UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

ERIC RODRIGUES FERREIRA

UMA TAXONOMIA DE TESTES DE SOFTWARE NO CONTEXTO ÁGIL

QUIXADÁ

2022

ERIC RODRIGUES FERREIRA

UMA TAXONOMIA DE TESTES DE SOFTWARE NO CONTEXTO ÁGIL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Profa. Dra. Carla Ilane Moreira Bezerra.

QUIXADÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F44t Ferreira, Eric Rodrigues.
Uma taxonomia de testes de software no contexto ágil / Eric Rodrigues Ferreira. – 2022.
74 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2022.
Orientação: Profa. Dra. Carla Ilane Moreira Bezerra.

1. Teste de Software. 2. Desenvolvimento ágil de software. 3. Classificação. I. Título.

CDD 005.1

ERIC RODRIGUES FERREIRA

UMA TAXONOMIA DE TESTES DE SOFTWARE NO CONTEXTO ÁGIL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ____/ ____/ ____.

BANCA EXAMINADORA

Profa. Dra. Carla Ilane Moreira Bezerra (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Jeferson Kenedy Morais Vieira
Universidade Federal do Ceará (UFC)

Prof. Dr. Ivan do Carmo Machado
Universidade Federal da Bahia (UFBA)

À minha família e meus amigos mais próximos. Em especial aos meus pais. Pai e Mãe, obrigado por todos os cuidados e bons conselhos, sem eles, certeza que hoje eu não estaria aqui e nada disso estaria sendo possível. Após ser por mim, tudo isso também é por vocês.

AGRADECIMENTOS

Antes de tudo, agradeço aos meus pais, Antônia Rosinete Rodrigues Maciel e Eriberto Ferreira Lima, que sempre acreditaram em mim e me incentivaram a correr atrás dos meus objetivos e sonhos, com a célebre frase: "A gente só consegue alguma coisa na vida, com estudo!". Que mesmo apesar de todas as dificuldades, se esforçaram para me ofertar uma educação de qualidade e digna, sempre dentro do que era possível, apesar de todas as dificuldades. Muito obrigado por tudo isso! Após ser por mim, tudo isso é por vocês e para vocês.

Agradeço a Profa. Dra. Carla Bezerra, por aceitar me orientar nesta jornada que é o TCC, por ter me guiado durante todo esse percurso, sempre analisando as ideias e prezando pelo que seria de mais qualidade para o trabalho, bem como o que estaria no alcance de ser feito dentro do contexto da atividade. Que por meio desse nosso trabalho, possamos contribuir de forma positiva com a Engenharia de Software, bem como a aplicação de suas práticas no mercado.

Agradeço também a UFC na totalidade, passando pelos professores e os demais servidores que trabalham arduamente dia a dia para que tudo isso funcione na mais perfeita forma. Pela ótima estrutura cedida, recursos e oportunidades a quão estamos expostos.

Por último, mas, não menos importante, agradeço a mim mesmo, por todo o foco, disciplina e perseverança, sem esses pilares este trabalho não seria possível, dado a complexidade do mesmo, empecilhos que surgiram e por se tratar de uma área que até então era distante para mim (pesquisa e desenvolvimento de trabalhos científicos). Que o meu futuro seja ainda mais próspero e satisfatório.

E um muito obrigado a todos que estiveram comigo durante esse árduo, longo e satisfatório processo.

“Quando eu vier, eu venho.”

(Nícolas Gabriel Rodrigues Silva)

RESUMO

Após os anos 2000, com a publicação do manifesto ágil, a popularização e adesão às metodologias ágeis no desenvolvimento de software cresceu de forma exponencial. Estabeleceu-se um novo cenário em que as coisas são feitas de forma iterativa e incremental, tendo o cliente como um aliado fundamental. Assim, nesse novo cenário ágil, geralmente ao final de cada iteração são realizados os testes do que foi desenvolvido, mas os mesmos também podem ocorrer no meio do período, ou até ao inicio, dependendo do projeto, visando assegurar a qualidade do que está sendo entregue ao cliente. Como as entregas são rápidas, várias abordagens vem sendo propostas para os testes ágeis, especialmente testes automatizados. No entanto, não foi identificado na literatura uma consolidação das abordagens, ferramentas e técnicas utilizadas para os testes no contexto de desenvolvimento ágil. Dessa forma, o presente trabalho visa criar e propor uma taxonomia de testes de software no contexto ágil de desenvolvimento de software. Para isso, foi realizado um mapeamento sistemático na literatura, onde foram selecionados 16 trabalhos, sendo extraídas informações, como: (i) técnicas de testes de software, (ii) etapas dos testes, (iii) ferramentas de testes, (iv) modelos de testes ágeis e (v) metodologias de testes ágeis. A partir dos resultados do mapeamento foi criada a taxonomia de testes ágeis com base na aplicação da técnica de taxonomias facetadas ou multifacetadas. A taxonomia de testes ágeis fornece uma base sólida para pesquisadores e times de software utilizarem como base nas tomadas de decisões como quais tipos de testes empregar na realização de um projeto, e isso simplificando em tempo de treinamento, aplicação e custos por parte da aplicação de testes.

Palavras-chave: Teste de Software; Desenvolvimento ágil de software; Classificação.

ABSTRACT

After the 2000s, with the publication of the agile manifesto, the popularization and adherence to agile methodologies in software development grew exponentially. A new scenario was established in which things are done in an iterative and incremental way, with the customer as a fundamental ally. Thus, in this new agile scenario, usually at the end of each iteration, tests of what was developed are carried out, but they can also occur in the middle of the period, or even at the beginning, depending on the project, in order to ensure the quality of what is being delivered to the customer. As deliveries are fast, several approaches have been proposed for agile testing, especially automated testing. However, a consolidation of approaches, tools and techniques used for testing in the context of agile development was not identified in the literature. Thus, this work aims to create and propose a software testing taxonomy in the context of agile software development. For this, a systematic mapping in the literature was carried out, where 16 works were selected, and information was extracted, such as: (i) software testing techniques, (ii) test stages, (iii) testing tools, (iv) models of agile testing and (v) agile testing methodologies. From the results of the mapping, the taxonomy of agile tests was created based on the application of the technique of faceted or multifaceted taxonomies. The taxonomy of agile tests provides a solid basis for researchers and software teams to use as a basis for decision-making such as which types of tests to employ in carrying out a project, and this simplifies training time, application and costs on the part of the application of tests.

Keywords: Software Testing; Agile software development; Classification.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

No decorrer dos últimos anos, a área da engenharia de software vem crescendo bastante e esse crescimento está diretamente ligado a evolução das suas novas sub-áreas que diariamente vem sendo desenvolvidas e sendo continuamente integradas (USMAN *et al.*, 2017). Para buscar entender melhor essa gama de conhecimentos que continuamente vem sendo desenvolvidas na engenharia de software, estão sendo proposta taxonomias, que visam estruturar e entender melhor esses diversos panoramas do conhecimento (USMAN *et al.*, 2017).

O conhecimento em diversas outras áreas é classificado e catalogado, por meio da prática de criação de taxonomias. Podemos compreender por taxonomia, como sendo um esquema de classificação (STEVENSON, 2010). Deste modo, uma taxonomia permite a elaboração e descrição de termos, bem como suas relações no contexto de sua área do conhecimento (USMAN *et al.*, 2017). Com isso, surgiu-se a necessidade da existência de terminologias comuns, para realizar a classificação do conhecimento (BRITTO *et al.*, 2016).

Desde os anos 2000, vem se notando um crescente aumento em publicações de taxonomias nas áreas que compreendem a engenharia de software, em uma ampla gama de locais, como os principais periódicos e conferências do domínio segundo Usman *et al.* (2017). Foi catalogado por Usman *et al.* (2017), que tais taxonomias podem ser agrupadas em algumas áreas do conhecimento em engenharia de software, como: construção com um total de 19,55% das taxonomias, projetos com 19,55%, requisitos com 15,50% e manutenção com um total de 11,81% (USMAN *et al.*, 2017).

Ultimamente, os métodos ágeis são cada vez mais difundidos no mercado, difusão essa que ocorre nas mais diversas áreas do conhecimento, como: engenharia de software e demais engenharias, bancos, fabricação em larga escala e etc (ALAIDAROS *et al.*, 2019).

Olhando mais especificamente para as áreas que envolvem a engenharia de software, notou-se que nos últimos tempos as grandes indústrias de software e organizações adotaram um processo de automação de testes, desde cenários mais simplistas até os mais sofisticados, aplicando práticas ágeis aos mesmos, inclusive em produtos ou aplicações de execução crítica, pondo em desuso modelos de processos mais formais (HYNNINEN *et al.*, 2018).

Dado que até em contextos mais críticos de execução, se está empregando agilidade, para garantir a qualidade dos produtos desenvolvidos a aplicação das práticas de testes se tornam necessárias, entendendo que uma das melhores formas de verificar a qualidade de um produto é o teste de software (BARRAOOD *et al.*, 2021). Os desenvolvedores necessitam acompanhar o

ritmo acelerado do processo para conseguirem dar vasão a alta demanda e entregas, para isso, as estimativas de desenvolvimento bem como as de testes tem uma importância fundamental (KAUR; KAUR, 2018).

Compreendendo que os testes de software garantem que, ao final do processo de desenvolvimento, o resultado obtido é o que se esperava construir (SAWANT *et al.*, 2012). Além dessa visão mais ao final do processo, a prática dos testes consegue identificar falhas e erros de sistema durante o processo de desenvolvimento, onde a resolução dos mesmos impacta positivamente na qualidade do produto final (SAWANT *et al.*, 2012).

Com isso, os métodos ágeis tornam a prática de testes um componente essencial em algumas etapas do desenvolvimento, buscando garantir a qualidade contínua do produto (ARRIETA *et al.*, 2016). Contudo, notou-se que se tratando de metodologias ágeis, suas aplicações são adaptadas conforme o contexto e aplicação do local de trabalho, como organizações quem adotam algumas práticas ágeis em seu processo de desenvolvimento (ATAWNEH, 2019).

Assim precisamos compreender que no contexto ágil, os testes de software são complexos e ainda tem muitos desafios e barreiras para sua aplicação (BARRAOOD *et al.*, 2021). Já que a mesma se concentra em ter o cliente envolvido, execução de iterações curtas com entregas regulares (FOWLER *et al.*, 2001). Assim, compreende-se que a prática de testes ainda é um campo turvo, a relação entre complexidade, desafios e barreiras, se dá por conta do desconhecimento sobre como as mesmas são abordadas no contexto ágil de desenvolvimento.

Neste contexto, o objetivo deste trabalho é propor uma taxonomia de testes de software no contexto de desenvolvimento com metodologias ágeis. Esta taxonomia poderá ser utilizada por projetos ágeis, engenheiros do domínio, pesquisadores do domínio e todos aqueles interessados na aplicação de testes no contexto ágil de desenvolvimento de software e permitirá ter um entendimento mais assertivo e estruturado sobre testes.

1.1 Objetivos

Nesta Seção é apresentado o objetivo geral e os objetivos específicos do presente trabalho.

1.1.1 Objetivo geral

O objetivo deste trabalho é propor uma taxonomia de testes de software no contexto de desenvolvimento ágil de produtos de softwares.

1.1.2 Objetivos Específicos

1. Realizar um mapeamento sistemático da literatura para identificar as práticas de testes utilizadas no contexto de desenvolvimento ágil.
2. Identificar um conjunto de práticas de testes ágeis a partir do mapeamento sistemático da literatura.
3. Elaborar uma taxonomia de testes ágeis a partir dos resultados do mapeamento sistemático.

1.2 Organização do Documento

Este trabalho está organizado da seguinte forma: no Capítulo 2, seguinte ao atual, são apresentados os termos e conceitos necessários para o entendimento do trabalho. No Capítulo 3, contém os procedimentos metodológicos que serão seguidos durante a realização deste trabalho. O Capítulo 4 apresenta os trabalhos relacionados com o presente trabalho. O Capítulo 5 apresenta o mapeamento sistemático para identificação de práticas de testes ágeis na literatura. O Capítulo 6 apresenta a taxonomia de testes ágeis elaborada a partir dos resultados do mapeamento. O Capítulo 7 descreve as limitações e ameaças a valida do presente trabalho, e por fim, o Capítulo 8 exibe as conclusões e as propostas de trabalhos futuros do presente trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos mais relevantes necessários para o entendimento deste trabalho. Na Seção 2.1 serão apresentados conceitos básicos de testes de software. Na Seção 2.2 serão expostos os conceitos e práticas que envolvem as metodologias ágeis. Na Seção 2.3 será abordado como as práticas de testes são abordadas nos contextos ágeis de desenvolvimento. Na Seção 2.4 serão descritos e decorridos sobre os conceitos mais fundamentais a respeito de taxonomias. Na Seção 2.5 conceitos que evidenciam a relação entre taxonomias e engenharia de software serão apresentadas e um panorama do presente cenário desta relação.

2.1 Testes de software

Dada a crescente de complexidade exponencial das atuais injunções de softwares (ordenações expressas, coerção de regras de negócios, requisitos, casos de uso e etc), com o respectivo aumento da pressão competitiva no mercado, que impulsiona a busca por garantia de qualidade em desenvolvimento de produtos de softwares (ARUMUGAM, 2019). Compreende-se que as atividades de teste de software é uma parte inevitável do ciclo de vida de desenvolvimento, e se pondo em linha com um alto nível de criticidade tanto no pré-quanto no pós-processo de desenvolvimento, tornando-se algo que deve ser suprido com metodologias aprimoradas e eficientes bem como as técnicas como foi citado por Arumugam (2019).

Pode-se entender como definição de testes, como um processo de avaliação sobre um sistema de software específico, para validar se o mesmo atende a sua especificação original dos requisitos ou não (ARUMUGAM, 2019). Teste de software buscam encontrar *bugs*, erros ou requisitos que sem encontrem em ausência no produto desenvolvido ou em desenvolvimento.

Foram encontradas na literatura, algumas definições sobre o que é teste de software:

1. Entende-se como teste de software, o processo de pôr um software em execução, de modo a buscar ou encontrar falhas no mesmo, como foi dito por Myers *et al.* (2011).
2. As atividades de teste fazem parte de processos mais amplos relacionados a qualidade de software, como os processos de verificação e validação de um software (WAZLAWICK, 2019).

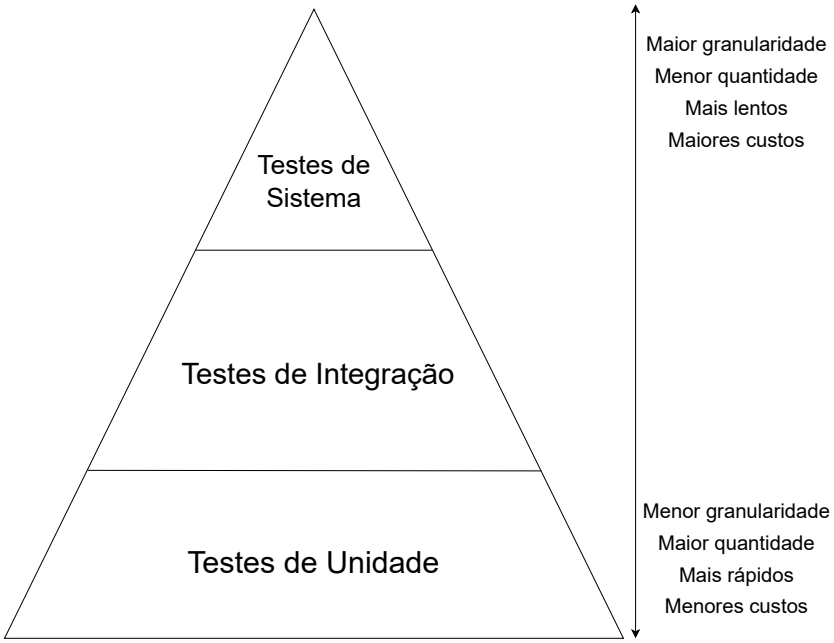
No contexto de testes, existem etapas e técnicas para execução e aplicação dos mesmos, onde as etapas básicas de teste de software são: teste de unidade, de integração e

de sistema (KOREL, 1990). As técnicas consistem em teste de caixa branca, cinza e preta (JOVANOVIĆ, 2006).

2.1.1 Etapas de teste de software

Compreendendo as etapas de teste como exemplificado na Figura 1:

Figura 1 – Pirâmide das etapas de testes.



Fonte: Adaptado de Valente (2020).

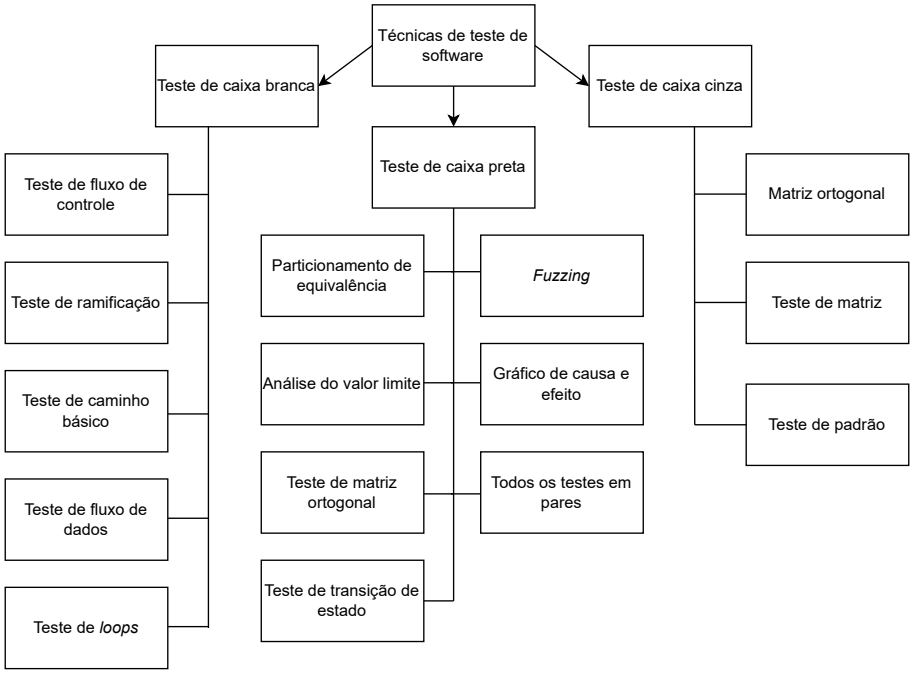
- **Teste de unidade:** são testes que buscam testar pequenas unidades de código. Em geral, são classes completas ou partes específicas de suas implementações, como um método em específico ou um conjunto de métodos, onde as mesmas são testadas de forma isolada para com o restante do software. Um teste de unidade é a chamada a métodos de uma dada classe para verificar se os retornos da execução, são como os valores esperados (VALENTE, 2020).
- **Teste de integração:** ocorrem ao passo que os testes de unidade já foram executados, dado que as classes ou unidades foram testadas e validadas individualmente. Após isso, chegasse ao ponto de onde as unidades testadas anteriormente, iniciarem o trabalho em conjunto entre si, havendo assim uma integração de execuções para uma completude maior das funcionalidades. Com isso, como as unidades se integraram, há a necessidade de testar a integração entre as mesmas.
- **Teste de sistema:** é executado ao passo que às duas etapas de testes citadas, foram

anteriormente executadas e validadas. Nesta presente etapa, o teste consiste em verificar uma versão mais atual do software, onde sobre o mesmo vão realizar uma série de ações estando do ponto de vista do usuário final, onde o mesmo irá navegar na *interface* e executar os percursos verificando se para uma dada ação está sendo retornado o que se espera ou se as pós-ações dadas a uma ação anterior são as como as aguardadas.

