Gerenciar muitos times (dividir trabalho entre os times, garantir *releases* simultâneas etc) |
| (SANTOS, 2015) | Modularidade das tarefas; Pequenas entregas | Menor custo; Flexibilidade à mudanças; Orientado a pessoas; Comunicação entre o time; Trabalho mais estimulante; Aumento da produtividade; Documentação mais fácil para manter | Não encontrado |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (MANUJA et al., 2014) | Pequenas entregas; Times pequenos; Equipe auto gerenciável; Reunião diária; Reunião de retrospectiva; Documentação reduzida; TDD; Melhoria contínua | *Feedback* constante; Cliente satisfeito; Flexibilidade à mudanças | Gerenciar muitos times (dividir trabalho entre os times, garantir *releases* simultâneas etc); Mudança de *mindset*; |
| (GHOSH, 2012) | Reunião diária; Reunião de planejamento | Não encontrado | Diversidade cultural; Práticas ágeis em diferentes níveis |
| (DUKA, 2012) | Testes de regressão; Times auto gerenciáveis; Melhoria contínua; Reunião de retrospectiva; Reunião de planejamento; Times multi funcionais | Menor custo; Participação ativa do cliente; *Feedback* constante | Diversidade cultural; Práticas ágeis em diferentes níveis |
| (PINO et al., 2012) | Documentação reduzida; Teste de regressão | Muito útil em empresas pequenas; *Feedback* do cliente | Não encontrado |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (CHOUDHARI; SUMAN, 2012) | Pequenas entregas; Controle de versão (Git); Comunicação com cliente; Atualização da documentação; Testes de regressão; Testes de integração; Documentação reduzida; Reunião de planejamento | Motivação para o time; Satisfação do cliente; Código com mais qualidade; Comunicação constante com cliente | Na manutenção, ter pouca documentação é uma desvantagem; Design simples também não é bom para a manutenção; Podem ter tarefas individuais em diferentes versões do sistema |
| (SANTOS; OLIVEIRA, 2012) | Pequenas entregas; Melhoria contínua; Teste unitário | Flexibilidade à mudanças; Comunicação com cliente; Comunicação entre o time; Trabalho mais estimulante para os desenvolvedores; Reduzir custos na fase de manutenção; Aumento da qualidade do *software* | Não encontrado |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (SILVA; SANTOS; NETO, 2012) | Refatoração; Programação em pares; TDD; Reunião diária; Reunião de planejamento; Retrospectiva; Reunião de revisão | Visibilidade do andamento das atividades; Métricas para apresentar à gerência; Equipe madura | Demandas extras no andamento da *sprint* (Scrum); Dificuldade na monitoração do projeto utilizando (Kanban); Mudança de *mindset* da organização |
| (TRIMBLE; WEBSTER, 2013) | Pequenas entregas; Time pequeno; Priorização de tarefas; Lições aprendidas; *Daily*; Testes automatizados | Comunicação com o cliente; Satisfação do time e do cliente; *Feedback* constante do cliente | Não encontrado |
| (STOICA; MIRCEA; GHILIC-MICU, 2013) | Pequenas entregas; Reunião de planejamento; Documentação no código; Reunião de revisão | Comunicação com o cliente ; Flexibilidade à mudanças; Custo mais baixo para mudanças; Fácil de testar; Risco mais fácil de gerenciar; Cooperação entre o time | Custo total é alto; Reuniões de planejamento e design; Cliente pode pedir mais; Gerenciar muitos times (dividir trabalho entre os times, garantir *releases* simultâneas etc) |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (STERN; PORR, 2017) | Documentação reduzida; Pequenas entregas; Teste; Teste unitário; Reunião de revisão; Priorização das tarefas; Reunião diária; Teste de usuário; Integração contínua; Teste de regressão; Teste automatizado | Comunicação com o cliente; Relacionamento com o cliente; Foco nas entregas mais importantes | Não encontrado |
| (RAMOS et al., 2013) | Priorização das tarefas; Teste funcional; Teste unitário; Reunião diária; Programação em pares; *brainstormings* semanais; Controle de versão | Maior organização; *Feedback* constante do cliente; Comunicação entre os membros; Acompanhamento mais próximo | Mudança de *mindset* da organização |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (MARTINS, 2013) | Reunião diária; Reunião de retrospectiva; Priorização das tarefas; Programação em pares; Revisão de código; Reunião de planejamento | Visibilidade do andamento das atividades; Métricas para apresentar à gerência; Flexibilidade à mudanças; Transparência; Melhoria contínua; Melhor relacionamento entre equipe e cliente; Aumentar qualidade | Mudança de *mindset* da organização |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (CHOUDHARI; SUMAN, 2014) | TDD; Refatoração; Teste unitário; Testes automatizados; Padrões de código; Programação em pares; Pequenas entregas; Integração contínua; Documentação reduzida; Reunião de planejamento; Priorização das tarefas; Reunião de revisão; Teste de integração; Reunião diária; Revisão de código; Teste de aceitação | *Feedback* constante; Colaboração do cliente; Melhorar qualidade de código; Time confiante; Testes sendo reutilizados | Pouca documentação |
