UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO

RENATO ATOUGUIA LEITE

SAAS PROCESS:

Um Processo de Desenvolvimento para Software Como Serviço

Recife

2022

RENATO ATOUGUIA LEITE

SAAS PROCESS:

Um Processo de Desenvolvimento para Software Como Serviço

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software e Linguagens de Programação

Orientador: Profº. Dr. Alexandre Marcos Lins de Vasconcelos

Recife

2022

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

L533s Leite, Renato Atouguia
SaaS Process: um processo de desenvolvimento para software como serviço / Renato Atouguia Leite. – 2022.
94 f.: il., fig., tab.

Orientador: Alexandre Marcos Lins de Vasconcelos.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.
Inclui referências e apêndice.

1. SaaS. 2. Ágil. 3. Scrum. 4. Kanban. 5. Devops I. Vasconcelos, Alexandre Marcos Lins de (orientador). II. Título

005.1 CDD (23. ed.) UFPE - CCEN 2022 – 85

Renato Atouguia Leite

“SAAS PROCESS - Um Processo de Desenvolvimento para Software Como Serviço”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software e Linguagens de Programação.

Aprovado em: 03/03/2022.

BANCA EXAMINADORA

_____

Prof. Dr. Vinicius Cardoso Garcia
Centro de Informática /UFPE

_____

Prof. Dr. Rafael Prikladnicki
Departamento de Fundamentos da Computação/UFRS

_____

Prof.Dr. Alexandre Marcos Lins de Vasconcelos
Centro de Informática /UFPE
(**Orientador**)

AGRADECIMENTOS

Agradecer a Deus é algo muito pessoal, mas necessário quanto a sua exposição, qualquer um que se proponha a convencer alguém sobre algo, deve dar exemplo. Esse trabalho é um exemplo da infinita bondade de Deus. Obrigado primeiramente a Ele.

A minha mãe, Cristina, que num leito de hospital, com sua imensa força, tolerou a falta de minha presença naquele momento derradeiro. Obrigado mãe, você sempre foi e sempre será minha maior incentivadora a buscar algo mais.

Obrigado a Bruna, que como noiva acompanhou, tolerou e incentivou tantas idas a UFPE para cumprir as agradáveis aulas no Cin. E já como esposa, me incentivou a não desistir, mesmo diante de tantas dificuldades, perdas e mudanças nas nossas vidas. Obrigado meu amor.

Obrigado a Alexandre Vasconcelos por me receber ainda como um possível aluno, me incentivar a ser aluno especial e especialmente pela paciência, crença e incentivo a tentar o programa de mestrado dentro do tema que defendo nesse trabalho. Não poderia deixar de lembrar de forma especial dos professores Fabio Queda, Jaelson Castro e Vinicius Garcia que me marcaram como estudante trazendo um ensino de qualidade e profundidade diferenciados.

A empresa que cedeu espaço para que a pesquisa fosse executada, dados coletados e analisados e principalmente as pessoas que fizeram parte desse processo.

Obrigado ao Centro de Informática pela excelente estrutura que faz os alunos entregarem o que tem de melhor e dedicação com o mínimo de impedimentos.

Obrigado ao amigo Jefferson Barbosa e demais “caroneiros”, que estiveram comigo nas centenas de viagens ao Cin, ainda como aluno especial, colaborando e sendo verdadeiros parceiros no início dessa jornada.

Obrigado a todos que de alguma maneira não me incentivaram, vocês me fizeram pensar ainda mais sobre o propósito e decidir seguir em frente.

RESUMO

Em empresas baseadas em SaaS - Software as a Service, as mudanças são mola propulsora e estimuladora para melhores soluções, novas funcionalidades e abordagens técnicas otimizadas, diferindo de cenários tradicionais de desenvolvimento de software como projetos, onde há fronteiras e requisitos bem definidos. O cenário enfrentado no desenvolvimento de SaaS é o de reformas constantes e sem limites definidos, desde a primeira versão entregue em produção, sem prazos ou escopos conhecidos. Isso porque requisitos são alterados, reagindo-se a demandas do negócio de maneira mais intensa, exigindo entregas contínuas e imediatas. Se no início do milênio, discutia-se o manifesto ágil impulsionado pelo Scrum e sua capacidade adaptativa, hoje, as mudanças nos requisitos de entrega de software em *cloud*, a exemplo do SaaS, mudam a cultura de desenvolvimento de software e impulsionam pesquisas que discutem a estrutura de processos adaptativos de desenvolvimento. É notório que o Scrum tem sido fundamental na entrega de software com *time-to-market* adequado, mas é preciso frisar que este processo apresenta dificuldades para a gestão de mudanças constantes de *backlog* e ausência de boas práticas voltadas à programação do software, como as entregues pelo eXtreme Programming - XP. É nesse cenário que a pesquisa se propõe a adaptar o Scrum Clássico ao modelo de entrega de SaaS, enaltecendo pontos fortes desse framework e eliminando aspectos que conflitem com o cenário de entrega contínua, orientada à manutenção e evolução que o SaaS implica, como o ciclo de Sprint *TimeBoxed*. Desta forma, o objetivo da pesquisa é analisar o impacto do modelo de entrega de software como serviço baseado em Scrum e propor uma adaptação do framework, chamado SaaS Process, unindo camadas estratégicas, táticas e operacionais, considerando mudanças como algo desejado e integrando conceitos provenientes de Kanban, qualidade de código utilizando XP, entrega de software por meio de DevOPS e auto-gestão dos times por meio de OKR. Como metodologia para execução da pesquisa executou-se um mapeamento sistemático da literatura para evidenciar o uso de metodologias ágeis durante as atividades de manutenção e evolução de software, as quais estão bem presentes no desenvolvimento de SaaS:. Além disso, foi executada uma pesquisa-ação em uma empresa de software paraibana, em duas etapas, uma *ad-hoc*, anterior ao período regular do mestrado e a segunda durante o período regular, como meio de avaliação da adaptação proposta.

Palavras-chave: Saas; processo; metodologia; ágil; scrum; kanban; devops.