2.1.2 Técnicas de teste de software

As técnicas de teste exemplificadas na Figura 2:

Figura 2 – Diagrama das técnicas de testes.



Fonte: Adaptado de Arumugam (2019).

– **Teste caixa-branca** ou comumente conhecido como teste transparente ou teste de vidro, o mesmo é orientado a lógica, o mesmo é considerado ser significativamente eficaz, pois esta técnica não testa somente a funcionalidade implementada no software, mas também faz testes na estrutura interna da aplicação. Quando vai-se projetar os testes, para prosseguir com os testes de caixa branca, se fazem necessárias habilidades de programação e desenvolvimento para a projeção dos mesmos (ABRAN *et al.*, 2004).

  Essa técnica tem aplicação bastante ampla, dado que a mesma pode ser aplicada a todas três etapas de teste de software, sendo: teste de unidade, integração e de sistema, citados anteriormente. Uma função extremamente importante relacionada a essa técnica de teste, é que a mesma cumpre a necessidade de determinar se os produtos de software mantêm

a proteção adequada dos dados. Como essa técnica recorre à lógica interna do software, a mesma se torna capaz de testar todos os caminhos independentes de um módulo, indo em cada lógica, cada decisão exercida ao nível de lógico do software, são iterados todos os *loops* e as estruturas de dados mais internas também são exercitadas. Contudo, os testes de caixa branca se torna uma técnica extremamente complexa, pois a mesma exige habilidades hábeis de programação e desenvolvimento (MILLER; HOWDEN, 1981).

– **Teste caixa-preta** é uma técnica que consiste essencialmente em realizar os testes na aplicação, mas sem considerar, ou sem entrar em detalhas relacionados a estruturação e implementação acerca do produto testado. Onde o teste é executado de tal forma que visa abranger todas ao o máximo de funcionalidades da aplicação em teste, para determinar se a mesma está atendendo ou não aos requisitos especificados pelo cliente, ou usuário. Onde o mesmo consegue encontrar funcionalidades com estão com funcionamento ou execução incorreta, onde cada funcionalidade é testada em seu mínimo e seu máximo com valores-base em cada caso. Dada que está técnica é fundamentalmente mais simples e fácil de ser aplicada, a mesma é a mais difundida e utilizada (ABRAN *et al.*, 2004; MILLER; HOWDEN, 1981).
– **Teste caixa-cinza** é compreendido como uma junção das técnicas de teste de caixa branca e preta, agregando o que há de melhor em cada uma das técnicas citadas. Essa categoria de técnica surgiu por conta da necessidade de testarem as funcionalidades de uma forma considerada melhor em certo cenários, dado que o testador tem conhecimento da estrutura interna da aplicação tanto em termo de organização quanto implementação, com isso, o teste pode ser feito de uma maneira mais eficiente que considere a estruturação interna da aplicação (JOVANOVIĆ, 2006).

2.2 Metodologias ágeis

A engenharia de software passou e continua passando por inúmeras mudanças e atualizações, estas com o intuito de acompanhar e se manter em alinhamento com os novos avanços tecnológicos, com os requisitos dos novos negócios modernos que vem emergindo. Para tal, foi-se necessário o desenvolvimento de abordagens que fossem mais eficazes e eficientes para se chegar ao produto final de software, o desenvolvimento ágil de softwares é uma dessas abordagens (AL-SAQQA *et al.*, 2020).

O desenvolvimento ágil, é uma abordagem consideravelmente leve, proposta para

suprir as limitações dos métodos mais tradicionais de desenvolvimento, como as abordagens em cascata, abordagem iterativa e incremental, espiral, evolutiva (MALL, 2018). Onde, segundo Braude e Bernstein (2016), os problemas destes métodos mais tradicionais está ligado ao fato de eles seguirem a risca ou de forma pesada uma série de procedimentos estáticos e sem interseções de execução entre outro, como:

1. O planejamento do projeto deve ser feito com antecedência.
2. Todos os requisitos devem estar escritos.
3. Projeto completo que satisfaça os requisitos escritos anteriormente.
4. Construir um produto de software que satisfaça todos os requisitos escritos.
5. Teste total do software para verificar se o mesmo atende aos requisitos e *design* propostos para o produto, ao final ou como último passo do ciclo.

Com tudo isso, surgiu-se a necessidade de abordagens mais leves, onde os principais objetivos dessas metodologias é acelerar o processo de desenvolvimento e fornecer respostas mais eficazes quando alterações são solicitadas (BRAUDE; BERNSTEIN, 2016).

Essas abordagens ágeis vieram para suprir certas limitações destes métodos mais tradicionais, como: redução de sobrecarga, redução de custo, adição de flexibilidade em diversas frentes, que proporciona a fácil adoção do processo de mudança de requisitos, onde tudo é feito gerenciando as tarefas (*tasks*) e as coordenando com base em um conjunto de valores e princípios segundo Al-Saqqa *et al.* (2020). Essas abordagens mais leves e flexíveis de desenvolvimento de produtos de software são chamadas metodologias ágeis de desenvolvimento de software (BRAUDE; BERNSTEIN, 2016).

Ágil é um campo extremamente amplo de crenças relacionadas ao desenvolvimento de software. Pode-se compreender como um quadro conceitual de trabalho, que para a engenharia de software, começa com uma fase inicial de planejamento, que se segue por um caminho para uma fase de implantação, com interações durante todo o ciclo de vida de um produto de software (AL-SAQQA *et al.*, 2020). O objetivo inicial das metodologias ágeis é em reduzir o nível de sobrecarga durante o processo de desenvolvimento do software, fornecer capacidade de adoção a mudanças sem pôr em risco todo o processo ou sem causar retrabalho, ou retrabalho em excesso.

Esses métodos mais flexíveis e leves, como foi falado por Jammalamadaka e Krishna (2013), tiveram sua origem e promoção através de uma aliança que se deu entre um total de 17 engenheiros de software que em 2001 realizaram a publicação do 'Manifesto de desenvolvimento ágil' (BECK *et al.*, 2001), onde um conjunto de valores e princípios para agilidade são descritos.

Compreendendo os quatro valores apresentados no manifesto:

1. **Indivíduos e interações acima de processos e ferramentas:** O primeiro grande valor apresentado no manifesto enfatiza que os processos formais e seus ambientes de arredondamentos fechados como sendo um fator-chave no processo de desenvolvimento de software, é incorreto, porque quanto mais importante for a comunicação, interação entre as equipes de desenvolvimento, maior será a qualidade final do produto de software desenvolvido (RODRÍGUEZ *et al.*, 2019).
2. **Software que trabalha sobre uma documentação completa:** A documentação em qualquer processo de desenvolvimento ágil de software é um componente extremamente valioso e de grande importância, mas a quantidade de tempo e recursos disponibilizados para a construção da mesma devem ser controlados e otimizados para não gerarem sobrecarga no andamento do processo de desenvolvimento. Contudo, tem-se que compreender que a usabilidade de uma documentação desenvolvida de forma intensiva é uma das grandes limitações tradicionais no processo de desenvolvimento (LARSON; CHANG, 2016), desde a escrita clara e objetiva, a sincronização das documentações com o código desenvolvido, tendo em vista que isso é um processo árduo e demorado em especial quando os requisitos mudam com muita frequência. Contudo, isso, tais documentações são raramente utilizadas fora da fase de implantação que se dá mais ao início do processo.

   Assim, a documentação no cenário ágil de desenvolvimento, são documentos que servem ao software em funcionamento (RODRÍGUEZ *et al.*, 2019) e, além disso, adicionam ainda mais valor ao processo de desenvolvimento. O manifesto traz à afirmativa de que o progresso real nas metodologias ágeis, são medido e quantificados através do software testado, em vez de documentação, pois é menos ambíguo e consegue responder de forma imediata se atende ou não a demanda (GUPTA; GOUTTAM, 2017).
3. **Colaboração do cliente sobre a negociação do contrato:** As metodologias ágeis de desenvolvimento de software foram desenvolvidas para terem como lidar com a constante mudança de requisitos em qualquer estágio do processo, gerando um *feedback* praticamente que constante ao cliente, as negociações e colaboração com as equipes de desenvolvimento são exigidos com bastante frequência durante o processo, para buscar atingir a todas as necessidades reais que os clientes tenham, ao contrário de acordos e contratos formais. Entretanto, os contratos que definem e descrevem o relacionamento entre as equipes de desenvolvimento e os clientes, e descreve o negócio de software que se está sendo

desenvolvido ainda é necessário (RODRÍGUEZ *et al.*, 2019).

4. **Responder às mudanças ao invés de seguir um plano:** Tendo em mente que o processo de desenvolvimento do software está em sequência e andamento, tanto os desenvolvedores quanto os clientes adquirirão mais conhecimento e uma maior compreensão sobre o sistema na totalidade, assim, a adição ou o possível cancelamento de alguns requisitos podem vir a se tornar necessários. A prioridade no manifesto ágil é atribuída a rápida resposta as mudanças que ocorrem durante o ciclo de vida do processo de desenvolvimento, ao contrário de seguir um plano estritamente definido, dado que o objetivo final é atingir a satisfação do cliente.

Mas além dos quatro valores, o manifesto estabelece doze princípios (BECK *et al.*, 2001), sendo:

- **Princípio 01:** Este princípio enfoca que, a maior prioridade no contexto ágil de desenvolvimento, tem que ser satisfazer o cliente, isso através de antecipação das entregas, aliado a entregas contínuas, para cada vez mais ir tornando o software mais valioso para o cliente. Pois, compreende-se que a entrega antecipada e continuamente feita irá criar um ambiente de confiança e flexibilidade entre o cliente para com a equipe de desenvolvimento, com isso, a equipe de desenvolvimento passará a receber *feedbacks* mais exatos das compreensões dos requisitos do produto que eles tiveram, por meio do *feedback* do cliente, que vai cada vez se tornando mais significativo. Com isso, os clientes tendem a se tornarem mais confiantes quando passam a ter em mãos, uma versão do produto, por mais que seja uma versão mais básica, assim eles podem elencar novos requisitos ou priorizar outros já existentes.
- **Princípio 02:** Seja bem-vinda a constante mudança de requisitos, mesmo que sejam ao final do processo de desenvolvimento. No contexto ágil os processos se aproveitam da mudança, para gerar vantagem competitiva para o cliente. Os impactos correspondentes a adoção dessas mudanças nos requisitos, devem ser minimizados por meio da utilização de práticas como *refactoring*.
- **Princípio 03:** Entregar com frequência uma versão funcional do software, entrega essa compreendida no intervalo de semanas a um mês, mas dando preferência por cumprir sempre a escala de tempo mais curta possível, entre uma entrega e outra. Onde este princípio coloca em restrição o primeiro princípio, recomendando um início e por muitas vezes as entregas para satisfazer as necessidades e dores do cliente.

- **Princípio 04:** Tanto os clientes como os desenvolvedores devem trabalhar diariamente com foco no projeto. Pois, essas integrações e relacionamentos de proximidade entre o cliente e o desenvolvedor, visam dar *feedbacks* mais assertivos e responder às perguntas levantadas pela equipe de desenvolvimento.
- **Princípio 05:** Indivíduos motivados para construir um software de qualidade em torno dos mesmos. Ceder o devido ambiente e suporte a que necessitarem, e principalmente confiar nos mesmos para realizar o trabalho. Pois, compreende-se que os componentes da equipe é o fator alicerce para o sucesso de um projeto, onde os métodos ágeis confia nesses indivíduos e os dão poderes de tomada de decisão mais apropriada para eles executarem seu trabalho, como, por exemplo: se eles julgarem melhor, eles podem alterar as etapas do processo de desenvolvimento caso julguem que alguma dada parte será um obstáculo para a equipe na totalidade.
- **Princípio 06:** A comunicação face-a-face é o melhor método tanto em eficiência quanto eficácia quando se fala em transmitir informações, tanto para dentro quando para fora da equipe de desenvolvimento. Onde este princípio dá enfoque total sobre a comunicação humana, a mesma direta entre os membros da equipe ágil, ao contrário de ficar fazendo utilização de comunicação escrita, por especificação ou planos descritos e escritos.
- **Princípio 07:** A medida primária de progresso se dá em medida que o software está funcionando. Pois, fragmentar o produto na totalidade em pedaços menores, consequentemente irá refletir em entregas menores, com isso pode-se haver a antecipação das entregas para com o cliente, as trazendo com mais frequência, sendo essa uma boa forma de medir o progresso do andamento do projeto, pois assim, os métodos de medição irã dar impacto mais honesto para os códigos em execução.
- **Princípio 08:** Os processos de contexto ágil, tendem a promover o desenvolvimento mais sustentável. O desenvolvimento sustentável quer dizer que a equipe deve manter um ritmo mais constante de trabalho, ou seja, realizar entregas sem erros finais não deve ser o objetivo final a ser alcançado de forma antecipada, consumindo todos os recursos que se tinham disponíveis. Ao contrário de entregar um alto volume de peças com qualidade, o importante é obter um *feedback* correto e assertivo é muito melhor até que um objetivo final seja eventualmente atingido.
- **Princípio 09:** O aumento da agilidade está ligado a um ótimo *design* e a atenção contínua à excelência. Uma obrigação de extrema importância é a entrega de código da mais alta

qualidade possível, entendendo que esta é uma obrigação que deve ser seguida sempre que dentro de um contexto ágil, por mais que tenham que haver refatorações (*refactoring*) de código como resposta a alguma mudança exigida.

- **Princípio 10:** Simplicidade. O objetivo de maior foco é em produzir um produto de software simples, mas que consiga lidar com as mudanças repentinas que não estão no escopo, mas que se sabe que podem ocorrer de forma inesperada, em consonância a se manter atendendo os requisitos e dores do cliente.
- **Princípio 11:** Quanto mais organizada a equipe, melhores serão as arquiteturas que irã emergir desta equipe, melhor serão os requisitos elicitados por essa equipe. Toda e qualquer equipe ágil, tem que ser auto-organizada, que compartilham responsabilidade de dado um projeto, determinarem a melhor maneira de tratar os ocorridos e decisões, com base nos conhecimentos prévios que os membros adquiriram ao longo do processo de desenvolvimento para buscar atingir a melhor estrutura e organização.
- **Princípio 12:** Em intervalos de tempo bem delimitados, decididos e regulares, a equipe na totalidade reflete sobre como a mesma pode vir a se tornar mais eficaz, com isso os membros da equipe ajustam e se sintonizam para porem seus comportamentos e ações conforme o que foi refletido.

Compreendendo todos esses pontos e devido aos custos inferiores, aliados a uma melhor produtividade, a uma melhor qualidade e satisfação por parte dos clientes, as metodologias ágeis causaram grande impacto na indústria de desenvolvimento de software, e um espaço de tempo de pouco mais de duas décadas de existência (ERICKSON *et al.*, 2005).

Com isso, compreende-se como sendo as principais vantagens do desenvolvimento de software que se dá no contexto ágil segundo Choudhary e Rakesh (2016) são:

- Melhora na comunicação e coordenação para com os membros da equipe.
- Rápidos lançamentos.
- *Design* flexível.
- Processo compreendido como mais razoável.

Métodos ágeis são basicamente processos que suportam e se integram de forma forte a filosofia ágil de desenvolvimento, ou seja, suporta e faz uso frequente dos valores e princípios ágeis. Cada método consiste basicamente na junção de diversas práticas e cerimônias. Ccada método vem a se diferir dos outros, por escolher-lhe um conjunto apropriado de terminologias e práticas que o mesmo julga como adequadas e necessárias (ELBANNA; SARKER, 2015).

Existem inúmeras categorias de metodologias ágeis, para resolverem *n* problemas, em *n* contextos, como: *Scrum*, *Extreme Programming* (XP), *Feature Drive Development* (FDD), *Dynamic System Development Model* (DSDM), *Crystal*, etc. Onde um fato importante de se entender, é que cada método desses tem suas funções internas e próprias, princípios próprios, ciclo de vida, funções, vantagens e também as desvantagens.

2.2.1 Scrum

Foi proposto por Schwaber e Beedle (2002) uma implementação mais concreta de um *framework* que fosse ágil, onde a aplicação do mesmo seria para o gerenciamento de projetos no processo de desenvolvimento de software, onde o *framework* em sua proposta é iterativo. Onde a proposta do mesmo é entregar o maior valor possível no menor espaço de tempo possível, onde o *scrum* é uma metodologia ágil que se concentra na orientação a equipes.

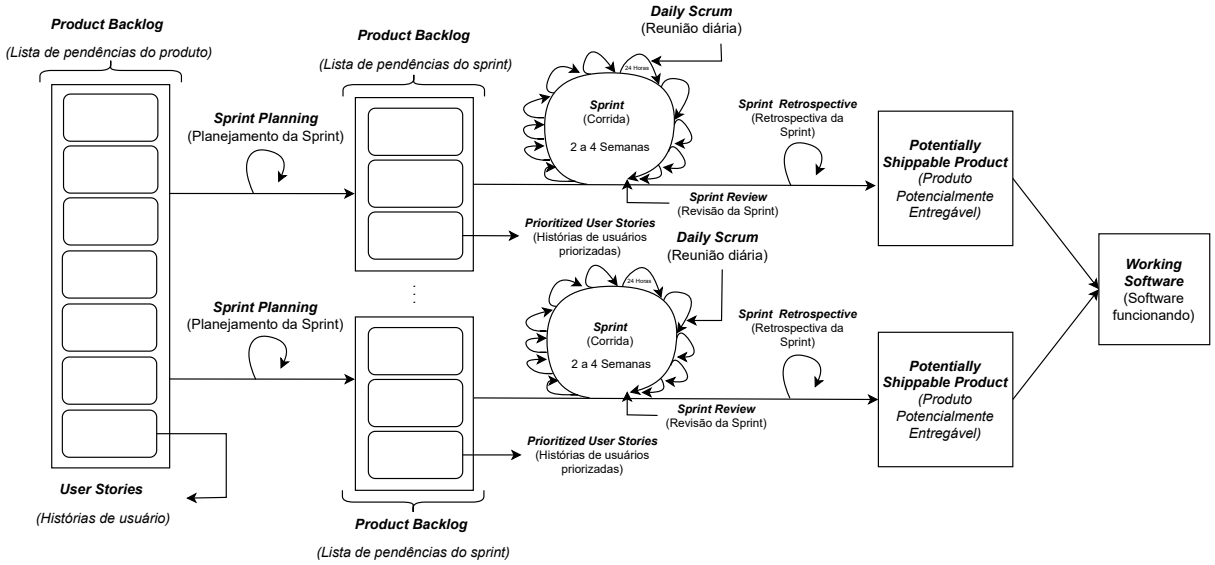
Onde o *scrum* estabelece um período curto de tempo em cada iteração, e essa curta caixa de tempo é nomeada no *framework* como *sprint*, em que o produto é desenvolvido incrementalmente, a mesma produz um artefato diferente ou mais por iteração, que é, coordenando o trabalho de uma dada *sprint* (SCHMIDT, 2016). Onde o *scrum* é considerado uma das metodologias ágeis mais utilizadas ultimamente (RODRÍGUEZ *et al.*, 2019).