| (JAVANMARD; ALIAN, 2015) | Pequenas entregas; Equipe auto gerenciável; Reflexões periódicas (retro); Melhoria contínua; Pouca documentação; Times pequenos | Flexibilidade à mudanças; Comunicação com cliente | Gerenciar projetos grandes |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (KAUR; JAJOO et al., 2015) | Pequenas entregas; Times pequenos | Não encontrado | Gerenciar muitos times (dividir trabalho entre os times, garantir *releases* simultâneas etc), Dificuldade em estimar |
| (HEEAGER; ROSE, 2015) | Documentação reduzida; Testes frequentes; Testes automatizados; Testes regressivos; Pequenas equipes | Foco na qualidade; Comunicação com o cliente | Dependendo do tipo de manutenção, manter *sprint* com tempo fechado é um desafio; na manutenção, o trabalho costuma ser mais individual, atrapalhando a prática do trabalho em time; pouca documentação pode ser um problema na manutenção, principalmente se tiver pessoas novas na equipe |
| (SANTOS, 2015) | Priorização das tarefas; Pequenas entregas; Revisão de código; Alta cobertura de teste | Flexibilidade à mudanças; Comunicação com cliente; *Feedbacks* constantes; Colaboração da equipe | Falta de capacitação |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (KILPALA; KÄRKKÄINEN, 2015) | Pequenas entregas; Priorização de tarefas; Comunicação face a face; Reunião diária; Documentação reduzida; Reunião de retrospectiva; Reunião de revisão; Teste de integração; Reunião de Planejamento; Melhoria contínua | Comunicação constante entre cliente e PO; Comunicação com o time; Confiança do time | Diversidade cultural; Mudanças de escopo no meio da iteração |
| (KUHRMANN et al., 2016) | Documentação reduzida; Times auto gerenciáveis; Pequenas entregas; *Feedback*; Melhoria contínua; Programação em par; Reunião diária; Refatoração | Comunicação com usuário; Realização de testes; Flexibilidade à mudanças; Comunicação entre o time | Gerenciar muitos times (dividir trabalho entre os times, garantir *releases* simultâneas etc) |
| (TARWANI; CHUG, 2016) | Pequenas entregas; Monitoração contínua; Times pequenos; Programação em pares | Clientes mais satisfeitos; Flexibilidade à mudanças | Mudança de *mindset* da equipe; Ruim para projetos grandes; Aumento de custo (em geral) |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (NAZ; KHAN; AAMIR, 2016) | Time auto gerenciável; Reunião de planejamento ; Reunião diária; Reunião de revisão; Reunião de Retrospectiva; Pequenas entregas | Flexibilidade à mudanças; Comunicação com o time | Não encontrado |
| (ZANOTTI; KAYLOR; DAVIDSEN, 2017) | Reunião diária; Reunião de planejamento; Reunião de revisão; Reunião de Retrospectiva; Pequenas entregas | Colaboração do time; Comunicação com o cliente; Flexibilidade à mudanças | Não encontrado |
| (ABRAHAMSSON et al., 2017) | Times pequenos; pequenas entregas; melhoria contínua; teste de regressão; teste automatizado; teste unitário | *Feedback* constante do cliente; Simplicidade; Melhoria contínua no *software*; Melhoria nos custos | Times muito grandes, produtos muito grandes |
| (BOGOJEVIĆ et al., 2017) | Reunião de revisão; Times pequenos; Teste de aceitação; Teste unitário | Colaboração do cliente; Flexibilidade à mudanças; Confiança do time;; Motivação para o time | Difícil ter um modelo único que se encaixa no processo da organização |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (SILVA, 2018) | Melhoria contínua; Priorização de tarefas | Pró atividade para resolução de problemas, Foco na qualidade, Simplicidade | Projetos grandes |
| (LOUS et al., 2018) | Programação em pares; TDD; Reunião diária; Reunião de retrospectiva; Entrevista com usuário; Controle de versão de código; Revisão de código; Integração contínua; Pequenas entregas; Reunião de planejamento | Comunicação com o time; Os métodos são fáceis de usar juntos; Acompanhamento do time mais próximo do líder | Times alocados em diferentes países; Diversidade cultural |
| (JR; DANTAS, 2019) | Reunião diária; Reunião de planejamento; Reunião de retrospectiva | Não encontrado | Não encontrado |

Continua na próxima página

**Tabela 15 – continua na página anterior**

| Autores | Práticas ágeis | Vantagens | Desafios |
|---|---|---|---|
| | | | |
| (WAZLAWICK, 2019b) | Pouca documentação; Entrega contínua; TDD ; Teste unitários; Testes de regressão; Testes de integração; Testes de aceitação; Controle de versão; Revisão de código; Alta cobertura de testes; Reunião de Retrospectiva | Não encontrado | Não encontrado |
| (COSTA, 2016) | Reunião de planejamento; Documentação reduzida; Pequenas entregas; Teste unitário; Teste funcional; Reunião de revisão; Refatoração; Padrão de código; Reunião de retrospectiva; Reunião diária | Comunicação com o cliente; Comunicação com o time; *Feedback* constante do cliente | Equipes grandes; *Mindset* de toda equipe precisa estar alinhado |