ABSTRACT

In SaaS-based companies, changes play a major and motivating role for better solutions, new features and enhanced technical approaches, which paints a whole new picture when compared to traditional scenarios of software development projects, because there are well-defined boundaries and requirements. The context of SaaS development has been faced with constant reforms that lack clear limits since the first version delivered in production, without deadlines or recognized scopes. Such a scenario takes place because of a change in requirements, which leads to greater intensity in regard to the demands of the business. As a result, constant and immediate delivery is required. If in the beginning of the millenium the agile manifesto of Scrum and its adaptive capacity were widely debated, nowadays the changes in cloud-based software delivery, such as SaaS, affect the culture of software development and lead to researches that aim to discuss the structure of development adaptive processes. Scrum notoriously plays a fundamental role in software development with suitable time to market, but it is necessary to emphasize the difficulties found in the management of constant changes in backlog and the lack of good practices of software programming, such as those delivered by eXtremeProgramming - XP. Therefore, the present research aims to adapt the classic Scrum to the SaaS delivery model, highlighting the strengths of this framework and putting an end to elements that interfere with the constant delivery process based on maintenance and evolution, which compose SaaS, such as the Time-boxed Sprint cycle. Thus, the purpose of this research is to analyze the impact of the delivery model of Scrum-based software as a service, as well as to propose an adaptation of the framework known as SaaS Process, merging strategic, tactical and operational layers. Moreover, it takes into account changes and integration of concepts derived from the Kanban method, code quality using XP, software delivery through DevOPS and self-management of the teams through OKR. As far as the methodology is concerned, a systematic literature mapping was carried out in order to produce evidence of the use of agile methodologies in the most common tasks in SaaS development: software maintenance and evolution. Furthermore, an action research consisting of two phases was conducted in a software company in Paraíba. The ad-hoc phase took place prior to the regular term of the master's degree program and the second phase took place throughout the regular term of the program in order to evaluate the proposed adaptation.

Keywords: Saas; processo; metodologia; ágil; scrum; kanban; devops.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

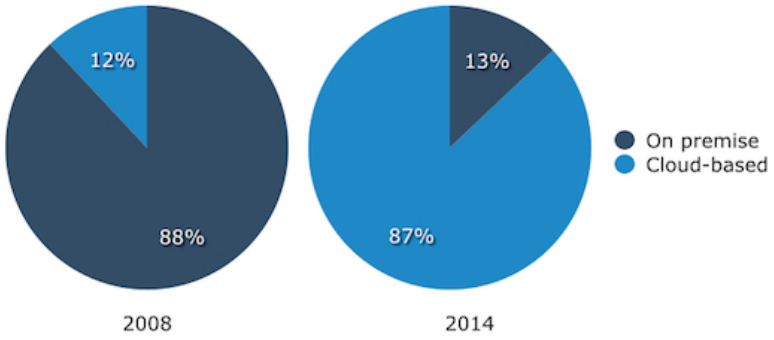
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Até algum tempo atrás, empresas de desenvolvimento de software tinham a venda de produto como a abordagem mais comum para entregar software. Este cenário é ilustrado no Gráfico 1, obtido do *report* de mercado (SOFTWARE ADVICE, 2014), que mostra que em 2008 mais de 80% das licenças eram dedicadas e perpétuas. Esse modelo foi muito comum e utilizado pela indústria por décadas, a exemplos de empresas como Microsoft®, Oracle®, SAP®, TOTVS®, entre outras. Essas empresas vendiam licenças de uso de seus softwares em conjunto a uma mídia física para instalação local. Apesar de essa ter sido a realidade por muitos anos, segundo Sommerville (2019), como os softwares são intangíveis e abstratos, esta maneira de disponibilização sob a forma de mídias físicas ou cópias digitais é algo complexo e pouco intuitivo, considerando aspectos como: instalação, configurações e atualizações que devem ser feitas pelo próprio usuário.

Gráfico 1- *Market-share Cloud vs On premise* (servidores dedicados)



Fonte: Software Advice (2014)

No entanto, este cenário vem mudando à medida que SaaS (Software as a Service) vem ganhando força. Ainda no Gráfico 1, no ano de 2014, a entrega de SaaS, baseado na computação em nuvem, se tornou predominante, com 87% do market-share sendo de software cloud-based. Segundo Gartner (p. 1, 2021), “até 2025, os 20 maiores fornecedores de software devem encerrar venda de software por licença tradicional, migrando para SaaS e aumentando seus custos internos em manutenção e entrega”. No modelo SaaS, o consumidor paga pela utilização do software de acordo com sua demanda (FITZSIMMONS e FITZSIMMONS, 2014), incluindo nesse custo a hospedagem e manutenção do sistema, ao passo que, no formato tradicional o usuário paga por uma cópia executável hospedado *on-premise* (SOFTWARE ADVICE, 2014), a um custo fixo único ou recorrente fixo.

Esta tendência de entrega de software de forma contínua e abstrata, como serviço já havia sido descrita em 2008 pelo trabalho de Goth (2008), desenvolvido com base numa entrevista com um importante diretor da Gartner. Nesse trabalho, o autor afirma que foi apontada a necessidade de se entender como os produtos são construídos no contexto de desenvolvimento de SaaS, onde o escopo é aberto e continuamente alterado.

1.1 O PROBLEMA

Métodos ágeis são utilizados majoritariamente em empresas que desenvolvem software ou tecnologia, com mais de 60% delas utilizando o framework Scrum como abordagem (DIGITAL.AI, 2021). Sutherland e Schwaber (p.4, 2020) definem o Scrum como "(...) um framework que ajuda pessoas, times e organizações a gerar valor por meio de soluções adaptativas para problemas complexos".

O fato de o Scrum ser genérico e prescritivo, abstraindo diferentes cenários de aplicação e modelos de negócio tem impulsionado adequações deste *framework* a diferentes cenários e negócios. Essas adequações são discutidas quanto à adoção em conjunto a outras metodologias ágeis, como o Kanban (LADAS, 2009), bem como discussões sobre o uso de Scrum para a integração entre diferentes áreas de uma empresa, como os times de modelagem de negócio e os times de desenvolvimento (JARIKRE et al., 2022). Diante do exposto, observa-se um cenário de hibridização de métodos, mesclando-se técnicas, e *frameworks* ágeis, indo além do Scrum puro para atender aos propósitos do negócio. Essa tendência tem influído de maneira substancial no próprio Scrum, que sinaliza na sua última versão do ScrumGuide de 2002 com menos enfase a cerimônia de cancelamento de Sprint e a introdução de metas de negócio ao framework (SUTHERLAND e SCHWABER, 2020).