Esta popularidade atrelada ao *scrum* se dá por conta de sua simplicidade tanto de aplicação quando de execução, e devido ao seu foco em gerenciar softwares, pois o mesmo se baseia em questões fundamentais, ao contrário de se fincar em técnicas de desenvolvimento de formas restritivas, tornando amplo o campo onde o mesmo se torna aplicável, podendo ser aplicado em praticamente qualquer domínio.

No ciclo de vida do *scrum*, como exemplificado na Figura 3, existem três fases mais fundamentais (RODRÍGUEZ *et al.*, 2019; ABRAHAMSSON *et al.*, 2017), sendo:

1. A fase mais inicial, é a fase onde se dá o planejamento, fase nomeada também como pré-*sprint*, onde os pontos levantados, são os objetivos mais gerais do sistema que se está sendo desenvolvido, esses pontos são todos alinhados, bem como a definição da equipe que fará o projeto, ferramentas, recursos e necessidades são apontadas. Algumas referências na literatura, definem essa etapa como iteração zero (WILLIAMS, 2010).
   - Um *product backlog* é concebido e utilizado para documentar os requisitos do cliente em formato de histórias de usuários, bem como os recursos também (CHOPADE; DHAVASE, 2017). As histórias de usuários são definidas seguindo um padrão

Figura 3 – Diagrama estrutural do *Scrum*.



Fonte: Adaptado de Al-Saqqa *et al.* (2020).

(RODRÍGUEZ *et al.*, 2019), como: como um <*usuário*>, eu quero <*ter*> para <*beneficio*>.

- Os requisitos entram numa fase onde são analisados, e após essa análise é definida as prioridades a cada um, bem como a estimativa do esforço que será associado para sua implementação.
- Onde o *product backlog* está sempre sujeito a atualizações, essas que podem ser contínuas, pois desde as histórias dos usuários criadas de forma incremental, bem como ao longo do processo de desenvolvimento, pois as definições já existentes podem mudar, como as histórias, priorização, etc.

2. Esta fase é conhecida como fase de desenvolvimento ou simplesmente *sprint*, onde a cada término de cada ciclo é gerado um artefato um valor incrementado ao produto final.
   - *Sprints* são como ciclos, que em sua são iterativos, possuindo duração fixa estipulada entre duas a quatro semanas, não podendo extrapolar os limites em termo de tempo.
   - Onde cada *sprint* passa pelas fases mais tradicionais do processo de desenvolvimento, como análise de requisitos, planejamento da mesma com base no *product backlog*, por meio do *design* e por fim determinando a fase de entrega pós, revisão da *sprint*.

É na reunião de planejamento da *sprint* que sempre é realizada ao início de cada uma das *sprints*, onde é decidido tanto a equipe de desenvolvimento quanto quem será o *product owner*. As histórias de usuários que vão sendo desenvolvidas são incrementalmente movidas para o *product backlog*, no que lhe concerne é posto no *product backlog* da *sprint*

tomando por base suas prioridades. Posteriormente são definidos as funcionalidades e recursos a serem implementados. Após isso, se inicia definitivamente uma *sprint*.

Durante a execução de uma *sprint*, um dos objetivos do *scrum master* é isolar a equipe de desenvolvimento das distrações externas a suas devidas atividades da *sprint*, os recursos e funcionalidades com isso vão sendo implementados e testados de forma diária. Diariamente a uma cerimônia chamada *daily*, que basicamente é uma reunião com duração de 15 minutos, realizada em um mesmo local e horário previamente definido e estritamente seguidos, onde o objetivo dessa cerimonia é aprimorar a comunicação entre a equipe, sincronizar as atividades, relatar impendimentos e avanços.

Ao final de cada *sprint* são realizadas duas reuniões, sendo a reunião de revisão da *sprint* em que todos os artefatos gerados em cada uma é posto como um possível incremento despachável para o produto final, onde o mesmo é analisado, avaliado e efetivamente aprovado. Após isso há a reunião de retrospetiva da *sprint*, que basicamente é para discutir os pontos positivos da *sprint* e os pontos de melhoria para as próximas.

Dentro do *scrum*, exitem algumas formas de monitor a *sprint* e seu andamento, como o gráfico de *burn down* utilizado para monitorar o *status* da *sprint*, como seu andamento, seu rendimento e cumprimento das estimativas. E no *scrum* pode haver *sprints* sendo executadas de formas paralelas, onde cada uma das que estão em execução estão construindo uma parte do produto e gerando os incrementos.

3. A última fase, a de encerramento do projeto, é a fase em que todos os requisitos almejados são alcançados. A versão mais recente e estável do produto está pronta para publicação e lançamento. Além de serem disponibilizados uma documentação completa nesta etapa e os manuais do usuário.

2.2.2 Extreme Programming (XP)

Popularmente conhecido como XP, esta metodologia foi proposta por Kent Benk em 1999, um dos primeiros métodos ágeis propostos de forma pública por Beck (2000), onde as motivações encontradas por ele para a criação do XP foi superar as dificuldades encontradas pelos métodos mais convencionais de desenvolvimento de software, que conseguisse estar a frente do processo de requisitos, se adaptasse as mudanças que ocorrem de forma rápida e inesperada, mas que são extremamente necessárias, onde um fundamento base era a criação de uma metodologia adequada para projetos orientados a objetos, que basicamente consiste em ter

vários programas em um mesmo local (CHOUDHARY; RAKESH, 2016).

Onde o XP busca reduzir de forma relevante os altos custos associados as mudanças de requisitos, onde para isso ele substitui os longos ciclos de desenvolvimento por inúmeros ciclos curtos e mais concisos, que por meio dessa estruturação do processo alcançar a satisfação do cliente se torna mais simples e menos burocrático (CHOUDHARY; RAKESH, 2016).

Para melhorar a qualidade dos produtos de software desenvolvidos dentro do XP, o mesmo se baseia em conceitos da engenharia de software e os leva para seu processo os aplicandos em um nível extremo (SCHWABER; BEEDLE, 2002).

O XP se apoia em treze práticas técnicas primárias, sendo as seguintes:

1. Sentar juntos.
2. Toda a equipe.
3. Espaço de trabalho informativo.
4. Trabalhar energizado.
5. Programação em pares (*Pair programming*).
6. Histórias.
7. Iteração curta.
8. *Release* trimestral.
9. Folgas.
10. Construção de dez minutos.
11. Integração contínua.
12. Aceitação e Teste de unidade.
13. *Design* incremental.

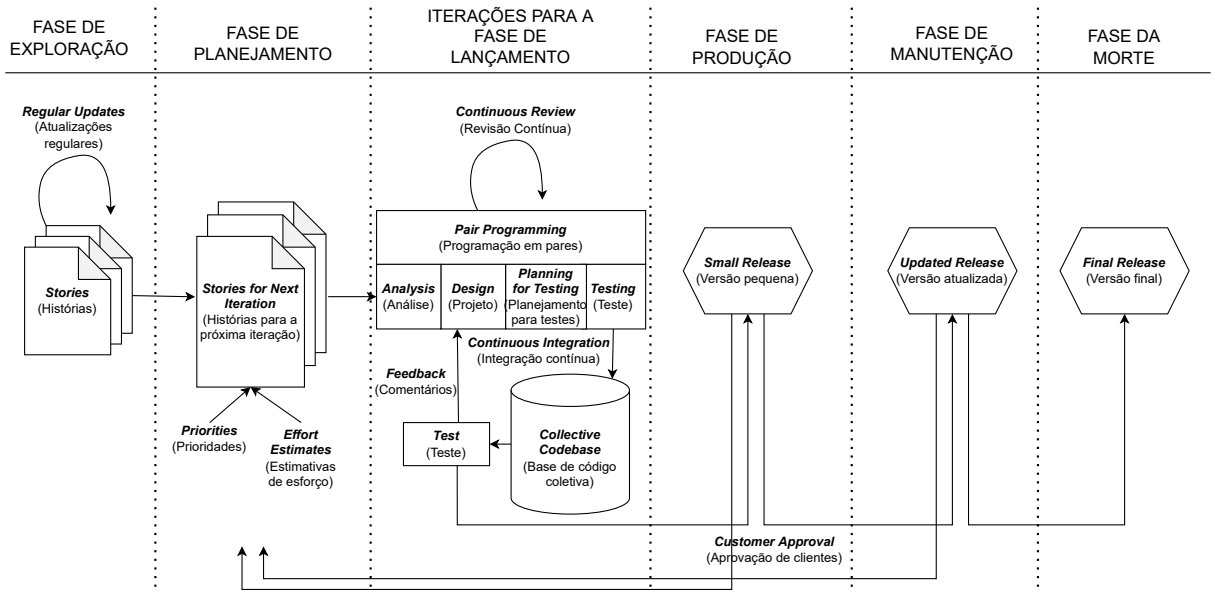
Onde as mesmas foram propostas em adição a onze práticas colorarias (WILLIAMS, 2010), que segue abaixo:

1. Análise da causa raiz.
2. Propriedade coletiva sobre o código.
3. Codificação e Testes.
4. Contrato com escopo negociado.
5. Envolvimento real por parte do cliente.
6. Implantação incremental.
7. Continuidade da equipe.
8. Redução do contingente da equipe.

9. Implantação diária.
10. Base de código única e unificada.
11. Práticas de pagamentos por uso.

Como resultado destas práticas, emergiu do XP cinco valores que são a chave do método: comunicação, simplicidade, *feedback*, coragem e trabalho de qualidade (COHEN *et al.*, 2003). O ciclo de vida do XP consiste na execução de seis fases, como serão descritas e exemplificadas na Figura 4, a seguir:

Figura 4 – Diagrama estrutural do XP.



Fonte: Adaptado de Abrahamsson *et al.* (2017).

1. **Fase de exploração**:
   - O cliente é parte fundamental da equipe dentro do XP, pois ele tem a responsabilidade de tomar as decisões sobre os requisitos e também sobre os recursos transformados em histórias de usuários, baseado no que ele considera ser importante para a primeira versão do produto.
   - A equipe que será responsável pelo projeto é apresentada aos recursos disponíveis para o mesmo, como ferramentas, tecnologias que serão utilizadas para a construção do mesmo, as práticas que serão aplicadas no projeto, para com isso os familiarizar no projeto.
   - Um protótipo de uma porção do sistema é feita em forma de amostra, para com isso testar a tecnologia que será utilizada para a implementação do mesmo e com isso descobrir qual a melhor arquitetura possível para o dado projeto.

   - Duração de algumas semanas a alguns meses.

2. **Fase de planejamento**:
   - Os programadores em conjunto realizam uma estimativa do esforço necessário para a implementação do produto, bem como planejam o cronograma associado a implementação.
   - Ocorre a priorização das histórias.
   - Duração de no máximo alguns dias.
3. **Fase de iteração para lançamento**:
   - Ocorrem várias iterações, onde cada iteração dura de uma a quatro semanas.
   - A primeira iteração, é considera especial, pois o objetivo da mesma é conceber uma arquitetura apropriada para a aplicação tomando como base a seleção de histórias apropriadas.
   - Os testes das funcionalidades na totalidade são planejados pelo cliente, e ao final de todas a iterações os mesmos são executados.
   - Os artefatos da última iteração estarão prontos para irem à produção.

   Onde nesta fase há duas práticas extremamente importantes, sendo a programação em par e a refatoração. Onde na programação em par: são dois desenvolvedores trabalhando na mesma funcionalidade ou parte do código, enquanto um escreve o código, o outro se mantêm observando o processo, onde ele apoia e revisa o código desenvolvido por seu par (SCHMIDT, 2016). Onde os benefícios associados a essa prática são: disseminação do conhecimento sobre o código, desenvolver a propriedade coletiva e responsabilidade comum sobre as linhas de códigos desenvolvidas. A refatoração: apoia a melhoria do produto de software, torna o código mais simples e mais receptivo a receber manutenções, onde é reconstruído o código sem alterar sua funcionalidade final (SCHMIDT, 2016).
4. **Fase de produção**:
   - É realizado uma validação adicional sobre o desempenho do sistema, realizando testes sobre o mesmo, e após isso o mesmo é liberado para o cliente.
   - Mesmo após a primeira versão do produto ser entregue ao cliente, o mesmo deve ser mantido sobre execução e uso até que as novas iterações sejam produzidas e gerem novos artefatos para integração e adição sobre o produto em produção utilizado pelo cliente.
5. **Fase de manutenção**: A chegada de mais colaboradores na equipe pode ser necessária,

para dar vasão e apoio as tarefas elencadas pelo cliente.

6. **Fase de morte**: Chegando nesta fase não há mais histórias de usuários, assim, não há mais alterações solicitadas, assim compreende-se que o estado de implementação atual do produto já satisfaz os requisitos e necessidades do cliente. Ou a morte pode ocorrer quando o produto que está sendo desenvolvido não está fornecendo os resultados esperados para o cliente, ou em última instância se qualquer nova adição ao produto for extremante cara.

2.3 Testes em metodologias ágeis

Desde os princípios históricos envolvendo os campos de estudos ágeis de desenvolvimento, a prática ou atividade de testes é bastante atrelada aos modelos ágeis, onde se considera que os testes é a base do desenvolvimento ágil de produtos de software (HELLMANN *et al.*, 2012).

Compreendendo que a prática da execução das atividades de testes no contexto ágil de desenvolvimento de software, é tida como uma das atividades ou etapas-base para as aplicações das metodologias, onde tal relação vem sendo notada desde os princípios da história da engenharia de software ágil (VANDERBURG, 2005). Onde uma grande maioria de metodologias ágeis confiam e se baseiam sobre os testes para gerarem produtos de software de qualidade e eficazes.

Dada a tamanha difusão dos métodos ágeis de desenvolvimento pelas mais diversas áreas do conhecimento incluindo engenharia de software, bem como a disseminação dos conhecimentos a respeito das práticas e execuções atreladas aos mesmos, algumas terminologias passam praticamente que negligenciadas quando se fala nessa relação entre agilidade e testes, dado que foram difundidas com uma forte associação aos métodos ágeis e não aos testes (HELLMANN *et al.*, 2012).

Como no desenvolvimento ágil de software as coisas são normalmente feitas de forma iterativa e em ciclos, como iterações sobre os requisitos, código, resultados e entregáveis, e como foi dito por Boehm *et al.* (1976) que a melhor forma de medir a qualidade de um produto de software é com a realização de testes sobre o mesmo. Como é executado e como se dá o ciclo de vida de um método ágil de desenvolvimento facilita e sede espaço para a execução das práticas de testes (FERREIRA *et al.*, 2007).

E dentro do presente contexto das metodologias ágeis, maioria dos métodos reservam um espaço em seu ciclo de vida para à aplicação e execução de testes, sejam de unidade, integra-

ção ou sistema. Como os métodos *Extreme Programming* (XP), *Dynamic System Development Model* (DSDM), *Test-Driven Developmentt* (TDD), *Feature Drive Development* (FDD).

2.4 Taxonomia

A palavra taxonomia tem sua origem advinda da língua grega antiga, onde: *Táxis* significa ordem, *Nomos* significa lei, ordem. Onde a origem desta terminologia foi devida a derivação de um dos diversos ramos da biologia, em que o mesmo trata da classificação lógica e científica dos seres vivos, isso fruto do exímio trabalho do médico e botânico sueco Carolus Linnaeus, com usa biologia sistemática (AQUINO *et al.*, 2009).

A classificação dos seres vivos é uma atividade consideravelmente antiga, dado que a mesma tem suas origens na Grécia antiga, e sua forma moderna remonta a mais de 250 anos, que foi quando Linnaeus incutiu tais conhecimentos no meio científico, o conhecimento sobre classificação binomial, conhecimentos esses que vigoram até os dias presentes (GODFRAY, 2002). Mas como esse era um conhecimento extremamente recente e novo, as descobertas advindas do mesmo ainda eram duvidosas, assim Linnaeus subestimou o enorme número de plantas e animais taxonomizados segundo Godfray (2002).

Dado a difusão sobre o tema, um maior volume trabalhos taxonômicos foram aparecendo de formas subsequentes mutualmente, onde mais e mais espécies eram descritas, geralmente ignorando outros trabalhos relacionados ou no mesmo campo de atuação, com isso houve um grande caos, advindo dessa avalanche de resultados e taxonomizações que vinham sendo propostas, ameaçando destruir todos os resultados anteriores já obtidos. Usando então as ferramentas disponíveis, os taxonomistas ainda no século XIX resolveram estipular um conjunto complexo de regras que determinava como uma espécie deve ser nomeada e associada a alguma (espécime) ou tipo, com a criação desse conjunto de regras a crise foi controlada e sanada (GODFRAY, 2002).

Dentre alguns fundamentos por trás das taxonomias, um extremamente importante é que os taxonomistas necessitam de um objetivo claro e alcançável, que também seja realista e relevante (GODFRAY, 2002).

Existem dois principais modelos de taxonomias, em que o primeiro modelo basicamente é o que não reside somente em uma única publicação ou em uma única instituição, a literatura é unificada fazendo referências cruzadas a mais de duas fontes colaborativas. Já o segundo modelo é de uma organização mais unitária onde o conhecimento é centralizado é um

único local (GODFRAY, 2002).

Como foi exemplificado por Madampe *et al.* (2021), os passos para se criarem uma taxonomia giram em torno dos seguintes:

- Revisão na literatura, seja: mapeamento sistemático, revisão sistemática ou outros. Ou uma observação empírica aprofundada e aprimorada sobre o objetivo de estudo.
- Leitura dos trabalhos revisados na totalidade ou compreensão sobre os elementos observados empiricamente.
- Codificação e agrupamento de ideias de mesmo sentido nos trabalhos lidos ou de comportamentos observados, ou categorias coletadas empiricamente.
- Revisão dos termos e codificações obtidos no passo anterior.
- Proposta da taxonomia final sobre as codificações revisadas.