Apesar do Scrum ser adequado e aplicável a cenários gerais, não possui em suas atividades elementos que atendam a camadas estratégicas e operacionais. Por conta disso, essa limitação é tratada isoladamente nas organizações utilizando elementos de outras metodologias ágeis, criando métodos híbridos ou flexibilizados sobre o Scrum. Em cenários onde há manutenção e entrega contínua, o Scrum, por vezes, força os times à inclusão de novos elementos e tarefas ao ciclo de vida de desenvolvimento, consequentemente tornando o processo de desenvolvimento informal e sem definição clara. Essas adaptações ocorrem mesmo que Sutherland e Schwaber (2020) afirmem que o *framework* é imutável e sua implementação parcial ou modificada não poder ser considerada Scrum. No cenário de SaaS, mudanças de

escopo e a necessidade de entrega contínua ferem o Scrum puro, pois o mesmo exige a execução total da Sprint ou o seu cancelamento em caso de mudanças no backlog do produto, não atendendo ao cotidiano da entrega como serviço. Adicionalmente, o framework presume que apenas ao final da Sprint, durante a cerimônia de review, o software deve ser entregue. Essa é mais uma situação que não atende, mais uma vez, ao modelo SaaS, onde o software entregue muitas vezes em cenários onde "alguns times liberam melhorias diariamente" (HENDERSON, p.10, 2017).

Existe em empresas de SaaS, na maior parte do tempo do desenvolvimento de soluções a evolução e correção de softwarem, num cenário de manutenção do software predominante. E existem lacunas em seus processos de desenvolvimento ligados à velocidade de atendimento de melhorias, evoluções e correções de seus serviços. Jagli e Yeddu (2017) afirmam que uma abordagem de desenvolvimento que privilegie o atendimento aos *Service-level-agreements,* priorizando a manutenção (correção ou evolução) dos softwares em cloud são fundamentais para que o SaaS seja entregue. Dessa maneira, o negócio não é atendido por métodos ágeis isolados e orientados a projetos, como o Scrum. Dentre as lacunas mapeadas, se destacam:

- Documentação superficial;
- Códigos com baixa aderência a boas práticas de codificação;
- Atendimento às mudanças de prioridade motivadas por melhorias movidas por *feedbacks* de insatisfação do usuário quanto a bugs, ausência de características, a grande concorrência por usuários no oferecimento de funcionalidades mais sofisticadas e promoções;

No contexto de SaaS, problemas ligados à manutenção, tais como entregas com falhas, fora do prazo ou ainda entregas que diferem do serviço esperado pelo cliente, podem levar à insatisfação do usuário (MOREIRA, 2015, p. 106). Nesse cenário, aliado ao fato de que o uso do software é pago sob demanda, a insatisfação do usuário pode gerar cancelamentos e perda de faturamento imediato. Além disso, ainda existe a necessidade de tratar fatores relacionados à gestão da equipe para que o serviço seja prestado de forma contínua e com maior qualidade. Moreira (2015) destaca alguns destes fatores:

- Baixa motivação das pessoas: em empresas focadas em SaaS, os colaboradores são submetidos a um processo de desenvolvimento pouco orientado a objetivos, e muito mais voltado à operação cotidiana, sem percepção clara de escopo entre entregas;

- A falta de entendimento do sistema, originada pela documentação superficial, aliada à complexidade do software legado dificulta a entrega de melhorias corretivas;
- Falta de integração entre novos requisitos e manutenção do software: ter um sistema desacoplado, com responsabilidades bem definidas entre módulos, seguindo premissas de micro serviços e cultura DevOps, com interfaces entre API bem definidas favorece a evolução do software. Essa é uma característica por diversas vezes de difícil implementação em empresas com software legado, em operação, deve ser considerada pois é fundamental para a evolução do serviço.

Em relação ao uso do Scrum, no contexto do desenvolvimento de SaaS, alguns problemas podem ser citados:

- Inadequação da cerimônia de Cancelamento de Sprint ao contexto de mudanças de escopo, prioridades e estratégias de negócio voláteis;
- Ausência de métodos de entrega contínua que não conflitem com a cerimônia de *Sprint-review* ao final da Sprint;
- Ligação entre a estratégia do negócio e diretrizes para entrega automatizada de código com qualidade.

Essas questões têm sido endereçadas pela academia e pelo mercado através do uso de processos híbridos (LADAS, 2009), bem como pela flexibilização de diversos métodos ágeis, a exemplo do Scrum (SUTHERLAND e SCHWABER, 2020). Este trabalho, portanto, busca contribuir para essa discussão.

1.2 OBJETIVO

De modo a abordar os problemas relatados anteriormente, este trabalho tem como objetivo adaptar o Scrum com foco no fornecimento de Software como Serviço – SaaS, considerando a ausência de pesquisas relacionadas a metodologias de desenvolvimento de software ágeis específicas para SaaS, constatada a partir da realização de um mapeamento sistemático da literatura – MSL.

O Scrum adaptado, chamado SaaS Process, foi constituído a partir da aplicação de ciclos de pesquisa-ação em uma empresa de software no estado da Paraíba. A versão final foi então aplicada ao longo de dois anos, nos anos de 2019 e 2020.

Como objetivos secundários pretende-se:

- Mapear na literatura as dificuldades em torno do tema;

- Utilizar uma metodologia de pesquisa que permita o diagnóstico, a aplicação e a avaliação de novos processos em um ambiente real de uma companhia de software;
- Medir os resultados da aplicação da adaptação do Scrum por meio de métricas tradicionais para acompanhamento de performance de times.

1.3 ESTRUTURA DO TRABALHO

A estrutura do trabalho possui capítulos que contemplam três etapas correspondentes à metodologia definida e explanada em detalhes no próximo capítulo. São elas: diagnóstico, aplicação e avaliação . Essa é uma abordagem de organização *top-down* e busca dar ao leitor a oportunidade de acompanhar a definição do problema, construção da possível solução e sua confirmação junto ao ambiente externo ao ambiente da pesquisa.