Tendo em mente tal exemplificação, uma segunda proposta para a criação de taxonomias e os passos que devem ser seguidos para criar a mesma, foi proposta por Ralph (2018), uma ótica que gira mais em torno dos seguintes passos de execução:

- Selecione uma estratégia, dado que teorias e taxonomia podem ser feitas de diversas formas.
- Realização de estudos secundários, dado que uma taxonomia pode ser a sintetização combinada de revisões, mapeamentos e sínteses.
- Teoria fundamentada e estudos de casos interpretativos, dado que um estudo de caso parte de uma investigação empírica.
- Estudos primários de única fonte, assim inclui: questionários, grupos focais, entrevistas, mineração de repositórios entre outros.
- Experiência pessoal, sintetizar anos de experiência através de uma reflexão cuidadosa.
- Seleção do local, onde se define um local para estudo, como, por exemplo: versionadores de códigos para busca de *code smells*.
- Realizar a coleta dos dados, por observação direta, entrevistas entre outras.
- Perguntas orientadoras, definição de questões para guiarem a coleta dos dados.
- Análise dos dados, realizar análises cíclicas, onde cada iteração retorna um resultado preliminar.
- Avaliação conceitual, gerar a teoria ou taxonomia e a revisar em par.

2.5 Taxonomia no contexto da engenharia de software

Ultimamente a engenharia de software está cada vez mais preocupada com a sua teoria base de conhecimentos, pois o conhecimento fundamental que se compreende por meio das teorias mais recentes e relevantes, fornecem um contra ponto rígido ao conhecimento prático expresso mediante técnicas e métodos mais antigos e consolidados (RALPH, 2018).

Segundo Ralph (2018) na engenharia de software há bastante conteúdo disponível para gerar e avaliar teorias que expliquem porque certas coisas ocorrem, as conhecidas teorias de variância. Mas a uma restrita quantidade de conteúdos que exemplifiquem como certas coisas ocorrem, como, por exemplo: teorias de processo, teorias para analisar e compreender situações, ou melhor dizendo taxonomias.

Compreendendo que toda taxonomia é um estudo que possui o intuito de classificar, e que toda taxonomia gera uma classificação (GODFRAY, 2002). E tendo o entendimento que pesquisas em engenharia de software geralmente geram taxonomias, pois dado que a mesma se concentra em classificação de funções, de fatores de sucesso, de dimensionalidade a respeito da qualidade de software (RALPH, 2018).

Mas dada essa dimensionalidade e conjuntabilidade entre taxonomia e engenharia de software, a comunidade foi desenvolvendo orientações metodológicas com foco em gerar e avaliar teoria de variância, e pouca orientação voltada a teorização dos processos ou as taxonomias foi de fato difundida (WOHLIN *et al.*, 2012).

Diante das atuais injunções que envolvem a engenharia de software, da diversa gama de conhecimento que vem sendo produzida semestralmente, a necessidade da existência ou criação de terminologias e definições mais precisar, acessíveis e simples em termo de acesso, vem se tornando cada vez mais necessária e importante (ŠMITE *et al.*, 2014).

Assim, ultimamente há um forte interesse da engenharia de software em taxonomias, dado no atual momento pouquíssimas taxonomias são propostas, e muitas possuem uma estruturação de decisão e classificação com métodos não tão bem descritos e motivados (USMAN *et al.*, 2017). Assim, a engenharia de software busca e incentiva a criação e proposição de mais taxonomias ao invés de simples catálogos empíricos, como foi dito por Ralph (2018).

No contexto da engenharia de software, atualmente têm-se algumas taxonomias criadas, como: *Bloom's taxonomy in software engineering education: A systematic mapping study* proposta por Britto e Usman (2015), que consiste em uma taxonomia para ensino de engenharia de software, *A taxonomy of software engineering challenges for machine learning*

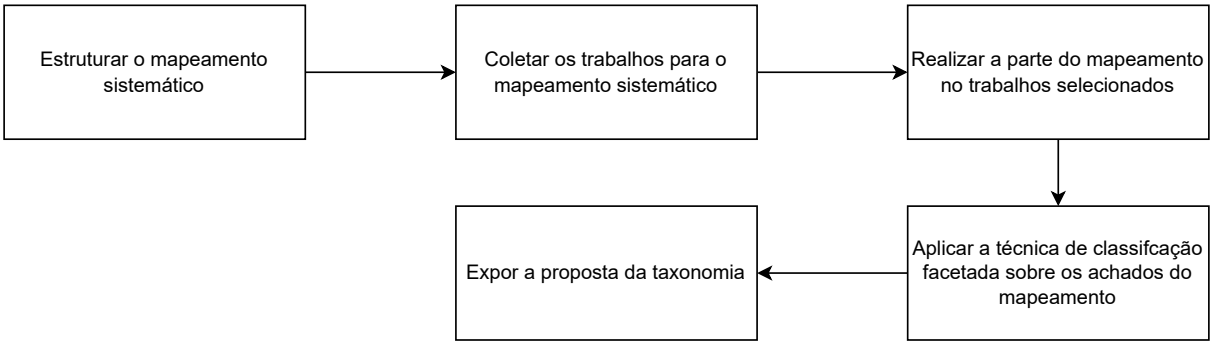
systems: An empirical investigation proposta por Lwakatare *et al.* (2019), que taxonomiza desafios da engenharia de software a respeito de sistema de aprendizado de máquina, *A taxonomy of software types to facilitate search and evidence-based software engineering* que foi elaborado por Forward e Lethbridge (2008), que basicamente expôs uma taxonomia de tipos de software, *A Faceted Taxonomy of Requirements Changes in Agile Contexts* proposta por Madampe *et al.* (2021), onde o mesmo taxonomiza mudanças em requisitos de software, *Towards a taxonomy of code review smells* proposta por Doğan e Tüzün (2022), onde o mesmo exibe uma taxonomia de *smells* em revisão de código.

Dado que a classificação de objetos de estudo de um dado campo do conhecimento, fornece terminologias comuns, que no que lhe concerne facilitam o compartilhamento do conhecimento entre a área (VEGAS *et al.*, 2009; VESSEY *et al.*, 2005; WOHLIN, 2014).

3 PROCEDIMENTOS METODOLÓGICOS

Neste capítulo são apresentadas as etapas necessárias para a realização do presente trabalho. A Figura 5 mostra uma visão geral do sequenciamento entre etapas e atividades que foram executadas, e as subseções que serão descritas a seguir, irão detalhar cada uma destas etapas.

Figura 5 – Passos para a realização do presente trabalho.



Fonte: Elaborado pelo autor.

3.1 Estruturar o mapeamento sistemático

Nesta etapa de execução do presente trabalho, será dado prosseguimento a estruturação do mapeamento sistemático que será executado no presente trabalho, onde para a execução do mesmo, foram selecionadas quatro grandes bibliotecas, sendo: *IEEE Xplore*, *ACM Digital Library*, *Science Direct* e *Springer Link*. E como auxílio na execução do mapeamento, será feito o uso da ferramenta *Parsifal*, onde a mesma é uma ferramenta para gerenciamento e condução tanto de mapeamentos quanto revisões sistemáticas.

O mapeamento será executado com base em uma *string* pré-definida, onde as pesquisas nas bibliotecas serão executadas com base na mesma, mas o motor de busca de cada biblioteca funcionada de forma independente as demais, assim cada motor indexa as informações referentes as pesquisas de forma diferente, dado que essa indexação é fechada no domínio de cada biblioteca.

Mas compreendendo esse cenário das várias formas de indexação adotadas por cada biblioteca, foram definidos alguns filtros dentro das mesmas, para tentar tornar a amostra o mais semelhante e concisa possível entre as quatro bibliotecas, dado que de forma inicial, notava-se muito retorno de áreas que não relacionadas ao tema do referido trabalho.

3.1.1 Filtros aplicados em cada biblioteca

O escopo da *string* foi mantido em todas as bibliotecas, com a variação dos filtros aplicados em cada uma. As únicas bibliotecas em que foi notado a necessidade de aplicar os filtros, foi a *Science Direct* e *Springer Link*, assim o filtro aplicado em cada uma está exposto abaixo:

Filtro aplicado na *Science Direct*:

1. Selecionado um *checkbox* em *Subject areas*, *checkbox* de valor *Computer Science*.

Filtro aplicado na *Springer Link*:

1. Selecionado dentro de *Content Type* os valores *Chapter* e *Conference paper*.
2. Selecionado dentro de *Discipline* o valor *Computer Science*.

Para se obter maior volume de trabalhos catalogados na etapa seguinte, e que esse volume possua uma qualidade com relação ao tema do presente trabalho, foi definido que a *strings* será toda no idioma inglês, dado que o motor de busca das bibliotecas selecionadas, trabalham melhor com buscas neste idioma, assim tal escolha traz ao presente trabalho uma garantia de que os trabalhos que serão catalogados e posto na coletânea do mapeamento, sejam contribuições mais relevantes e possuam maior relação com o projeto proposto.

Inicialmente a *string* definida foi a seguinte:

*(“agile development” **OR** “agile practices”) **AND** (“test process” **OR** “software testing”).*

E a forma definida para a execução do mapeamento, é que na fase de triagem mais inicial dos trabalhos, os mesmos serão coletados com base nos retornos da *string*, após isso o que será avaliado em cada um, é o título e o *abstract* do mesmo, caso algum não tenha o *abstract* será avaliado o resumo e em última instância a introdução do trabalho, e por fim uma análise mais geral dos trabalhos, assim um trabalho entra no mapeamento caso os pontos citados acima levem a compreender que o mesmo é uma contribuição relevante no domínio de pesquisa do projeto, segundo a ótica do autor do presente trabalho.

3.2 Coletar os trabalhos para o mapeamento sistemático

Nesta etapa, é onde se concentra a parte da catalogação dos trabalhos que entrarão no mapeamento e serão posteriormente analisados de forma mais concisa, mas o que vai ser feito nesta etapa é a execução da *string* definida na seção passada dentro das bibliotecas, a aplicação

dos filtros de cada biblioteca, posterior a isso, a catalogação dos trabalhos, onde será feito a obtenção do Bibtex de cada trabalho, e esse Bibtex será inserido no *Parsifal*, a ferramenta de suporte a execução de mapeamentos utilizada pelo presente trabalho.

Ao fim desta etapa, o Bibtex de todos os trabalhos retornados pela *string* nas bibliotecas, terá sido coletado e estarão todos agrupados na ferramenta, e posterior a isso poderá ser dado prosseguimento a execução do mapeamento sistemático do presente trabalho.

3.3 Realizar a parte do mapeamento nos trabalhos selecionados

Nesta presente etapa do presente trabalho, é onde se concentra o foco do projeto, dado que é nesta etapa que será executada a avaliação e coleta dos dados dos trabalhos selecionados conforme notado um ponto de contribuição positivo do trabalho selecionado para com o presente trabalho.

Como a qual vai ser realizada essa parte, é uma avaliação feita em par. Para que ao final, a avaliação tenha no mínimo uma revisão. A forma pela qual foi escolhida para a leitura dos trabalhos é *top-down*, para dar foco a coleta de dados e na forma cronológica como as mesmas vão aparecer no decorrer da leitura dos trabalhos.

A sequência lógica de aplicação de conhecimentos e fundamentos sobre as etapas de seleção dos trabalhos, serão dados da seguinte forma, a primeira sendo a remoção de trabalhos duplicados, atividade executada totalmente pela ferramenta *Parsifal*, a segunda é a triagem dos trabalhos com base no título e *abstract*, e após essa etapa é executada uma revisão em par sobre o resultado, em seguida uma terceira etapa de triagem, com foco em uma leitura geral dos trabalhos, mas sem tanto aprofundamento ainda, e após essa etapa, é executada uma segunda revisão sobre os achados, e após essa última revisão, se tem os trabalhos elencados para a extração de conhecimento para a construção da taxonomia.

3.4 Aplicar a técnica de classificação facetada sobre os achados do mapeamento

Nesta referida etapa, é onde serão pegos os dados e compreensões resultantes da execução da etapa anterior, e sobre os mesmos vai ser aplicado a técnica de classificação facetada, sendo uma técnica que se põe como uma das formas de se elaborar uma taxonomia. Isso será feito com base no que foi feito por Madampe *et al.* (2021) em seu trabalho elencado na Seção 4 de trabalhos relacionados ao presente trabalho.

Onde vai ser executada uma leitura sobre os trabalhos selecionados, e os trechos que forem se adequando a mesmo grupo de ideias ou princípios, serão assim agrupados, e esse tema geral que representa os diversos trechos será posto como termo definidor ou definição. Sendo assim essa definição um resultado de uma classificação sistemática do conhecimento, onde a mesma recorre a categorias semânticas, que podem ser mais gerais ou específicas do assunto, que ao final são combinadas entre si, para com isso gerar uma entrada que irá resultar em uma saída de classificação completa.

3.5 Expor a proposta da taxonomia

Chegando nesta etapa, o presente trabalho visa ter a taxonomia pronta para ser proposta e exposta, dado que todos os passos anteriores foram executados, gerando em cada um, artefatos que serviram como dados de entrada para a etapa seguinte ao mesmo, e ao fim desse processo onde todas as etapas foram concluídas, a taxonomia estará elaborada.

4 TRABALHOS RELACIONADOS

Foram identificados na literatura alguns estudos que se relacionam com este projeto de pesquisa tanto com relação à criação de taxonomias quanto com relação à agilidade no desenvolvimento de software. Nesta seção estes estudos são apresentados e suas contribuições e o modo como se relacionam com o presente trabalho são descritos.

4.1 *Towards a taxonomy of code review smells*

Doğan e Tüzün (2022) realizaram um estudo para fornecer uma compreensão mais estruturada e empírica a respeito das más práticas seguidas em processos e procedimentos de revisão de código (*code review*) que ocorrem durante o ciclo de desenvolvimento de um produto. Os autores então propuseram uma taxonomia com sete classes de definição e agrupamento, onde a mesma classifica o *smell*, expõe uma definição para o *smell*, as possíveis causas raízes para o mesmo e seus potenciais efeitos colateiras sobre o processo de *code review*. Para a elaboração da taxonomia, os mesmos realizaram uma revisão na literatura, para com isso reunirem as más práticas relacionadas a revisão de código que foram e estavam sendo discutidas na literatura. Posteriormente realizaram uma pesquisa com um total de 32 profissionais de software experientes e realizaram entrevistas com especialistas para a obtenção das opiniões dos mesmos.

Fazendo utilização dos resultados do trabalho de MacLeod *et al.* (2017) onde eles elencaram as melhores práticas de revisão de código bem como os desafios atrelados a essa prática, eles puderam elaborar e propor a taxonomia de maus cheiros em revisões de códigos, para validar tal proposta, foi analisado mais de 226.292 revisões de códigos coletadas de um total de 8 projetos de código aberto (OSS). Como processo de validação observaram que existe um total considerável de maus cheiros em revisões de códigos, independente da escala e tamanho do projeto, onde os resultados empíricos mostraram que um total de 72,2% das revisões são afetadas por pelo menos um mau cheiro.

Neste presente trabalho, será desenvolvida uma taxonomia com base em um mapeamento sistemático de trabalhos da literatura atual e relevante para com o tema, abordando somente a literatura branca como foi feito por MacLeod *et al.* (2017) no trabalho exposto acima.

4.2 *A faceted taxonomy of requirements changes in agile contexts*

Madampe *et al.* (2021) propuseram um estudo onde buscavam entender o melhor possível as mudanças de requisitos no contexto do desenvolvimento ágil, e para tal propuseram uma taxonomia de mudanças de requisitos no contexto ágil de desenvolvimento. Foi executada uma abordagem com métodos mistos compreendidos sobre uma série de estudos: um estudo desses foi baseado em entrevistas, feita com um total de 10 participantes dispersos na Nova Zelândia e Austrália, outro estudo baseado em revisão da literatura e um terceiro com foco em uma pesquisa aprofundada com um total de 40 participantes pelo mundo todo.

Como resultados do trabalho, obtiveram que diversas características das mudanças de requisitos no contexto ágil encontradas, é relacionada a diferentes tipos e formas de relacionamentos, mas, em suma, compreenderam que o surgimento está ligada diretamente a uma grande variedade de eventos. Para propor a taxonomia, utilizaram como base o View (1990), um glossário padrão da IEEE para engenharia de software, que apresenta diversas definições, sendo uma delas, a definição do que são requisitos. A taxonomia que os autores elaboraram e apresentaram fornece um guia para os profissionais das áreas da engenharia de software que desejam gerenciar problemas relacionados a mudança de requisitos nos contextos ágeis de desenvolvimento.

Neste trabalho, será seguida a abordagem utilizada por Madampe *et al.* (2021) para construir e elaborar a taxonomia, dando enfoque que no presente trabalho será feito o uso das partes que se referem ao uso das classificações facetadas.

4.3 *An empirical characterization of bad practices in continuous integration*

Zampetti *et al.* (2020) produziram um estudo, a respeito da caracterização empírica de más práticas em integração contínua. Dado que a integração contínua (CI) veio para introduzir vários pontos benéficos no desenvolvimento de um produto de software, pontos esses como: alta qualidade, confiabilidade, escalabilidade. Contudo, trabalhos publicados recentemente apontam haver barreiras e más práticas em torno da adoção da prática. O trabalho foi realizado por meio de entrevistas semiestruturadas com um total de 13 especialistas, e com mineração em mais de 2.300 postagens (*posts*) na plataforma/fórum *Stack Overflow*. Como resultado, foi compilado um catálogo com um total de 79 maus cheiros em CI, nos quais os mesmos pertencem a um total de 7 categorias relacionadas a diversas dimensões, como: *pipeline*, agenciamento, processos, etc.

Alguns entendimentos dos resultados do trabalho confirmam as afirmações contidas

nas literaturas existentes, mas, o que o estudo também destaca, é que novas más práticas foram descobertas/catalogadas, como, por exemplo: relacionadas a ferramentas de análise estática, abuso de *scripts shell*.

O trabalho exposto acima, se relaciona com o presente trabalho, no âmbito de que dentro do que está previsto nas ameaças a validade do trabalho, no cenário em que não se obteve insumos para a proposta da taxonomia, o presente trabalho vai fazer uso dos métodos de categorização empírica utilizados por Zampetti *et al.* (2020), para propor uma categorização empírica dentro do tema do referido trabalho.

4.4 *Mapping software testing practice with software testing research — SERP-test taxonomy*

Engström e Petersen (2015) conceberam um estudo, referente a um mapeamento de práticas de teste de software, onde os mesmos a nomearam de taxonomia de teste SERP, dado que SERP é uma das metodologias abordadas pela dupla na construção do trabalho, que foca em coleta de trabalhos com base no retorno do mecanismo de pesquisa, onde se aproveita os trabalhos que vem listados por primeiro segundo os critérios do buscador. Foi executado um mapeamento sistemático da literatura sobre os procedimentos SERP de coleta de trabalhos, onde o foco se deteu sobre a coleta de trabalhos e esforços anteriores focados na construção de taxonomias dentro da engenharia de software. Por meio da análise puderam realizar a triagem entre trabalhos que focavam suas taxonomias tanto no contexto acadêmico quando no industrial.

O trabalho obteve inúmeros entendimentos a respeito das práticas de testes, mas, o que foi focado durante toda a escrita do trabalho, e é o que se apresenta como ponto mais importante da taxonomia que propuseram, é que a prática/aplicação de testes de software, variam bastante, dado o contexto no qual estão incutidos, assim a veracidade varia de baixa aplicação a extrema aplicação, dado que há contextos mais simplórias e também mais complexos.

O trabalho supracitado, se relaciona com o presente trabalho, no âmbito de que ambos contribuem de forma positiva para a engenharia de software, no que se refere a propostas de taxonomias referente a testes de software, onde o presente trabalho vai recorrer à análise de contextos utilizadas por Engström e Petersen (2015), para propor uma taxonomia que além de prover entendimento sobre o tema, seja minimamente direcionada a aplicação por contextos.

4.5 Análise comparativa dos trabalhos relacionados

O Quadro 1 compara o presente trabalho com os 4 trabalhos relacionados descritos anteriormente em 5 pontos principais, sendo estes: a aplicação de mapeamento sistemático sobre a literatura, análise de práticas de testes no contexto de desenvolvimento de software ágil, utilização da técnica de classificações facetadas para propor a taxonomia e a proposta de uma taxonomia ao invés de um catálogo com bases empíricas.

Quadro 1 – Comparativo entre os trabalhos relacionados e o trabalho proposto.

Trabalhos	**Mapeamento sistemático na literatura branca**	**Testes Ágeis**	**Aplicação da técnica de classificações facetadas**	**Cria uma taxonomia ao invés de catálogos empíricos**
Trabalho Proposto	Sim	Sim	Sim	Sim
Doğan e Tüzün (2022)	Sim	Não	Não	Sim
Madampe *et al.* (2021)	Sim	Não	Sim	Sim
Zampetti *et al.* (2020)	Sim	Não	Não	Não
Engström e Petersen (2015)	Sim	Sim	Não	Sim

Fonte: Elaborado pelo autor.

5 MAPEAMENTO SISTEMÁTICO DE PRÁTICAS DE TESTES ÁGEIS

Este capítulo apresenta o mapeamento sistemático realizado para práticas de testes em metodologias ágeis.

5.1 Protocolo do mapeamento sistemático

O objetivo do mapeamento é baseado em investigar sobre a literatura as práticas de testes de software no contexto ágil de desenvolvimento. baseando-se em pesquisas por trabalhos atuais e relevantes no domínio, nas principais bibliotecas do mundo, onde foram selecionadas quatro bibliotecas.

Onde as mesmas foram selecionadas com base em alguns critérios, que foi com relação à relevância e visibilidade dos trabalhos contidos nas mesmas, a facilidade de acesso às mesmas, a simplicidade de acesso e uso, com base nisso, as bibliotecas selecionadas foram: *IEEE Xplore*, *ACM Digital Library*, *Science Direct* e *Springer Link*, como foi apresentado na seção 3.1 do referido trabalho.

Para isso foram definidas um total de quatro questões de pesquisa, onde as seguintes questões de pesquisa (QPs) que foram elencadas para a execução do mapeamento detalhadas a seguir:

- **QP1**. Quais as técnicas de testes de software, que estão sendo empregadas no contexto de desenvolvimento ágil? Nesta questão de pesquisa, busca-se entender quais as técnicas de testes estão sendo aplicados dentro do contexto dos trabalhos selecionados no mapeamento, como, por exemplo: (i) Testes caixa-branca; (ii) Testes caixa-preta; (iii) Testes caixa-cinza.
- **QP2**. Quais as etapas de testes de software, que estão sendo abordadas e executadas na prática, dentro do contexto ágil de desenvolvimento? Nesta questão de pesquisa, tem o objetivo de entender quais as etapas de testes de software, que estão sendo colocadas em prática nos trabalhos selecionados pelo mapeamento, como, por exemplo: (i) Teste de unidade; (ii) Teste de integração; (iii) Testes de sistema; (iv) Testes de aceitação; (v) Testes de usabilidade; (vi) Testes de regressão; (vii) Testes de carga.
- **QP3**. Quais as ferramentas de teste de software, que comumente estão sendo empregadas em conjunto com as técnicas e etapas de teste, como suporte e apoio a execução dos testes, no cenário ágil de desenvolvimento de softwares? Na seguinte questão de pesquisa, busca-se compreender quais ferramentas que estão sendo mais aplicadas para garantir

e dar suporte a prática e execução da testagem dos softwares desenvolvidos de forma ágil, seja em pequena ou larga escala, como, por exemplo: (i) *JMeter*, uma ferramenta totalmente implementada sobre a linguagem *Java*, que fornece suporte a execução de testes de carga, performance, escalabilidade, robustez; (ii) *Cypress*, uma ferramenta totalmente implementada sobre a linguagem *Javascript* que permite e facilita a realização de testes tanto de interface, quanto de usabilidade em páginas web; (iii) *CircleCI*, ferramenta que automatiza a execução de testes em esteira, sejam unitários, de integração, *rollbakc* e etc.

– **QP4**. Quais os modelos de testes vem sendo comumente adotados dentro dos cenários ágeis de desenvolvimento de software? Com esta questão de pesquisa, tenta-se obter o entendimento a respeito dos de testes que estão sendo empregados na modelagem e escrita dos cenários ou fases de teste, como: (i) Testes A/B; (ii) Testes baseados em cenários; (iii) Testes baseados em diagramas de sequência; (iv) Testes baseados em protótipos; (v) Testes baseados em cobertura.
– **QP5**. Quais as metodologias de testes de softwares que ultimamente vem sendo adotadas pelos cenários ágeis de desenvolvimento de software e aplicações? Nesta dada questão de pesquisa, busca-se compreender quais metodologias de teste de software que estão sendo aplicadas pelos trabalhos selecionados no mapeamento, como, por exemplo: (i) **TDD** *Test-Driven Development*; (ii) **BDD** *Behavior Driven Development*; (iii) **ATDD** *Acceptance test-driven development*; (iv) **FDD** *Feature Driven Development*; (v) **DDD** *Domain-Driven Design*.

Baseado nas questões de pequisa citadas acima, a intervenção do presente mapeamento vai se dar em alguns contextos e aspectos, considerados relevantes e de uma importância elevada, onde a intervenção gira em torno de termos como testes de software, contexto ágil de desenvolvimento, testes, prática de testes, modelos de testes, entre outros. Para isso, foi realizado um PICOC, que basicamente visa verificar cinco elementos de uma pergunta, para com isso validar se uma pergunta realmente é pesquisável, onde as questões do mesmo estão representadas no Quatro 2.

A *string* de busca aplicada nas bibliotecas foi a seguinte: *("agile development"* ***OR*** *"agile practices")* ***AND*** *("test proceess"* ***OR*** *"software testing").* Como já foi especificado e descrito na Subseção 3.1.1. O período de coleta nas bibliotecas, se deu do dia 21/09/2022 até 25/09/2022.

Segue no Quadro 3, a relação de trabalhos inicialmente elencados em cada biblioteca,

Quadro 2 – PICOC definido para o mapeamento.

Índice	**Questões de pesquisa**
População	Artigos, trabalhos e contribuições que sejam relevantes publicas nas maiores bibliotecas acadêmicas do mundo.
Intervenção	Testes de software, contexto ágil de desenvolvimento, testes, prática de testes, modelos de testes.
Comparação	Conceitos, lacunas de pesquisa.
Resultado	Uma visão mais estruturada e organizada sobre as práticas de testes de software no contexto ágil de desenvolvimento por meio de uma taxonomia.
Contexto	Locais em que se deseje aplicar ou compreender testes no contexto ágil.

Fonte: Elaborado pelo autor.

dado que foi performada a mesma *string* em todas, com a diferença que duas das mesmas tiveram alguns filtros aplicados como foi exposto na subseção 3.1.1:

Quadro 3 – Trabalhos mapeados por cada biblioteca.

Bibliotecas	**Quantidade de trabalhos**
ACM Digital Library	336
Science Direct	277
Springer Link	257
IEEE Xplore	121

Fonte: Elaborado pelo autor.

Os critérios de seleção definidos para a execução do presente mapeamento tem seu foco na inclusão dos trabalhos, onde foram elencados e definidos um conjunto de critérios de inclusão, que basicamente define o que um trabalho tem que ter para fazer parte do mapeamento, sendo os seguintes critérios de inclusão:

- Abordar temáticas relacionadas a desenvolvimento ágil, práticas ágeis.
- Abranger, dentro de suas temáticas, temas como: processos de teste, teste de software.
- Responder a pelo menos uma das questões de pesquisa definidas para o estudo.

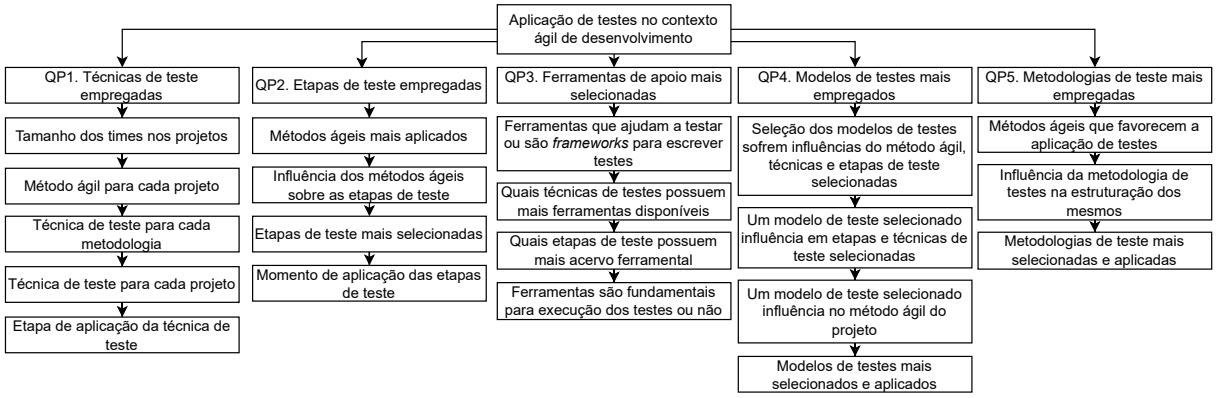
E os critérios de exclusão, foram definidos como exclusos, que façam com quem um trabalho não se enquadre no mapeamento, sendo os seguintes listados:

- Artigo no qual não obtivemos acesso.
- Artigos com foco de estudo que não o específico do presente trabalho.
- Artigos duplicados.
- Artigos que não estejam na língua inglesa.
- Artigos com menos de três páginas.

– Livros serão desconsiderados.
– Artigos que foquem somente em agilidade e não abordem a parte de testes.
– Artigos/trabalhos que sejam basicamente re-escrita de livros.
– Artigos/trabalhos que necessitem de aquisição financeira para obter acesso aos mesmos.

A Figura 6, apresenta uma visão geral a respeito das definições, selecionadas para a realização da extração dos dados por meio dos trabalhos previamente selecionados por meio da execução do mapeamento sistemático executado pelo presente trabalho.

Figura 6 – Diagrama de extração de dados do mapeamento.



Fonte: Elaborado pelo autor.

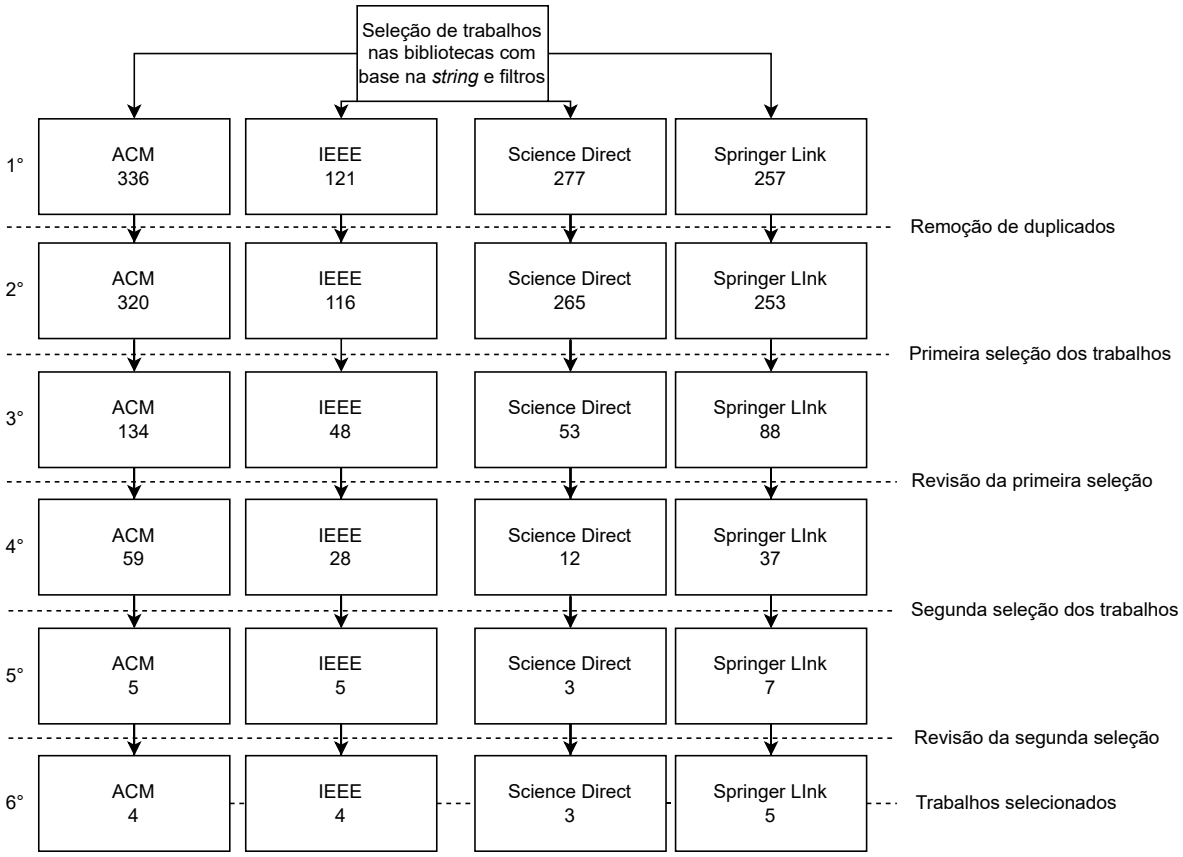
5.2 Execução do mapeamento sistemático

Na Figura 7, segue a ilustração do passo a passo de execução do mapeamento sistemático executado no presente trabalho, exemplificando todas as fases desde a coleta de trabalhos com base na *string* definida em cada biblioteca, até a última revisão sobre os trabalhos elencados.

Após a seleção dos trabalhos nas quatro bibliotecas definidas para o mapeamento, com base no retorno da *string* definida para as mesmas, bem como em conjunto com a aplicação de alguns filtros que tinham como objetivo uma busca por trabalhos que apresentassem maior qualidade perante ao que se espera pelo presente trabalho, filtros esses definidos na Subseção 3.1.1.

Onde foi iniciada a primeira etapa da execução do mapeamento, etapa marcada como (1°) na Figura 7, onde foram coletados todos os trabalhos retornados por cada uma das bibliotecas. Com a observação que até esta etapa, não foi realizada nenhuma curadoria sobre o conteúdo dos trabalhos, pois o critério foi somente o retorno da *string* nas bibliotecas. Posterior

Figura 7 – Execução do mapeamento do presente trabalho.



Fonte: Elaborado pelo autor.

a essa primeira fase da primeira etapa, foi realizada a remoção de trabalhos duplicados, etapa essa executada totalmente de forma automática pela ferramenta *Parsifal*, e após essa remoção de duplicatas, houve o *input* de trabalhos para a próxima etapa do mapeamento.

Após conclusão da primeira etapa da execução do mapeamento, foi dado início a segunda, a qual está marcada na Figura 7, como (2°), onde nesta etapa já foi iniciada a leitura e análise de conteúdo dos trabalhos selecionados, onde nesta etapa o que foi considerado foram os títulos e *abstract* dos trabalhos, em última instância, quando o trabalho não possuía *abstract*, foi feito a leitura de sua introdução, e os critérios aplicados nesta etapa para o trabalho passar ou não por esta etapa, foram os definidos ao final da Seção 5.1, após essa primeira fase da segunda etapa, foi realizada a primeira revisão do mapeamento, com foco em que a mesma foi realizada em par, onde foram revisados os trabalhos selecionados, visando buscar algum trabalho que tenha passado pelos critérios, mas que não apresentava uma qualidade conforme o esperado pelo presente trabalho, e ao fim desta revisão, foi concluída a terceira (3°) etapa do mapeamento, com isso tinha-se o *input* de dados/trabalhos para o início da quarta etapa do mapeamento.

Na quarta (4°) etapa do mapeamento, foi realizada uma leitura por todos os trabalhos

que haviam sido selecionados até essa etapa do mapeamento, com a observação que não foi uma leitura final, mas sim, uma leitura com teor médio de aprofundamento no conteúdo, já avaliando o teor estrutural, contribuição, bem como o resultado e possíveis dados que agregariam valor para o presente trabalho, ao fim dessa triagem, foi executada a segunda revisão do mapeamento, novamente uma revisão em par, sobre os trabalhos resultantes das aplicações dos critérios, e ao fim desta segunda e última revisão, que se compreende a quinta (5°) etapa, presente na Figura 7, foi obtido os trabalhos selecionados, fase essa que se compreende como a sexta (6°) e última presente na Figura 7.

Ao fim, o total de trabalhos selecionados por cada biblioteca está descrito no Quadro 4, logo abaixo:

Quadro 4 – Trabalhos finais mapeados por cada biblioteca.

Bibliotecas	**Quantidade de trabalhos**
Springer Link	5
ACM Digital Library	4
IEEE Xplore	4
Science Direct	3

Fonte: Elaborado pelo autor.

Para uma consulta mais clara de quais são os trabalhos selecionados em cada uma das bibliotecas, como foi exposto o total de trabalhos por bibliotecas no Quadro 4, supracitado, foi elaborado o Quadro 5, que está exposto logo abaixo, onde o mesmo contém as informações referente a cada um dos trabalhos selecionados por meio do mapeamento sistemático, executado pelo presente trabalho, informações essas bem como um identificador único dentro do presente trabalho, nomeado de *id* na estrutura do quadro, que será utilizado para identificar o referido trabalho nos textos que seguirão dispostos mais abaixo, bem como informa o título, a referência direta do mesmo e por último, mas não menos importante, a biblioteca de onde foi extraído o mesmo.

5.3 Análise geral dos dados do mapeamento sistemático

Ao todo, desde o início do mapeamento foram analisados 991 trabalhos, sobre esse total, pode-se compreender melhor a distribuição de trabalhos por cada biblioteca com base na Figura 8, que está disposta logo abaixo, ou por meio do Quadro 3:

Para se obter uma visão melhor da proporção final de trabalhos selecionados por

Quadro 5 – Conjunto final dos trabalhos selecionados pelo mapeamento.