Para documentar a pesquisa, além deste capítulo introdutório, o trabalho é formado pelos seguintes capítulos:

No capítulo 2 é apresentada a metodologia utilizada para a realização do trabalho, com a exposição das técnicas e métodos utilizados para condução da pesquisa.

No capítulo 3 é apresentado o referencial teórico da pesquisa, considerando aspectos fundamentais da engenharia de software, metodologias ágeis e conceitos relacionados à entrega de serviços, no modelo SaaS e suas arquiteturas correspondentes.

No capítulo 4 é realizado o detalhamento acerca da MSL, trazendo o desenho do mapeamento, incluindo sua estrutura de busca, critérios de seleção e conclusão. Esse capítulo tem papel fundamental para pesquisa por corroborar com a ideia defendida pelo autor a respeito da dificuldade do mercado em lidar com serviço e métodos ágeis tradicionais, apresentando os resultados da MSL e a ausência de pesquisas correlatas ao tema do trabalho.

O capítulo 5 indica uma possível solução ao cenário observado na empresa alvo do estudo de caso e corroborado pela MSL, onde a dificuldade em se utilizar metodologias tradicionais e isoladas traz lacunas ao processo de desenvolvimento, citadas na seção 1.1, frente às necessidades de negócio. Dessa forma, foram definidos os elementos que compõem a proposta de adaptação do Scrum e o seu formato final.

O capítulo 6 apresenta os recortes da pesquisa-ação ao longo de três iterações para a melhoria do processo de software proposto. Foram definidas métricas avaliativas para comparar cada recorte e os efeitos positivos e negativos de cada mudança.

O capítulo 7 tem o intuito de confrontar as conclusões obtidas nos capítulos anteriores, apresentando os resultados de uma pesquisa externa, utilizando-se questionário com os desenvolvedores de diferentes regiões e mercados. O objetivo desse capítulo é finalizar com as impressões positivas ou negativas sobre o uso do Scrum adaptado voltado ao SaaS.

O capítulo 8 apresenta as conclusões do trabalho, limitações da pesquisa e sugestões de trabalhos futuros.

2 METODOLOGIA DE PESQUISA

É importante compreender o contexto da organização, os propósitos dos indivíduos e como as técnicas, as ferramentas e os processos interagem para formar os processos de desenvolvimento de software, as atividades de melhoria de processo de software (MPS) e o comportamento dos atores dentro do processo. Segundo Easterbrook (2008), a importância do comportamento humano no processo de desenvolvimento de software é tão relevante que métodos de pesquisa foram adequados ao entendimento dos aspectos sociais e psicológicos envolvidos. Dessa forma, a metodologia de pesquisa utilizada nesse trabalho será detalhada neste capítulo.

2.1 QUADRO METODOLÓGICO

Segundo Bhattacharya (2008), a observação é a forma mais comum de se avaliar o resultado de uma pesquisa, gerando conhecimento sobre os fenômenos em estudo. Desta forma, o presente trabalho visa dar continuidade a uma pesquisa informal que vem sendo conduzida desde 2012.

Os primeiros quatro anos da pesquisa foram conduzidos de forma *ad-hoc* (período no qual o pesquisador não possuía vínculo acadêmico junto ao centro de informática), através da observação das atividades de Melhoria de Processo de Software - MPS em suas duas primeiras fases. Posteriormente, por cerca de 2 anos, a condução dos estudos foi realizada enquanto o autor deste trabalho era aluno ouvinte do CIn-UFPE, utilizando-se dos métodos citados nesse capítulo. Por fim, durante 2 anos no período de mestrado, foi aplicada e concluída a pesquisa *in-loco* na empresa alvo.

Para a realização da pesquisa foram utilizadas ferramentas de apoio à coleta de dados como entrevistas, ferramentas de controle de produção - *PivotalTracker* -, registro de Daily-meetings e eventos *Scrum*. Através do uso destes métodos foi possível avaliar todas as etapas da construção de um processo híbrido de desenvolvimento de software, concebido pela junção das principais ferramentas e frameworks de metodologias ágeis.

No momento em que este documento foi escrito, o estudo passava pela implantação de um terceiro MPS, previsto para ser finalizado em um ano após o encerramento do período

regular do programa de mestrado. Essa terceira iteração, focada na inclusão de uma ferramenta de gestão de demandas para camadas estratégicas, será apresentada nos capítulos seguintes.

O quadro 1 apresenta as principais características da metodologia de pesquisa utilizada neste trabalho. Ao longo do texto, serão indicadas as motivações e justificativas para cada escolha. Ao final deste capítulo, é apresentado um gráfico com o ciclo de vida de toda a pesquisa com intuito de resumir e facilitar o entendimento ao leitor.

Quadro 1 - Quadro metodológico

Quadro Metodológico	
Concepção Filosófica	Pragmática (EASTERBROOK, 2008)
Natureza	Qualitativa e Quantitativa (MARCONI e LAKATOS, 2010)
Abordagem	Indutiva (MARCONI e LAKATOS, 2010)
Método de Pesquisa	Pesquisa-Ação
Coleta de Dados	Mapeamento sistemático da literatura, Observações e Ferramentas de apoio à produção.
População	Empresa Privada aderente ao modelo SaaS
Amostragem	Intencional de caso típico (EASTERBROOK et al, 2008)
Análise e Síntese	Interpretativa

Fonte: O autor (2021)

2.1.1. Concepção Filosófica

A pesquisa possui como objetivo principal o desenvolvimento de uma adaptação do framework Scrum voltado ao modelo de negócio SaaS. Sendo assim, conforme afirma Easterbrook (2008), pode ser considerada pragmática, já que o seu produto é criado para auxiliar a resolver problemas práticos.

2.1.2 Natureza da Pesquisa

A pesquisa possui natureza majoritariamente qualitativa buscando entender como o processo de desenvolvimento influi na qualidade, ritmo e atendimento de requisitos levantados pelos clientes. Segundo Merrian (1998), a natureza qualitativa de uma pesquisa está associada à compreensão de como experiências individuais se comportam dentro de um contexto estudado. Para coleta e análise de dados, foram utilizados como apoio métodos quantitativos, objetivando concretizar e documentar os resultados das intervenções realizadas.