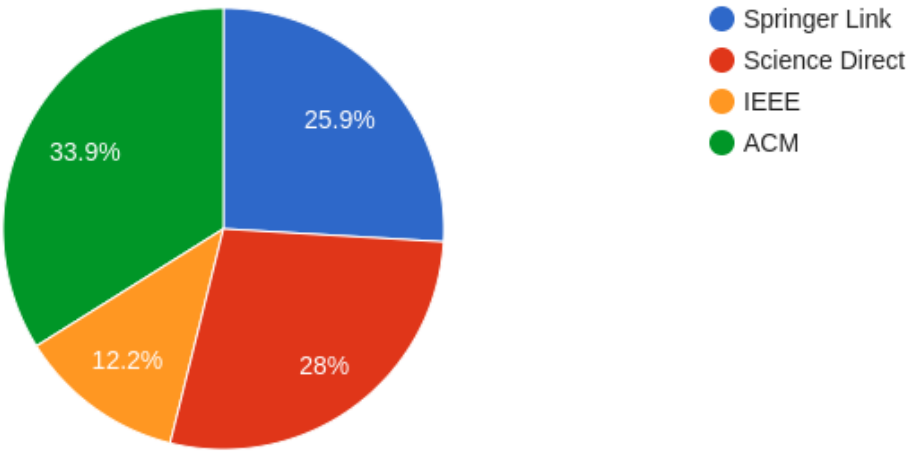
ID	**Título**	**Referência**	**Biblioteca**
E1	*When agile meets OO testing: a case study*	(GOESCHL *et al.*, 2010)	ACM
E2	*Exploring influences on student adherence to test-driven development*	(BUFFARDI; EDWARDS, 2012)	ACM
E3	*A study on agility and testing processes in software organizations*	(KETTUNEN *et al.*, 2010)	ACM
E4	*Results from an Ethnographically-informed Study in the Context of Test Driven Development*	(ROMANO *et al.*, 2016)	ACM
E5	*A Comparative Study of Agile Methods, Testing Challenges, Solutions e Tool Support*	(REHMAN *et al.*, 2020)	IEEE
E6	*Strategies for Agile Software Testing Automation: An Industrial Experience*	(COLLINS *et al.*, 2012)	IEEE
E7	*Taming the embedded tiger - agile test techniques for embedded software*	(SCHOOENDERWOERT; MORSICATO, 2004)	IEEE
E8	*Test automation: Flexible way*	(IESHIN *et al.*, 2009)	IEEE
E9	*A survey of software testing practices in Canada*	(GAROUSI; ZHI, 2013)	Science Direct
E10	*Findings from a multi-method study on test-driven development*	(ROMANO *et al.*, 2017)	Science Direct
E11	*Studying test-driven development and its retainment over a six-month time span*	(BALDASSARRE *et al.*, 2021)	Science Direct
E12	*Establishing an Agile Testing Team: Our Four Favorite "Mistakes"*	(JOHANSEN; PERKINS, 2002)	Springer Link
E13	*Security Testing in Agile Web Application Development - A Case Study Using the EAST Methodology*	(ERDOGAN *et al.*, 2010)	Springer Link
E14	*The Focus on Usability in Testing Practices in Industry*	(LARUSDOTTIR *et al.*, 2010)	Springer Link
E15	*Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development*	(SILVA *et al.*, 2016)	Springer Link
E16	*Incremental Development of Model Transformation Chains Using Automated Testing*	(KÜSTER *et al.*, 2009)	Springer Link

Fonte: Elaborado pelo autor.

cada biblioteca, para com a quantidade inicial de trabalhos que se tinha ao início da execução do mapeamento, pode-se consultar a Figura 9, exposta logo abaixo:

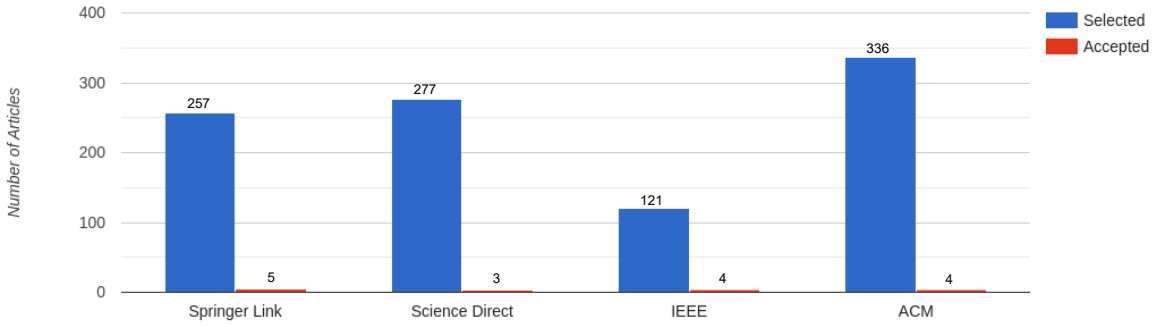
Para se obter uma menção em valores reais, a respeito do gráfico acima, pode-se consultar o Quadro 4, onde o mesmo armazena a quantidade final de trabalhos selecionados por cada biblioteca. Baseado nessas informações, pode-se também obter a compreensão da disposição dos trabalhos selecionados, baseado no intervalo de anos dos trabalhos que estiveram presente no mapeamento executado no presente trabalho, onde pode-se inferir a quantidade de trabalhos, selecionado em determinados anos, dentro da faixa estimada, como pode-se conferir isso na Figura 10, exposta logo abaixo:

Figura 8 – Porcentagem de trabalhos por biblioteca.



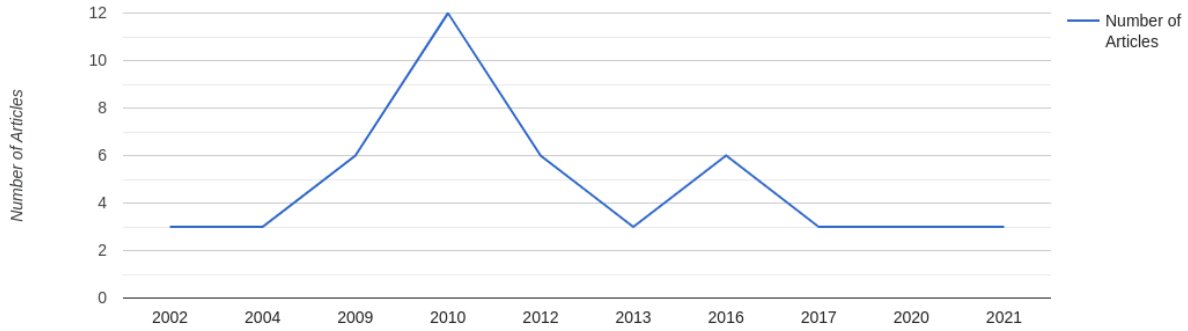
Fonte: Exportado da ferramenta *Parsifal*.

Figura 9 – Quantidade de trabalhos selecionados pela quantidade inicial.



Fonte: Exportado da ferramenta *Parsifal*.

Figura 10 – Quantidade de trabalhos selecionados por ano.



Fonte: Exportado da ferramenta *Parsifal*.

5.4 Análises e resultados das questões de pesquisa

Nesta presente Subseção, será desenvolvida toda uma análise por parte dos dados coletados por meio dos trabalhos selecionados no mapeamento sistemático da literatura, executado no presente trabalho, onde a análise se dará passando de forma individual por cada uma das questões de pesquisa, que foram elencadas para o mapeamento, onde os dados foram extraídos com base no que foi definido ao fim da Seção 5.1.

5.4.1 QP1 - Quais as técnicas de testes de software, que estão sendo empregadas no contexto de desenvolvimento ágil?

Tomando como base o que se foi coletado por meio dos trabalhos selecionados no mapeamento via extração de dados, extração essa executada com base no que foi definido e está exposto na Figura 6, bem como as definições da própria QP1, que está definida mais ao inicio da Seção 5.1, assim segue exposto no Quadro 6, a sumarização dos dados coletados, que se segue logo abaixo:

Quadro 6 – Resultado da coleta de dados para QP1.

Trabalho	Caixa branca	Caixa preta	Caixa cinza
E1	X	X	X
E2	X		
E3	X		
E4	X	X	
E5	X	X	
E6	X		
E7	X		
E9	X	X	
E10	X	X	
E11	X		
E12		X	
E13	X		
E14		X	
E15		X	
E16	X	X	

Fonte: Elaborado pelo autor.

Com base nas informações expostas no quadro acima, pode-se concluir que no geral, as técnicas de testes mais aplicadas estão em torno das técnicas de caixa branca, dado que são técnicas que acontecem ainda ao nível de código, ou seja, é uma técnica que em suma tem sua execução bem atrelada a visão das responsabilidades de um desenvolvedor, como, por exemplo, as escritas de testes de unidade, de integração e semelhantes, as técnicas de teste de caixa preta aparecem também com grande relevância pelo que foi coletado, onde no geral, as mesmas são aplicadas como uma forma complementar as de caixa branca, ou seja, se testa em caixa branca e a posterior em caixa preta, salvo algumas exceções que podem ser notadas analisando o quadro, como, por exemplo, os trabalhos E12, E14 e E15 que empregam somente e diretamente as práticas de teste de caixa preta a fim de garantir qualidade do que está sendo desenvolvido e entregue ao cliente.

Ainda com base no que se foi coletado, a aplicação efetiva e prática das técnicas de caixa cinza, ainda são um campo com espaço para desenvolvimento e expansão, dado que apenas o trabalho E1 apresenta sua utilização real, e dado que a mesma é uma mescla dos mundos que seria os testes de caixa branca e preta, a compreensão em torno da mesma pode estar com pouca definição por parte dos possíveis aplicantes da mesma, e isso pode estar atrelado também ao tamanho de porte da equipe ou projeto em que se pode empregar a mesma, dado que no E1, tomando como base o contexto exposto, era de uma organização com tamanho elevado e projetos de alta complexidade.

Foi observado também, como era a estrutura geral dos times ágeis envolvidos nos projetos, a fim de coletar dados e compreender o tamanho médio ou aproximado dos times. A respeito disso foi notado que existem equipes que atuam com uma quantidade inferior a três integrantes, outras com equipes como prega a literatura, com médio de quatro integrantes, algumas outras ultrapassando o recomendado como equipes indo até oito integrantes. E em poucos casos, mas casos reais, equipes com mais de nove integrantes contribuindo e forma ativa com as atividades.

No geral, os métodos ágeis empregados pelas equipes e organizações sem resumem em *Scrum* e *XP*, dado que os mesmos são massivamente os que mais possuem ocorrências. Mas, há alguns casos mais distintos, onde ainda sim são aplicados os mesmos como base, mas com algumas adições, como é o caso do *Scrum combo XP*, que basicamente visa unificar o melhor das práticas de ambos os métodos. Ocorreu também o *Scrumban*, que no geral é uma junção de *Scrum* e *Kanban*.

5.4.2 QP2 - Quais as etapas de testes de software, que estão sendo abordadas e executadas na prática, dentro do contexto ágil de desenvolvimento?

De forma semelhante como ao da QP anterior, segue exposto no Quadro 7, situado mais abaixo no texto, as análises e dados coletados da QP em questão, bem como uma sumarização dos entendimentos tomados com base nos dados expostos.

Quadro 7 – Resultado da coleta de dados para QP2.

Trabalho	**Etapas de teste**
E1	Testes de unidade, Testes de aceitação
E2	Testes de unidade, Testes de referência
E3	Testes de unidade
E4	Testes de unidade, Testes de carga
E5	Testes de unidade, Testes de aceitação
E6	Testes de unidade, Testes de regressão, Testes funcionais
E7	Testes de unidade
E8	Testes de unidade
E9	Testes de unidade, Testes de integração, Testes de sistema, Testes funcionais, Testes de desempenho
E10	Testes de unidade, Testes de integração, Testes de sistema
E11	Testes de unidade
E12	Testes de implantação
E13	Testes de unidade, Testes de integridade
E14	Testes de segurança, Testes de penetração
E15	Testes de unidade, Testes de integração
E16	Testes de unidade, Testes de usabilidade, Testes de aceitação, Testes de sistema

Fonte: Elaborado pelo autor.

Tomando como base os dados que estão expostos no Quadro 7, localizado mais abaixo no texto, já se pode ter uma conclusão a respeito do quão difundido e aplicado é a etapa de testes de unidade dentro dos trabalhos analisados, onde a aplicação do mesmo independe de tamanho da organização, tamanho do projeto e complexidade dos mesmos, dado que aparece em praticamente todos os trabalhos. Em seguida, algumas outras tantas etapas de teste aparecem nos contextos dos trabalhos, como, por exemplo, de testes de aceitação, carga, regressão, funcional, sistema, usabilidade entre outros. Que no geral também são etapas bem conhecidas e difundidas pela literatura. A respeito da análise dos dados referente aos métodos ágeis que estão sendo mais

aplicados nos trabalhos, pode ser consultada na Subseção 5.4.1, situada anteriormente no texto.

Retomando uma das análises exposta na Subseção 5.4.1, onde foi observado um forte uso das técnicas de caixa branca e por seguinte preta, baseado nas etapas aplicadas pelos trabalhos analisados na presente QP, tal análise se torna mais clara e fundamentada, dado que testes de unidade e integração, por exemplo, em suma são testes de caixa branca, ligados mais a visão do desenvolvedor, e testes de aceitação, funcionais, sistema, usabilidade e similares, são em suma testes de caixa preta, ratificando as relações encontradas na QP anterior. Dado que são as principais ocorrências observadas por parte desta QP.

Outro ponto de análise que pode ser tomado, é em relação a algumas etapas de testes que surgiram por parte de alguns trabalhos selecionados, mas que não se tem muito acervo na literatura sobre os mesmas ainda, como, por exemplo: testes de referência, que diz respeito a um tipo de teste executado geralmente por *designers*, em suma os que desempenham cargos de *UI*/*UX*, onde os mesmos executam uma verificação fina comparando o que foi prototipado ao nível de interface com o que foi realmente implementado na interface final para o usuário, a fim de buscar diferenças de *pixels*, espaçamentos e etc, como foi exposto pelo trabalho E1, E2 e E5. Fazendo assim uma comparação do que está feito para com a referência.

Outra etapa menos recorrente na literatura, foi a de teste de implantação, que em suma são tipos específicos de testes realizados em momento de subida de projetos para produção e afins, mas sem tantos detalhes dos passos e procedimentos. Outros dois foram os testes de segurança e penetração, que dizem mais a respeito a etapas de testes que focam em segurança, onde as mesmas podem ser combinadas em conjunto ou independentes, mas dizem respeito a testes que visam garantir a segurança do que se está desenvolvendo, como, por exemplo, os testes de penetração: com execução de tentativas de injeção de *SQLs* maliciosos por meio de formulários de cadastro, bem como inserção de *scripts* que rodam no *client-side* para a tentativa de obtenção de dados, para buscar assegurar que as aplicações desenvolvidas estão dando aporte e preparadas para tratar tais ocorrências. Como foi observado nos trabalhos E9, E13 e E14.

5.4.3 QP3 - Quais as ferramentas de teste de software, que comumente estão sendo empregadas em conjunto com as técnicas e etapas de teste, como suporte e apoio a execução dos testes, no cenário ágil de desenvolvimento de softwares?

De forma semelhante as QPs anteriores, segue exposto no Quadro 8, as análises e dados que foram coletados da QP referida, bem como uma visão geral e sumarização a respeito

dos entendimentos dos achados, baseados nos dados expostos:

Quadro 8 – Resultado da coleta de dados para QP3.

Trabalho	***Frameworks***	**Ferramentas**
E2	*JUnit*	
E3	*JUnit*	*Plug-ins*
E4	*JUnit*, *Mockito*	*JMeter*
E5		Testes automatizados
E6	*JUnit*	Testes automatizados, *Google Test*, *Sikuli*, *Selenium*, *FitNesse Test Tool*
E9	*JUnit*, *XUnit*, *NUnit*	*IBM Rational Functional Tester*, *Selenium*
E10	*JUnit*	
E11	*JUnit*	*SonarQube*
E13	*JUnit*	
E15	*JUnit*, *JBehave*	*Selenium*

Fonte: Elaborado pelo autor.

[1]

Observando os dados que foram coletados por meio da QP3, foi inicialmente notada uma separação quando se refere a ferramentas. Separação essa tida em dois ramos, o primeiro que são os *frameworks*, como, por exemplo: *JUnit*, que inclusive é o que mais possui ocorrência dentre os trabalhos analisados, onde o mesmo se configura para dar suporte a escrita de testes na linguagem *Java*, teste de unidade, por exemplo. O *Mockito*, que também é escrito sobre a linguagem *Java*, e o mesmo tem o objetivo de oferecer suporte a escrita e execução de testes de integração na referida linguagem, ainda relacionado a linguagem *Java*. Também aparece o *JBehave*, que se põe como um *framework* para escrita de testes, mas com foco para o BDD.

Ainda se referindo aos *frameworks*, mas já foram da linguagem *Java*, dado que a da vez é o *C#*, com algumas poucas ocorrências, mas ocorrências de importância elevada, dado o contexto das organizações em que foram empregados, se tem o *XUnit*, focado para escrita de testes dentro do ecossistema *.NET* e também *NUnit*, mas focado a escrita de testes dentro da linguagem *C#* em si.

Já quando se fala em ferramentas finais, que não *frameworks*, foram coletadas ocorrências variadas, como, por exemplo: *plug-ins* ou extensões, que foram basicamente expostos

[1] https://junit.org/junit5/ - https://site.mockito.org/ - https://jmeter.apache.org/ - https://xunit.net/ - https://nunit.org/ - https://jbehave.org/ - https://google.github.io/googletest/ - http://doc.sikuli.org/ - https://www.selenium.dev/ - http://docs.fitnesse.org/FrontPage - https://www.ibm.com/products/rational-functional-tester - https://www.sonarqube.org/

como adicionais que podem ser colocados nas *IDEs* para otimizar algumas atividades, houve também a ocorrência da ferramenta *JMeter*, uma ferramenta totalmente escrita na linguagem *Java*, que oferece suporte e apoio a execução de testes tanto de carga, quanto performance, tempo de resposta, robustez e etc, seja em *APIs*, *Back-ends*, *Front-ends* e etc.

Houveram também ocorrência de ferramentas que trabalham fornecendo tipos de automação, como, por exemplo: *Google Test*, *Sikuli*, *FitNesse Test Tool*, *IBM Rational Functional Tester*, *SonarQube* e *Selenium*. Dentre essas, algumas se destacam, como, por exemplo, a *IBM Rational Functional Tester*, focada na automação de testes não funcionais e o *Selenium*, o qual é uma ferramenta extremamente famosa e que não atoa é a ferramenta que mais possui ocorrências dentre todas as citadas pelos trabalhos.

E houve também a ocorrência das expressões testes automatizados, como podemos notar algumas, no Quadro 8, mas quando ocorreram não foi aprofundado pelos trabalhos do que se tratava essas ferramentas de testes automatizados em si.

5.4.4 QP4 - Quais os modelos de testes vem sendo comumente adotados dentro dos cenários ágeis de desenvolvimento de software?