O uso combinado das duas abordagens (quantitativas e qualitativas), conforme evidencia o quadro metodológico, torna o entendimento de comportamentos dos indivíduos, seja de times ou de clientes, um ponto relevante para a avaliação do presente estudo. Para isso serão usadas observações e alterações do ambiente, através das melhorias de processo de software, inclusão de atividades no processo de desenvolvimento e análise dos seus efeitos. Desta forma, as fraquezas de um método podem ser compensadas pela força de outro (CRESWELL apud EASTERBROOK, 2008). Serão levantados dados quantitativos relacionados às MPS, em conjunto com análises comportamentais qualitativas dos indivíduos pois como afirma Easterbrook (2008), o principal objetivo é usar resultados quantitativos para explicar os dados qualitativos.

2.1.3 Abordagem

A abordagem científica dessa pesquisa é indutiva, visto que, através de um estudo de caso e da pesquisa-ação buscou-se generalizar os resultados observados, de modo a que o processo de desenvolvimento de software desenvolvida seja aplicável ao contexto de SaaS como um todo. Segundo Marconi e Lakatos (2010, p. 86), “Indução é um processo por intermédio do qual partindo de dados particulares, suficientemente constados, infere-se uma verdade geral [...]”.

Considerando que o presente trabalho utiliza, em sua coleta de dados, diferentes contextos ao longo de anos, o autor busca avaliar tudo o que foi empiricamente constatado através de diferentes fontes de dados, utilizando, por exemplo, questionários (MARCONI e LAKATOS, 2010), métricas de desenvolvimento de software e tempo de vida de cada versão da MPS.

2.1.4 Método de Pesquisa

As metodologias de pesquisa voltadas a mapear o estado da arte, validar soluções e avaliar resultados, tais como RSL, MSL e Pesquisa-Ação, são comuns e advêm principalmente da área de saúde. O que todas estas metodologias possuem em comum é o caráter empírico dos estudos, mapeando a trajetória entre a identificação do problema e a criação de uma solução.

A diversidade de metodologias de pesquisa permite uma escolha adequada baseada no contexto do estudo, a exemplo de pesquisas de campo em que métodos como pesquisa-ação e

estudos de caso auxiliam no entendimento e transformação dos contextos em que estão inseridos, por serem de cunho prático e criarem conhecimento científico baseado nas experiências de intervenção no mundo real (SUSMAN e EVERED, 1978).

Em ambientes naturais, ou seja, aqueles onde não há condições de realização de um experimento, os estudos se prevalecem dos eventos naturais para embasar sua intervenção e medir os resultados. As metodologias oriundas de áreas como a de saúde, buscam entender os meios, buscando os porquês e não "os quês". Já em estudos de ambientes artificiais prevalece o foco nos objetivos e criações do ser humano.

É importante frisar que considerando um cenário de entendimento do problema, avaliação do percurso da solução e do objetivo proposto para o final do processo, surgiram vieses para o pesquisador, sendo assim necessário conduzir a pesquisa através de multi-métodos, onde o principal dos métodos será o estudo de caso. A abordagem de utilização de mais de um método ajuda a preencher as possíveis lacunas entre o estudo de caso e a pesquisa-ação.

Dessa forma, o autor considerou o estudo de caso para constatação em campo da necessidade mapeada durante a condução de um Mapeamento Sistemático da Literatura - MSL. O estudo de caso enfatiza a abordagem qualitativa por meio de uma generalização interpretativa do *case* observado e das evidências coletadas nos demais métodos.

A observação e mapeamento do problema da pesquisa se deu numa empresa, como uma fase preliminar à pesquisa-ação, de forma a obter dados do contexto de mercado sobre o tema pesquisado, com ênfase na confirmação do observado no MSL. Conforme Easterbrook (2008), observações são parte de estudos de caso, pois permitem entender o tema da pesquisa, tornando-o apto a confirmar o observado na fase preliminar e gerar insumos necessários à intervenção. Conforme afirma Yin (2005), em situações em que o ambiente se confunde com o fenômeno estudado, se torna difícil ao pesquisador separar a causa do sintoma, como nessa pesquisa em que o risco de uma distorção entre as dificuldades inerentes à maturidade do processo da empresa alvo e da imaturidade dos times, tornava difícil ao pesquisador a distinção entre ambos, sendo esse um risco assumido nesse trabalho.

Após as conclusões relativas à observação, foram aplicados três ciclos de Pesquisa-ação afim de alterar o processo de desenvolvimento de software aplicado na empresa alvo desse trabalho. Como afirmam Susman e Evered (1978), a pesquisa-ação é uma abordagem que permite melhorar a performance de uma empresa utilizando-se uma intervenção em ciclos. Runeson e Höst (2009) afirmam que ela é uma boa abordagem no que diz respeito à inclusão

da pesquisa como elemento ativo ou agente de mudança no ambiente de desenvolvimento de software.

Um dos objetivos desta pesquisa é gerar e avaliar um novo processo de desenvolvimento. Por essa razão, usou-se a pesquisa-ação para balizar os ciclos de criação de uma nova abordagem apta ao modelo de serviço, com entregas contínuas e uma estrutura leve para empresas e equipes que utilizam soluções de software *in-house* no modelo SaaS.

Os ciclos utilizados para a pesquisa-ação foram baseados naqueles indicados no artigo de Susman e Evered (1978), sendo eles: (i) análise inicial, (ii) intervenção e (iii) avaliação. Todos juntos formando um ciclo completo, que se repetiu na pesquisa por duas vezes, em dois anos. Um exemplo de uso desse ciclo pode ser visto no artigo de Heeager e Rose (2015), onde se busca uma melhoria no Scrum voltada à manutenção de software.

2.1.5 Coleta de Dados

Para tornar a pesquisa menos enviesada, foram utilizados multi-métodos para a coleta de dados, como descrito a seguir:

- Mapeamento Sistemático: permite que sejam encontrados padrões no estado da arte do tema escolhido, bem como lacunas de pesquisa (KITCHENHAM, 2004). Assim, abrem-se possibilidades de novos estudos primários (como este) a fim de preencher as lacunas encontradas;
- Observações: foram feitas observações de ciclos completos de entrega em equipes de desenvolvimento na empresa alvo a fim de constatar as dificuldades na gestão, desenvolvimento e entrega de software em modelos SaaS.
- Ferramentas de apoio ao desenvolvimento: coletou-se dados para cálculo de métricas de desenvolvimento de software no período da pesquisa. A ferramenta foi o PivotalTracker®.

Através do conjunto de métodos de coleta de dados acima elencados, foram identificadas características da população e documentado o ambiente de uma empresa com foco em SaaS. Para isso, foram utilizadas métricas de desenvolvimento ágil detalhadas ao longo do trabalho e dados foram coletados e cruzados, buscando mitigar os riscos à validade do estudo.

2.1.6 População

A população foi escolhida intencionalmente utilizando o critério de caso típico (EASTERBROOK et al., 2008; MERRIAM, 2009), ou seja, uma empresa cujo *core-business* é o setor de serviços e que utiliza software desenvolvido *in-house* para ofertar sua solução ao mercado.

Esse público-alvo se caracteriza pela necessidade de entregar software com rapidez e qualidade compatível com o contrato sobre demanda, no qual o cliente remunera de forma proporcional ao seu uso (demanda), e também por possuir equipes de desenvolvimento de software para atender clientes que demandam melhorias na entrega das funcionalidades com *time-to-market* agressivos.

2.1.7 Amostragem

A empresa alvo onde foi realizada a observação e a pesquisa-ação se situa na Paraíba, com cerca de quarenta colaboradores, dos quais aproximadamente trinta eram pessoas envolvidas no processo de desenvolvimento de software diretamente.

2.1.8 Análise e Síntese

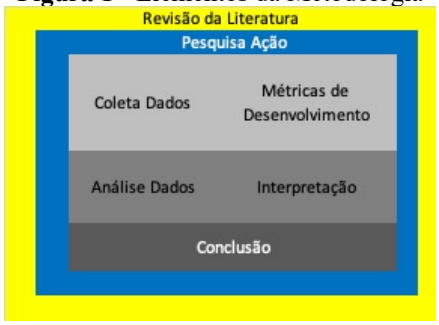
Os dados coletados ao longo da pesquisa foram compilados e abordados sobre uma ótica interpretativa. Segundo Merrian (1998), os métodos qualitativos são indicados em contextos em que o envolvimento de pessoas e suas mais distintas nuances sociais devem ser consideradas, de tal forma que a abordagem interpretativa auxilia o pesquisador a decodificar e traduzir os efeitos das intervenções sobre o ambiente em que se intervém.

Na pesquisa esse cenário é presente e se manifesta na performance dos times em diferentes momentos do tempo, sob a égide da experiência, relações pessoais entre os componentes e contextos de mercado, a exemplo da imensa demanda por mão de obra, causadora de um intenso turnover e fragilização das relações entre componentes dos times.

2.2 DESENHO DA PESQUISA

Considerando tratar-se de uma pesquisa baseada em mais de um método, utilizando-se ferramentas híbridas de coleta de dados, um resumo da metodologia utilizada na pesquisa é apresentado na Figura 1, abaixo:

Figura 1 - Elementos da Metodologia



Fonte: O autor (2021)

Como ilustrado, as etapas da pesquisa iniciaram-se com a revisão da literatura para fins de diagnóstico, onde foi detectada a falta de adequação das metodologias ágeis tradicionais para atendimento do modelo de SaaS, sendo manutenção o núcleo do desenvolvimento e entrega contínua com qualidade e competitividade pré-requisito do modelo. Em sequência, foi proposta e aplicada uma adaptação do Scrum para atendimento do modelo SaaS em três iterações, sendo essas etapas de aplicação e intervenção e ano final, coletados os dados e avaliado seus efeitos sob a ótica das métricas tradicionais de desenvolvimento de software ágil.

Foi executado um mapeamento sistemático da literatura tendo como foco o tema "Metodologia ágil e manutenção de software", visto que a característica principal da entrega de serviço por software – SaaS – é a constante evolução do software, entendida como manutenção evolutiva. O termo SaaS não foi utilizado no mapeamento sistemático, pois numa busca preliminar não foram obtidas ocorrências de trabalhos suficientes para a execução do protocolo de pesquisa completo. Dessa forma, este termo foi substituído por manutenção de software, a qual está fortemente ligada ao contexto de SaaS. As questões de pesquisa a serem respondidas pelo mapeamento foram as seguintes:

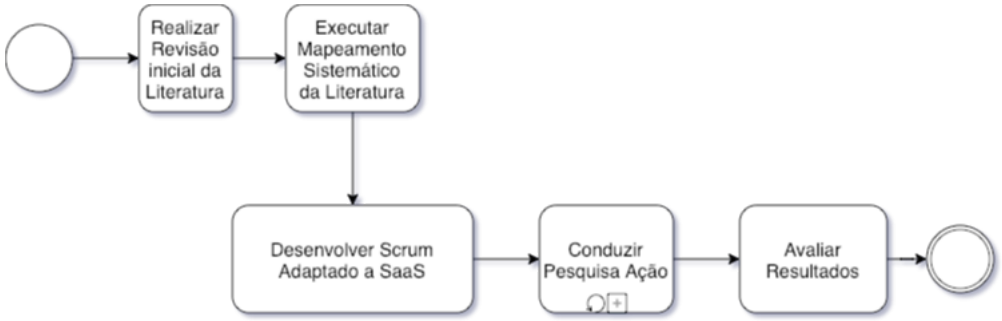
- Existe na literatura referências sobre manutenção e métodos ágeis?
- Há um ou mais métodos ágeis que se adequem à manutenção de software considerando suas variações e características?
- Qual o resultado em termos de performance dos times com o uso de uma metodologia ágil híbrida voltada à manutenção, especificamente no cenário de SaaS?
- Como o mercado enxerga os desafios na conciliação dos métodos ágeis tradicionais e manutenção, especificamente no cenário de SaaS?

As evidências apontadas por meio da MSL se mostraram presentes na observação do cenário da pesquisa-ação e coleta de dados ao longo da aplicação das Melhorias de Processo

de Software - MPS, através de ferramentas de controle de processos ágeis, como o PivotalTracker®, no âmbito de manutenção em SaaS.

A utilização de uma abordagem multi-métodos (HESSE-BIBER, 2010) visa minimizar os possíveis vieses trazidos pelo autor e pela própria característica dos métodos empíricos, principalmente os estudos de caso (EASTERBROOK, 2008). Esta abordagem é ilustrada na figura 2, que apresenta o fluxo de utilização dos métodos, como descritos a seguir.