Assim como ocorreu com as QPs anteriores, segue exposto no Quadro 9, as análises e dados que foram previamente coletados por meio dos trabalhos selecionados pelo mapeamento. Análises essas que enfocam o contexto da referida QP, após a exposição do quadro, segue uma visão geral e sumarização a respeito dos entendimentos obtidos a partir dos achados da própria QP. As definições aplicadas para a referida coleta dos dados, são os que estão expostos na Figura 6, bem como a definição da própria QP que está presente na Seção 5.1, bem mais ao fim das declarações das QPs que foram elaboradas para o mapeamento, assim seguem os dados expostos a seguir:

Com base nos achados expostos no quadro situado acima, um modelo de testes que se destaca, possuindo inúmeras ocorrências dentre os trabalhos analisados, é o teste baseado em cobertura, comumente conhecimento como *coverage*. Que em suma é um modelo que se preocupa em cobrir ao máximo possível o que se precisa testar, e com base no que foi coletado por meio dos trabalhos, não necessariamente cobrir da melhor forma. Como, por exemplo, analisando e trabalhando sobre as classes de equivalência, mas sim, apenas fazendo com que a cobertura esteja sempre o mais próximo dos 100%. Por mais que nenhuma classe de equivalência tenha sido testada, assim focando na quantidade/cobertura do teste, do que com a qualidade do

Quadro 9 – Resultado da coleta de dados para QP4.

Trabalho	**Modelos de teste**
E1	Testes exploratórios, Testes contínuos, Testes orientados a planos
E2	Testes baseados em cobertura
E3	Testes baseados em cobertura
E4	Testes baseados em cobertura
E5	Testes de API
E6	Testes baseados em cobertura
E7	Testes baseados em cobertura
E9	Testes exploratórios, Testes baseados em cobertura, Testes baseados em casos
E13	Testes baseados em transformação do modelo
E14	Testes baseados em diagramas UML
E15	Testes baseados nos usuários

Fonte: Elaborado pelo autor.

teste em si.

Foram notadas também outras ocorrências, como teste orientados a planos, que basicamente diz respeito a elaboração de planos de testes, que visam guiar toda a fase de testagem, independente das técnicas, etapas e modelos escolhidos pelo projeto. Testes de *API*, que basicamente visam em executar testes somente onde estão alocadas as regras de negócios da aplicação. Testes baseados em casos de uso, uma forma de testar semelhante à orientada a planos, mas que seu foco de execução se dá sobre os casos de uso da aplicação. Testes baseados em usuários, cujo objetivo é testar baseados nas *personas* elencadas para o projeto.

Com base nos achados também, foi compreendido que não há nenhuma relação entre haver influências comprovadas relacionadas a seleção de modelos de testes, dado que a seleção do mesmo independe de do método ágil que foi selecionado, bem como também independe das técnicas e etapas de teste adotadas para um projeto.

5.4.5 QP5 - Quais as metodologias de testes de softwares que ultimamente vem sendo adotadas pelos cenários ágeis de desenvolvimento de software e aplicações?

De forma equivalente as QP anteriores, seque exposto no Quadro 10, as análises e dados que foram extraídos pela referida QP, dos trabalhos selecionados para o mapeamento, bem como uma visão mais geral a respeito dos entendimentos dos achados, baseados nos dados expostos logo a baixo:

Quadro 10 – Resultado da coleta de dados para QP5.

Trabalho	**Metodologias de teste**
E2	TDD
E3	TDD
E9	TDD, TLD, BDD
E10	TDD, TLD
E11	TDD
E14	TDD
E15	BDD, ATDD

Fonte: Elaborado pelo autor.

De todas as cinco QPs, essa foi a que menos houve retorno de dados extraídos dos trabalhos, mas o que se nota do que foi extraído, é que dentre as metologias de teste de software, a que mais prevalece e ocorre entre os trabalhos, é o TDD *Test Driven Development*, pois o mesmo está em todas as ocorrências de metodologias ágeis, com exceção do trabalho E16, mostrando o quão difundido e aplicado está o conhecimento a respeito do TDD, e dentre todas as metodologias de teste apresentadas pelos dados, a mesma se configura como a mais simples de ser aplicada, onde em suas ocorrências independe do método ágil do projeto, dado que houve ocorrência tanto no *Scrum* quanto no *XP*, que em suma são os métodos ágeis mais aplicados pelos trabalhos.

Notou-se também, algumas ocorrências do BDD *Behavior Driven Development*, não tão popular quanto o TDD, mas isso se configura pela diferença de complexidade de aplicação que existe entre ambos, e que da mesma forma como o TDD, independeu do método ágil adotado para o projeto, mas, em contrapartida, exigiu equipes mais experientes para sua aplicação correta. Houveram também ocorrências mais isoladas e não tanto expressas na literatura, como o ATDD *Acceptance Test Driven Development*, uma espécie de TDD dirigido e guiado a testes de aceitação, que se configura como complexo de ser aplicado. Uma última ocorrência, mais não menos importante, foi o TLD *Test Last Development*, que basicamente é uma metodologia de teste que foca em realizar os testes somente ao fim de todo o percurso de implementação.

Alguns outros entendimentos extraídos dos dados, mas focado em uma análise geral das cinco QPs, é que em suma, realizar ou aplicar a prática de testes dentro das metodologias ágeis de desenvolvimento é mais fácil e simples, dado que se testa o software ou aplicação em partes isoladas inicialmente, e isso se configura melhor e mais produtivo do que testar um software todo de uma única forma e vez.

A aplicação de TDD, BDD e afins como foi notado nos trabalhos independe do método ágil, pois houve ocorrências tanto no *Scrum* quanto no *XP*, mas o TDD e o BDD performam melhor e são mais simples de aplicar quando dentro de um contexto guiado pelo *XP*. E pelo que foi notado nos trabalhos, está em ascensão o TLD, que foca na execução de testes ao final de toda a bateria de desenvolvimento, indo um pouco contra o que prega a prática de testes dentro do contexto ágil de desenvolvimento.

E uma coisa que ficou bem claro dentre a maioria dos trabalhos, é a necessidade de haver assim como há a revisão de códigos, haver também a revisão de testes, dado que esse é um cenário que cada vez cresce mais, as aplicações estão demandando mais e mais testes, que no que lhe concerne, são mais e mais complexos, assim a revisão dos mesmos, assim como ocorre com os códigos fonte dos projetos, está se tornando cada vez mais necessário.

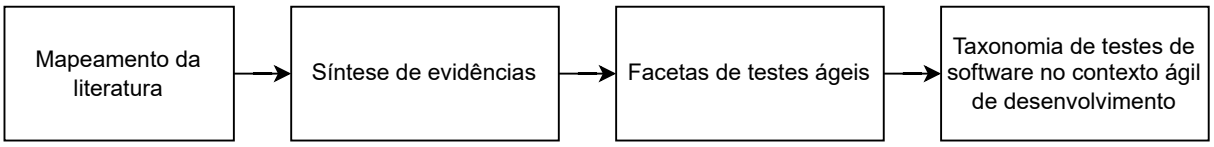
6 TAXONOMIA

Este Capítulo apresenta a estruturação e a proposta da taxonomia de práticas de testes em metodologias ágeis. Onde o conteúdo do mesmo foca no processo de proposta da taxonomia, desde o fluxo de proposta, passando pela aplicação da classificação facetada sobre os achados do mapeamento sistemático da literatura, para por fim chegar a proposta da mesma.

6.1 Fluxo de proposta da taxonomia

Dado que a base de conhecimento para a proposta da taxonomia se dá sobre a execução do mapeamento sistemático, feito pelo presente trabalho, o mesmo se põe como marco inicial do fluxo de proposta da taxonomia, seguido uma síntese de evidências, síntese essa que pode ser consultada na Seção 5.4, em seguida se elenca algumas facetas, onde as mesmas são baseados no que foi encontrado por parte do mapeamento e por fim se realiza a proposta da taxonomia. Como pode ser observado por meio da Figura 11, toda à diagramação desse fluxo:

Figura 11 – Diagrama de fluxo de proposta da taxonomia.



Fonte: Elaborado pelo autor.

Como pode ser observado, o fluxo de proposta da taxonomia é um fluxo linear, onde cada retângulo que representa uma atividade ou grupos de atividades, só pode ser iniciado caso o anterior tenha sido concluído com sucesso, com a observação de que os dois primeiros retângulos já foram concluídos por meio do mapeamento sistemático da literatura, que pode ser conferida sua execução no Capítulo 5.

6.2 Facetas de testes ágeis

Baseado no que foi coletado e compreendido por meio do mapeamento sistemático, foram elencados um total de sete facetas dentro do tema de domínio, onde as mesmas seguem expostas na lista a baixo:

- Dimensão dos times ágeis.
- Métodos ágeis aplicados.

– Técnicas utilizadas em contextos ágeis de teste.
– Etapas utilizadas em contextos ágeis de teste.
– Ferramentas utilizadas em contextos ágeis de teste.
– Modelos utilizadas em contextos ágeis de teste.
– Metodologias utilizadas em contextos ágeis de teste.

Onde cada uma das facetas diz respeito ao que foi coletado por meio do mapeamento sistemático quando faltando em trabalhos, bem como o que foi extraído desses trabalhos selecionados com base no que foi definido para a extração de dados.

Quando se refere a faceta de dimensão dos times ágeis, foi analisado os aspectos relacionados ao tamanho dos times que estão atuando nos projetos, tais aspectos podem ser conferidos no Quadro 11, situado logo a baixo:

Quadro 11 – Aspectos da faceta de dimensões dos times.

Aspecto de dimensão	**Quantidade de integrantes**
Times extremamente pequenos	Com menos de 3 integrantes
Times pregados pela literatura	Com em média 4 integrantes
Times mediamente crescidos	Com até 8 integrantes
Times populosos	Com mais de 9 integrantes

Fonte: Elaborado pelo autor.

Observando a faceta de quais métodos ágeis são aplicados de forma real e prática nos projetos relatados pelos trabalhos, elencados pelo mapeamento, tem-se a visão geral, que está situada no Quadro 12, situado logo em seguida:

Quadro 12 – Aspectos da faceta de métodos ágeis aplicados.

Métodos aplicados
Scrum
XP
Scrum combo XP
Scrumban

Fonte: Elaborado pelo autor.

Analisando a faceta de técnicas ágeis de testes que estão sendo empregadas, com base no que foi coletado e extraído de dados por meio dos trabalhos selecionados, foi obtido a visão geral que segue exposta no Quadro 13, situado logo após esse trecho:

Quadro 13 – Aspectos da faceta de técnicas utilizadas em contextos ágeis de teste.

Técnicas de testes	**Referência dos trabalhos**
Testagem baseada em código fonte	Testagem em caixa branca
Testagem baseada em sistematização	Testagem em caixa preta

Fonte: Elaborado pelo autor.

Analisando a faceta de etapas ágeis de testes que estão sendo aplicadas, foi obtido a compreensão, que está sumarizada na visão geral que segue exposta no Quadro 14, situado logo após esse trecho:

Quadro 14 – Aspectos da faceta de etapas utilizadas em contextos ágeis de teste.

Etapas de testes	**Referência dos trabalhos**
Testagem de unidades lógicas	Testes unitários
Testagem de unidades conjuntas	Testes de integração
Testagem de funcionamento real	Testes funcionais, Testes de regressão
Testagem de design aplicado	Testes de referência, Testes de usabilidade
Testes de concretude e falhas	Testes de segurança, Testes de penetração em software

Fonte: Elaborado pelo autor.

Analisando a faceta de ferramentas ágeis de testes que estão sendo utilizadas para simplificar a aplicação dos testes, foi obtido a compreensão, que está sumarizada na visão geral que segue exposta no Quadro 15, situado logo após esse trecho:

Quadro 15 – Aspectos da faceta de ferramentas utilizadas em contextos ágeis de teste.

Aspectos das ferramentas	**Ferramentas**
Ferramentas de criação	*JUnit*, *XUnit*, *NUnit*, *Mockito*, *JBehave*
Ferramentas de suporte	*SonarQube*, *Selenium*, *JMeter*, *Google Test*, *IBM Rational Functional*

Fonte: Elaborado pelo autor.

Analisando a faceta de modelos ágeis de testes que estão sendo selecionados para aplicação dentro dos trabalhos, que foram elencados por meio do mapeamento sistemático, onde as compreensões a respeito do que foi coletado e sumarizado para essa referida faceta, pode ser consultado na exposição dos achados da QP4, bem como a análise de resultados desses achados por meio da QP4, que pode ser visualizado na Subseção 5.4.4, sendo obtido de compreensão está compilada na visão geral que segue exposta no Quadro 16, situado logo após esse trecho:

Analisando a faceta de metodologias ágeis de testes que estão sendo selecionadas e

Quadro 16 – Aspectos da faceta de modelos utilizadas em contextos ágeis de teste.

Modelos ágeis de teste	**Referência dos trabalhos**
Testes para *coverage*	Testes de unidade, Teste de integração, Testes de sistema
Testes de domínio	Testes de API, Teste de interface
Testes guiados a modelos	Testes orientados a planos, Testes baseados em casos, Testes baseados em UML, Testes baseados em usuários

Fonte: Elaborado pelo autor.

aplicadas, foi obtido a compreensão, que está sumarizada na visão geral que segue exposta no Quadro 17, situado logo após esse trecho:

Quadro 17 – Aspectos da faceta de metodologiasutilizadas em contextos ágeis de teste.

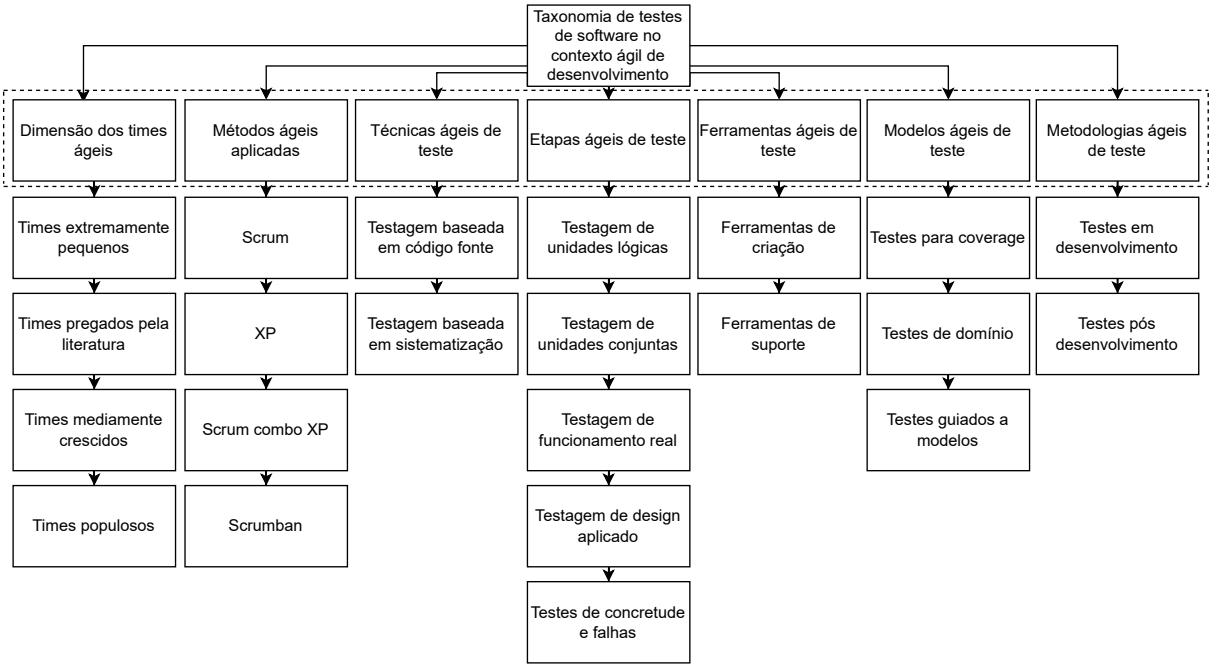
Aspectos das metodologias	**Metodologias**
Testes em desenvolvimento	TDD, BDD, ATDD
Testes pós desenvolvimento	TLD

Fonte: Elaborado pelo autor.

6.3 Taxonomia de testes de software no contexto ágil de desenvolvimento

Por fim, segue exposto por meio da Figura 12, a proposta final da taxonomia de testes de software no contexto ágil de desenvolvimento, proposta essa baseada no mapeamento sistemático da literatura que foi executado pelo presente trabalho.

Figura 12 – Diagrama da taxonomia proposta.



Fonte: Elaborado pelo autor.

7 LIMITAÇÕES E AMEAÇAS A VALIDADE

Este capítulo apresenta algumas limitações encontradas durante a execução do presente trabalho, foram encontradas algumas limitações que impactaram tanto na execução quanto no resultado do mesmo, em conjunto com tais limitações, foram observados também alguns pontos, que podem ser postos como ameaças a validade para o presente trabalho.

7.1 Limitações encontradas

Referente ao presente trabalho, foram encontradas algumas limitações que impactaram diretamente no resultado do mesmo, como o acesso aos trabalhos no momento da execução do mapeamento, dado que alguns necessitava de algum tipo de assinatura para obter acesso ao conteúdo do mesmo, e tal ocorrência foi notada em duas das bibliotecas que estão presentes no mapeamento.

Outra limitação que também afetou a execução do mapeamento, foi a falta de uma "equipe" ou pelo menos mais uma pessoa para participar das revisões dos trabalhos, etapa a etapa da execução do mapeamento, onde o objetivo era ser o escritor/autor do presente trabalho em conjunto com pelo menos mais uma pessoa para realizar as revisões primárias, e a professora orientadora do presente trabalho realizar as revisões secundárias, que seria sobre as primárias, mas como isso não foi possível, ficou somente o escritor/autor do presente trabalho nas revisões primárias, e professora orientadora nas secundárias como planejado previamente.

7.2 Ameaças a validade

Referente ao presente trabalho, foram elencadas duas ameaças, que podem ser tomadas como as maiores, que podem ter maior influência sobre o resultado e no que lhe concerne a validade final, a primeira sendo a inexperiência do escritor/autor do presente trabalho tanto no planejamento quanto na execução do mapeamento sistemático da literatura, dado que no planejamento há a definição de muitas etapas, formas de avaliar, filtrar e selecionar bem como a própria execução em si do mapeamento, a falta de experiência afeta diretamente na qualidade do mapeamento no que lhe concerne no resultado do mesmo.

Uma segunda ameaça que também pode influenciar diretamente no resultado, em conjunto com a ameaça exposta anteriormente, é o tempo disponível para a execução do mapeamento sistemático, que pelo contexto do presente trabalho foi um tempo relativamente que

pequeno, que se intensifica ainda mais como ameaça quando se considera a outra ameaça já citada relacionada a experiência, pois dado que a experiência por parte do escritor/autor do presente trabalho era baixíssima, ter uma janela de tempo maior para a execução do mapeamento poderia mitigar alguns pontos que influenciassem menos ainda no resultado.