Figura 2- Desenho da Pesquisa



Fonte: O autor (2021)

1. **Realizar revisão inicial da literatura:** buscar a bibliografia base sobre manutenção de software combinada ao uso de métodos ágeis. O intuito foi embasar por meio de literatura básica de ES as características de um processo de desenvolvimento de software, suas etapas e como as metodologias ágeis lidam com esse arcabouço de manutenção. Essa etapa forneceu ao trabalho a fundamentação teórica no campo da engenharia de software;
2. **Executar Mapeamento Sistemático da Literatura:** aplicar a busca sistemática de trabalhos em torno do tema da pesquisa. Esta etapa teve como principal objetivo a identificação de lacunas de pesquisa nos estudos relacionados a SaaS em conjunto com a engenharia de software baseada em métodos ágeis;
3. **Definir Uma adaptação do Scrum para SaaS:** descrever uma abordagem adaptada de desenvolvimento de SW voltada a software como serviço. A adaptação compreende todas as etapas de desenvolvimento de software, desde o levantamento de requisitos e controle de *backlog* até o *deploy* do serviço, ou seja, o software desenvolvido. Inclui-se também elementos de gestão estratégica (OKR). Além disso, no capítulo referente à pesquisa-ação, são detalhados seus efeitos colaterais, pontos fortes e uma proposta de aplicação para empresas que já aplicam metodologias ágeis.
4. **Executar Pesquisa-ação:** tem o objetivo de aplicar atividades de Melhoria de Processo de Software, em três ciclos, com enfoque no fornecimento de serviços através de software na

empresa alvo. O objetivo dessa etapa foi avaliar a aplicabilidade das ações de melhoria no ambiente produtivo, visualizar os ganhos e perdas em torno da aplicação de dois ciclos de MPS e documentar as lições aprendidas em cada ciclo. Os insumos obtidos na pesquisa-ação permitiram a construção de uma adaptação do Scrum voltada a SaaS, como descrito na próxima etapa;

5. **Avaliar resultados:** através da comparação de dados levantados por meio do sistema de apoio e rastreamento PivotalTracker®, de antes e depois do período regular do mestrado, foram comparadas e discutidas métricas de performance de desenvolvimento de software.

2.3 AMEAÇAS À VALIDADE

Estudos empíricos devem possuir requisitos de validação a serem satisfeitos para que tenham contribuição para a comunidade científica (EASTERBROOK, 2008). Alguns tipos de validação apontados como imprescindíveis para a generalização são a validade interna, a validade de construção e a validade externa.

A validade interna é aquela que busca evidenciar se a relação existente entre os temas da intervenção e seus efeitos são legítimos ou não, quando puderem ser causados por fatores externos não previstos ou observados (TRAVASSOS; GUROV; AMARAL, 2002). A exemplo neste trabalho das Melhorias de Processo de Software – MPS, que foram realizadas e que contribuíram para cumprimento dos objetivos traçados da pesquisa, em ambientes que os resultados possam ter sido influenciados por fatores comportamentais dos times envolvidos.

A validade de construção busca evitar vieses da parte dos participantes das amostras, considerando que é natural que ao serem observados e medidos, os participantes se comportem e respondam às questões da forma como acreditam que devam (TRAVASSOS et al., 2002).

A validade externa é a busca pela possibilidade de generalização. Nesse contexto, a amostra deve ser trabalhada para ser a mais representativa possível, para que os comportamentos ou efeitos observados sejam válidos para toda a população.

Sendo assim, com o objetivo de mitigar os riscos à generalização (validade externa), garantir o isolamento do problema de pesquisa (validade interna) e a interpretação correta dos dados coletados na pesquisa-ação, as seguintes técnicas foram aplicadas para que a subjetividade e o viés que a pesquisa carrega fossem minimizados:

- Garantir, por meio de relatórios, que os participantes enxerguem os resultados da pesquisa da mesma forma que os pesquisadores irão reportá-los. Dessa forma, busca-se

estabelecer uma forma mais robusta de avaliação da solução proposta sob a ótica da aplicação das metodologias ágeis no contexto do modelo SaaS;

- Buscar um contato constante com os atores externos que utilizem SaaS, observando-os em momentos distintos e em diferentes contextos. O objetivo dessa estratégia é auxiliar a validade externa daquilo que se conclui do estudo, cruzando os resultados desta pesquisa com outros de mesma temática;
- Deixar claro os possíveis vieses e mensurar seu impacto sobre os resultados, com rico detalhamento sobre o que foi alterado no processo de software orientado a SaaS, de modo a garantir que o que está foi pesquisado corresponde à realidade e não apenas a um sentimento do pesquisador.
- Utilizar e interpretar métricas de desenvolvimento de software ágil amplamente difundidas, tais como lead-time, ritmo e quantidade de entregas (KUPIAINEN; MÄNTYLÄ; ITKONEN, 2015), assegurando que a pesquisa atende ao mínimo necessário à validade de construção.

2.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Em busca de diminuir possíveis vieses na pesquisa foram compiladas diferentes abordagens metodológicas para criação de um multi-método, o mais robusto possível, de forma a minimizar e complementar pontos fracos de cada método isoladamente.

Em se tratando de um trabalho com pesquisa de campo, aspectos qualitativos devem ser considerados. Contudo, buscou-se pautar a pesquisa em métricas e coleta de dados de maneira objetiva, além de compilá-las através de métodos que tragam o mínimo de subjetividade.

Desse modo, espera-se que a pesquisa seja reconhecida em sua tentativa de ter validade interna e externa para a academia em torno de um tema importante e latente para a entrega de software com qualidade e agilidade desejadas.

3 REFERENCIAL TEÓRICO

Desenvolver soluções baseadas em software exige das empresas investimento em Engenharia de Software (ES), conforme a natureza da solução e a forma como ela é entregue: produto, serviço ou encomenda – fábrica de software com transferência de tecnologia. Assim, a ES deve se adequar ao negócio e aos softwares desenvolvidos (SOMMERVILLE, 2019).