8 CONCLUSÕES E TRABALHOS FUTUROS

Com base nos resultados obtidos por meio do mapeamento sistemático da literatura que foi executado pelo presente trabalho, onde a execução e os dados estão expostos no Capítulo 5, focando as análises nos seguintes Quadros: Quadro 6, Quadro 7, Quadro 8, Quadro 9, Quadro 10, bem como o resumo geral das compreensões que existe após os mesmos, e olhando a taxonomia contida no Capítulo 6, dentro do capítulo mais especificamente observando a Figura 12, pode-se concluir que o contexto de desenvolvimento ágil favorece a aplicação das práticas de testes, ao passo de que as soluções são entregues por partes ao cliente, testar essas partes se tornou uma tarefa mais simples, ao invés de se testar uma aplicação toda de uma vez.

Ainda existe uma grande lacuna entre o que a literatura prega relacionado as práticas de testes, ao que é realmente empregado de fato, mas já pode-se observar que está havendo uma popularização da aplicação das práticas de testes, onde o esperado é que sejam inicialmente empregadas as práticas mais simples de serem executadas, assim, baseado nisso já se pode notar que os testes estão sendo considerados fator primordial quando se trata em manutenção de qualidade do que está sendo desenvolvido e entregue ao cliente ou usuário final.

Também pode-se observar que algumas possíveis tendências estão surgindo, como, por exemplo: a execução de testes focados em segurança, testes executados por profissionais das áreas relacionadas a *designer*, bem como o possível surgimento da metodologia de teste TLD, que foca nos procedimentos de teste bem ao final das fases de desenvolvimento. Bem como pode ser notado, que de certa forma, aplicações tanto de TDD quanto de BDD, se apresentaram de forma forte e influente nos trabalhos.

Referente a trabalhos futuros, é visto com válido e necessário, a criação e aplicação de um *Survey*, a fim de validar os achados do presente trabalho, onde o foco da aplicação idealmente seria no mercado, mais especificamente com analistas de testes e desenvolvedores no geral, para com isso, realizar uma validação do quão efetiva e aplicável é a taxonomia. Pensando em trabalhos futuros mais extremos e complexos, talvez refazer os passos executados no presente trabalho, mas focando em mitigar os pontos elencados no Capítulo 7, como, por exemplo, uma equipe bem definida para a execução do mapeamento, para com isso se propor uma nova taxonomia com uma quantidade menor de pontos de falhas.

REFERÊNCIAS

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. Agile software development methods: Review and analysis. **arXiv preprint arXiv:1709.08439**, 2017.

ABRAN, A.; MOORE, J. W.; BOURQUE, P.; DUPUIS, R.; TRIPP, L. **Software engineering body of knowledge**. **IEEE Computer Society, Angela Burgess**, p. 25, 2004.

AL-SAQQA, S.; SAWALHA, S.; ABDELNABI, H. Agile software development: Methodologies and trends. **International Journal of Interactive Mobile Technologies**, v. 14, n. 11, 2020.

ALAIDAROS, H.; OMAR, M.; ROMLI, R. The key factors of evaluating agile approaches: A systematic literature review. **International Journal of Supply Chain Management (IJSCM)**, ExcelingTech Publishers, v. 8, n. 2, p. 1–11, 2019.

AQUINO, I. J.; CARLAN, E.; BRASCHER, M. B. Princípios classificatórios para a construção de taxonomias. **PontodeAcesso**, v. 3, n. 3, p. 196–215, 2009.

ARRIETA, C. J. G.; MARTÍNEZ, J. L. D.; BOHÓRQUEZ, M. O.; MANOTAS, A. K. D. L. H.; CORREA, E. M. D. L. H.; ORTEGA, R. C. M. Agile testing practices in software quality: State of the art review. **Journal of Theoretical and Applied Information Technology**, 2016.

ARUMUGAM, A. K. Software testing techniques & new trends. **International Journal of Engineering Research & Technology (IJERT)**, v. 8, n. 12, 2019.

ATAWNEH, S. The analysis of current state of agile software development. **Journal of Theoretical Applied Information Technology**, v. 97, p. 22, 2019.

BALDASSARRE, M. T.; CAIVANO, D.; FUCCI, D.; JURISTO, N.; ROMANO, S.; SCANNIELLO, G.; TURHAN, B. Studying test-driven development and its retainment over a six-month time span. **Journal of Systems and Software**, v. 176, p. 110937, 2021. ISSN 0164-1212. Acesso em: 22 jun. 2022. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121221000340.

BARRAOOD, S. O.; MOHD, H.; BAHAROM, F. A comparison study of software testing activities in agile methods. In: **KNOWLEDGE MANAGEMENT INTERNATIONAL CONFERENCE (KMICE) 2021**. [*S. l.*: *s. n.*], 2021.

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

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. *et al.* **Manifesto for agile software development**. Snowbird, UT, 2001.

BOEHM, B. W.; BROWN, J. R.; LIPOW, M. Quantitative evaluation of software quality. In: **PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING**. [*S. l.*: *s. n.*], 1976. p. 592–605.

BRAUDE, E. J.; BERNSTEIN, M. E. **Software engineering**: modern approaches. [*S. l.*]: Waveland Press, 2016.

BRITTO, R.; USMAN, M. Bloom's taxonomy in software engineering education: A systematic mapping study. In: IEEE. **2015 IEEE FRONTIERS IN EDUCATION CONFERENCE (FIE)**. [*S. l.*], 2015. p. 1–8.

BRITTO, R.; WOHLIN, C.; MENDES, E. An extended global software engineering taxonomy. **Journal of Software Engineering Research and Development**, Springer, v. 4, n. 1, p. 1–24, 2016.

BUFFARDI, K.; EDWARDS, S. H. Exploring influences on student adherence to test-driven development. In: **PROCEEDINGS OF THE 17TH ACM ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION**. New York, NY, USA: Association for Computing Machinery, 2012. (ITiCSE '12), p. 105–110. ISBN 9781450312462. Https://doi.org/10.1145/2325296.2325324. Acesso em: 22 jun. 2022.

CHOPADE, M. R. M.; DHAVASE, N. S. Agile software development: Positive and negative user stories. In: IEEE. **2017 2ND INTERNATIONAL CONFERENCE FOR CONVERGENCE IN TECHNOLOGY (I2CT)**. [*S. l.*], 2017. p. 297–299.

CHOUDHARY, B.; RAKESH, S. K. An approach using agile method for software development. In: IEEE. **2016 INTERNATIONAL CONFERENCE ON INNOVATION AND CHALLENGES IN CYBER SECURITY (ICICCS-INBUSH)**. [*S. l.*], 2016. p. 155–158.

COHEN, D.; LINDVALL, M.; COSTA, P. Agile software development. **DACS SOAR Report**, Citeseer, v. 11, p. 2003, 2003.

COLLINS, E.; DIAS-NETO, A.; JR., V. F. d. L. Strategies for agile software testing automation: An industrial experience. In: **2012 IEEE 36TH ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE WORKSHOPS**. [*S. l.*: *s. n.*], 2012. p. 440–445.

DOĞAN, E.; TÜZÜN, E. Towards a taxonomy of code review smells. **Information and Software Technology**, Elsevier, v. 142, p. 106737, 2022.

ELBANNA, A.; SARKER, S. **The risks of agile software development:** learning from adopters. **IEEE Software**, IEEE, v. 33, n. 5, p. 72–79, 2015.

ENGSTRÖM, E.; PETERSEN, K. Mapping software testing practice with software testing research—serp-test taxonomy. In: IEEE. **2015 IEEE EIGHTH INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS (ICSTW)**. [*S. l.*], 2015. p. 1–4.

ERDOGAN, G.; MELAND, P. H.; MATHIESON, D. Security testing in agile web application development - a case study using the east methodology. In: SILLITTI, A.; MARTIN, A.; WANG, X.; WHITWORTH, E. (Ed.). **Agile Processes in Software Engineering and Extreme Programming**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 14–27. ISBN 978-3-642-13054-0.

ERICKSON, J.; LYYTINEN, K.; SIAU, K. Agile modeling, agile software development, and extreme programming: the state of research. **Journal of Database Management (JDM)**, IGI Global, v. 16, n. 4, p. 88–100, 2005.

FERREIRA, J.; NOBLE, J.; BIDDLE, R. Agile development iterations and ui design. In: IEEE. **Agile 2007 (AGILE 2007)**. [*S. l.*], 2007. p. 50–58.

FORWARD, A.; LETHBRIDGE, T. C. **A taxonomy of software types to facilitate search and evidence-based software engineering**. In: **Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds**. [*S. l.*: *s. n.*], 2008. p. 179–191.

FOWLER, M.; HIGHSMITH, J. *et al.* The agile manifesto. **Software development**, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35, 2001.

GAROUSI, V.; ZHI, J. A survey of software testing practices in canada. **Journal of Systems and Software**, v. 86, n. 5, p. 1354–1376, 2013. ISSN 0164-1212. Acesso em: 22 jun. 2022. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121212003561.

GODFRAY, H. C. J. Challenges for taxonomy. **Nature**, Nature Publishing Group, v. 417, n. 6884, p. 17–19, 2002.

GOESCHL, S.; HERP, M.; WAIS, C. **When Agile Meets OO Testing:** a case study. In: **Proceedings of the 1st Workshop on Testing Object-Oriented Systems**. New York, NY, USA: Association for Computing Machinery, 2010. (ETOOS '10). ISBN 9781450305389. Disponível em: https://doi.org/10.1145/1890692.1890702.

GUPTA, S.; GOUTTAM, D. Towards changing the paradigm of software development in software industries: An emergence of agile software development. In: IEEE. **2017 IEEE INTERNATIONAL CONFERENCE ON SMART TECHNOLOGIES AND MANAGEMENT FOR COMPUTING, COMMUNICATION, CONTROLS, ENERGY AND MATERIALS (ICSTM)**. [*S. l.*], 2017. p. 18–21.

HELLMANN, T. D.; SHARMA, A.; FERREIRA, J.; MAURER, F. Agile testing: Past, present, and future–charting a systematic map of testing in agile software development. In: IEEE. **2012 AGILE CONFERENCE**. [*S. l.*], 2012. p. 55–63.

HYNNINEN, T.; KASURINEN, J.; KNUTAS, A.; TAIPALE, O. Software testing: Survey of the industry practices. In: IEEE. **2018 41ST INTERNATIONAL CONVENTION ON INFORMATION AND COMMUNICATION TECHNOLOGY, ELECTRONICS AND MICROELECTRONICS (MIPRO)**. [*S. l.*], 2018. p. 1449–1454.

IESHIN, A.; GERENKO, M.; DMITRIEV, V. Test automation: Flexible way. In: **2009 5TH CENTRAL AND EASTERN EUROPEAN SOFTWARE ENGINEERING CONFERENCE IN RUSSIA (CEE-SECR)**. [*S. l.*: *s. n.*], 2009. p. 249–252.

JAMMALAMADAKA, K.; KRISHNA, V. R. Agile software development and challenges. **International Journal of Research in Engineering and Technology**, v. 2, n. 08, p. 125–129, 2013.

JOHANSEN, K.; PERKINS, A. Establishing an agile testing team: Our four favorite "mistakes". In: WELLS, D.; WILLIAMS, L. (Ed.). **Extreme Programming and Agile Methods — XP/Agile Universe 2002**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 52–59. ISBN 978-3-540-45672-8.

JOVANOVIĆ, I. Software testing methods and techniques. **The IPSI BgD Transactions on Internet Research**, v. 30, 2006.

KAUR, A.; KAUR, K. Systematic literature review of mobile application development and testing effort estimation. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, 2018.

KETTUNEN, V.; KASURINEN, J.; TAIPALE, O.; SMOLANDER, K. **A Study on Agility and Testing Processes in Software Organizations**. In: **Proceedings of the 19th International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2010. (ISSTA '10), p. 231–240. ISBN 9781605588230. Acesso em: 22 jun. 2022. Disponível em: https://doi.org/10.1145/1831708.1831737.

KOREL, B. Automated software test data generation. **IEEE Transactions on software engineering**, IEEE, v. 16, n. 8, p. 870–879, 1990.

KÜSTER, J. M.; GSCHWIND, T.; ZIMMERMANN, O. Incremental development of model transformation chains using automated testing. In: SCHÜRR, A.; SELIC, B. (Ed.). **Model Driven Engineering Languages and Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 733–747. ISBN 978-3-642-04425-0.

LARSON, D.; CHANG, V. A review and future direction of agile, business intelligence, analytics and data science. **International Journal of Information Management**, Elsevier, v. 36, n. 5, p. 700–710, 2016.

LARUSDOTTIR, M. K.; BJARNADOTTIR, E. R.; GULLIKSEN, J. The focus on usability in testing practices in industry. In: FORBRIG, P.; PATERNÓ, F.; PEJTERSEN, A. M. (Ed.). **Human-Computer Interaction**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 98–109. ISBN 978-3-642-15231-3.

LWAKATARE, L. E.; RAJ, A.; BOSCH, J.; OLSSON, H. H.; CRNKOVIC, I. A taxonomy of software engineering challenges for machine learning systems an empirical investigation. In: SPRINGER, CHAM. **INTERNATIONAL CONFERENCE ON AGILE SOFTWARE DEVELOPMENT**. [*S. l.*], 2019. p. 227–243.

MACLEOD, L.; GREILER, M.; STOREY, M.-A.; BIRD, C.; CZERWONKA, J. Code reviewing in the trenches: Challenges and best practices. **IEEE Software**, IEEE, v. 35, n. 4, p. 34–42, 2017.

MADAMPE, K.; HODA, R.; GRUNDY, J. A faceted taxonomy of requirements changes in agile contexts. **IEEE Transactions on Software Engineering**, IEEE, 2021.

MALL, R. **Fundamentals of software engineering**. [*S. l.*]: PHI Learning Pvt. Ltd., 2018.

MILLER, E.; HOWDEN, W. E. **Tutorial, software testing & validation techniques**. [*S. l.*]: IEEE Computer Society Press, 1981.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [*S. l.*]: John Wiley & Sons, 2011.

RALPH, P. Toward methodological guidelines for process theories and taxonomies in software engineering. **IEEE Transactions on Software Engineering**, IEEE, v. 45, n. 7, p. 712–735, 2018.

REHMAN, A. U.; NAWAZ, A.; ALI, M. T.; ABBAS, M. A comparative study of agile methods, testing challenges, solutions & tool support. In: **2020 14TH INTERNATIONAL CONFERENCE ON OPEN SOURCE SYSTEMS AND TECHNOLOGIES (ICOSST)**. [*S. l.*: *s. n.*], 2020. p. 1–5.

RODRÍGUEZ, P.; MÄNTYLÄ, M.; OIVO, M.; LWAKATARE, L. E.; SEPPÄNEN, P.; KUVAJA, P. Advances in using agile and lean processes for software development. In: **Advances in Computers**. [*S. l.*]: Elsevier, 2019. v. 113, p. 135–224.

ROMANO, S.; FUCCI, D.; SCANNIELLO, G.; TURHAN, B.; JURISTO, N. **Results from an Ethnographically-Informed Study in the Context of Test Driven Development**. In: **Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (EASE '16). ISBN 9781450336918. Acesso em: 22 jun. 2022. Disponível em: https://doi.org/10.1145/2915970.2915996.

ROMANO, S.; FUCCI, D.; SCANNIELLO, G.; TURHAN, B.; JURISTO, N. Findings from a multi-method study on test-driven development. **Information and Software Technology**, v. 89, p. 64–77, 2017. ISSN 0950-5849. Acesso em: 22 jun. 2022. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584917302550.

SAWANT, A. A.; BARI, P. H.; CHAWAN, P. Software testing techniques and strategies. **International Journal of Engineering Research and Applications (IJERA)**, Citeseer, v. 2, n. 3, p. 980–986, 2012.

SCHMIDT, C. Agile software development. In: **Agile Software Development Teams**. [*S. l.*]: Springer, 2016. p. 7–35.

SCHOOENDERWOERT, N. V.; MORSICATO, R. Taming the embedded tiger - agile test techniques for embedded software. In: **AGILE DEVELOPMENT CONFERENCE**. [*S. l.*: *s. n.*], 2004. p. 120–126.

SCHWABER, K.; BEEDLE, M. **Agile software development with scrum. Series in agile software development**. [*S. l.*]: Prentice Hall Upper Saddle River, 2002. v. 1.

SILVA, T. R.; HAK, J.-L.; WINCKLER, M. Testing prototypes and final user interfaces through an ontological perspective for behavior-driven development. In: BOGDAN, C.; GULLIKSEN, J.; SAUER, S.; FORBRIG, P.; WINCKLER, M.; JOHNSON, C.; PALANQUE, P.; BERNHAUPT, R.; KIS, F. (Ed.). **Human-Centered and Error-Resilient Systems Development**. Cham: Springer International Publishing, 2016. p. 86–107. ISBN 978-3-319-44902-9.

ŠMITE, D.; WOHLIN, C.; GALVIŅA, Z.; PRIKLADNICKI, R. An empirically based terminology and taxonomy for global software engineering. **Empirical Software Engineering**, Springer, v. 19, n. 1, p. 105–153, 2014.

STEVENSON, A. **Oxford dictionary of English**. [*S. l.*]: Oxford University Press, USA, 2010.

USMAN, M.; BRITTO, R.; BÖRSTLER, J.; MENDES, E. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. **Information and Software Technology**, Elsevier, v. 85, p. 43–59, 2017.

VALENTE, M. T. **Engenharia de Software Moderna**: Princípios e práticas para desenvolvimento de software com produtividade. [s. l.: s. n.]. 2020.

VANDERBURG, G. **A simple model of agile software processes–or–extreme programming annealed**. In: **Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**. [*S. l.*: *s. n.*], 2005. p. 539–545.

VEGAS, S.; JURISTO, N.; BASILI, V. R. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. **IEEE Transactions on Software Engineering**, IEEE, v. 35, n. 4, p. 551–565, 2009.

VESSEY, I.; RAMESH, V.; GLASS, R. L. A unified classification system for research in the computing disciplines. **Information and Software Technology**, Elsevier, v. 47, n. 4, p. 245–255, 2005.

VIEW, T. Ieee standard glossary of software engineering terminology. **IEEE std**, p. 610–12, 1990.

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

WILLIAMS, L. Agile software development methodologies and practices. In: **Advances in computers**. [*S. l.*]: Elsevier, 2010. v. 80, p. 1–44.

WOHLIN, C. **Writing for synthesis of evidence in empirical software engineering**. In: **Proceedings of the 8th acm/ieee international symposium on empirical software engineering and measurement**. [*S. l.*: *s. n.*], 2014. p. 1–4.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [*S. l.*]: Springer Science & Business Media, 2012.

ZAMPETTI, F.; VASSALLO, C.; PANICHELLA, S.; CANFORA, G.; GALL, H.; PENTA, M. D. An empirical characterization of bad practices in continuous integration. **Empirical Software Engineering**, Springer, v. 25, n. 2, p. 1095–1135, 2020.