É natural que devido ao caráter abstrato do software haja poucos limites para seu potencial crescimento e complexidade. Contudo, não havendo técnicas de ES robustas, o efeito percebido é a entropia de código, baixa qualidade de entregas, quebra de acordos sobre requisitos, além de pressão sobre prazos curtos, escopos de grande magnitude e recursos escassos, dificultando assim a compreensão e evolução dos softwares (CANFORA et al., 2014). Desenvolver software para diferentes modelos de comercialização ou áreas exige diferentes técnicas (SOMMERVILLE, 2019), como na temática desta pesquisa que versa sobre o desenvolvimento de soluções por meio de SaaS, considerado na literatura como um software instalado remotamente, utilizado sob demanda via internet e com custo baixo de evolução (OLIVEIRA *et al.*, 2019) – um só software com dados isolados - e escalável sob demanda.

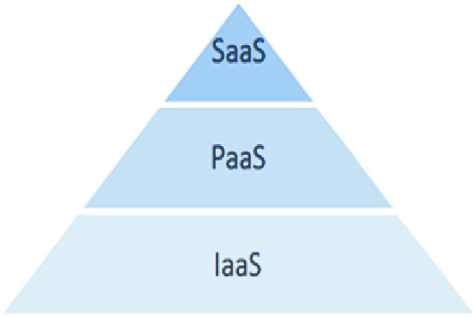
Nesse capítulo será tratado o desenvolvimento de SaaS, embasando sob a ótica da manutenção de software. Serão tratadas também as diferentes e as mais comuns metodologias de ciclo de vida de software (SOMMERVILLE, 2019) utilizadas em todo na indústria. São elas: cascata, incremental, ágil e orientada a reuso (CMS, 2005). E assim, embasar a discussão sobre uma adaptação do Scrum, voltando o trabalho a criação de um ciclo de vida de desenvolvimento de software adaptativo voltado a SaaS.

3.1 SOFTWARE AS A SERVICE - SAAS

Para explicar SaaS é preciso citar e detalhar a pirâmide *as a service* (GARTNER, 2009), que além do próprio software define também a plataforma e a própria infraestrutura de entrega como serviço. A Figura 3 ilustra esse formato da entrega de serviços citado por Gartner (2009), sendo a infraestrutura como serviço (IaaS) base para plataformas (PaaS) e no topo o SaaS. Assim, abstrai-se da camada mais superior da solução a complexidade de entrega e manutenção provida pela camada inferior, variando a entrega mediante a entrega de serviço conforme demanda da camada superior. Dessa maneira, a Platform as a Service - PaaS abstrai a entrega de ferramentas para provimento do SaaS, ao ponto que a IaaS entrega capacidade

computacional sob demanda para diferentes plataformas, com total transparência de gestão, entrega e manutenção do seu serviço.

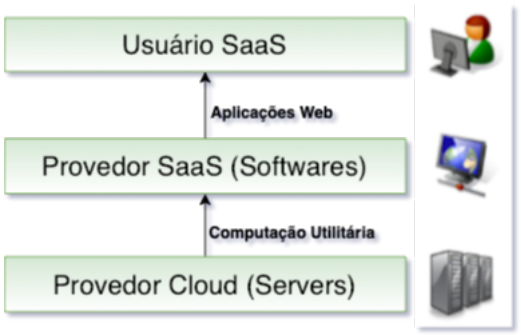
Figura 3 – Pirâmide as a Service



Fonte: Gartner (2009)

Esse desenho é a concretização de um modelo de computação utilitária que emergiu há cerca de 10 anos no mercado (ARMBRUST et al., 2009). Quanto a utilitária, se explica como uma cadeia de processamento, popularmente conhecida como *cloud,* compartilhada e utilizada através da internet sob a forma de SaaS (ARMBRUST et al., 2009). A Figura 4 ilustra esse cenário de entrega, na visão dos usuários, sendo a camada SaaS um utilizador do software entregue por meio da internet, provido em uma plataforma – PaaS e hospedada em uma IaaS, fornecedora de capacidade computacional.

Figura 4 – Estrutura de entrega As a Service



Fonte: Adaptado Armbrust et al. (2009)

Empresas têm decidido oferecer seus softwares por meio de SaaS por trazerem vantagens quanto à escala, por meio da pirâmide *as a service,* com oferta de infraestrutura sob

demanda (OLIVEIRA *et al.*, 2019), tendo um custo de hospedagem menor que em soluções *in-house* (CHO; CHAN, 2015) e consequentemente melhor serviço prestado.

SaaS além de possuir um modelo de negócio próprio, também se associa a um processo de desenvolvimento único (TSAI; BAI; HUANG, 2014), onde a manutenção de software é de responsabilidade única do provedor (CHO; CHAN, 2015) e que também é a principal atividade dos times de desenvolvimento. Sabe-se também que a manutenção é a atividade mais custosa do ciclo de vida de um produto de software (AZIZ; AHMED; LAGHARI, 2009), sendo alvo de diversas pesquisas empíricas na engenharia de software. Assim, empresas de software que trabalham com SaaS, e naturalmente fazem das rotinas de manutenção o cotidiano do seu trabalho, se deparam cada vez mais com desafios em evoluir e manter seus softwares. Esses desafios quando não tratados ou endereçados podem causar problemas de disponibilidade (OLIVEIRA *et al.*, 2019) e *time-to-market,* refletindo-*se* em insatisfação dos usuários.

3.2 MANUTENÇÃO DE SOFTWARE E SAAS

O processo de evolução, correção de defeitos e entrega do SaaS exige das organizações que a manutenção em todas suas nuances seja o principal trabalho dos times de desenvolvimento. Segundo Aziz, Ahmed e Laghari (2009) existem três tipos básicos de manutenção: a corretiva, a adaptativa e a preventiva. A manutenção corretiva tem o objetivo de corrigir falhas que podem levar o software a ter um comportamento inesperado ou em contradição com suas especificações. A manutenção adaptativa recai sobre alterações nos requisitos do usuário para atender novas demandas do negócio. Por fim, a manutenção preventiva é realizada para tratar problemas que possam se manifestar no futuro durante a operação do software.

A manutenção do software tem sido uma área chave de pesquisa devido ao fato desta ser considerada a etapa mais custosa e crítica do ciclo de vida do software (AZIZ; AHMED; LAGHARI, 2009; ZHOU e LEUNG, 2007; LI e HENRY, 1993). Esta fase chega a consumir entre 60 e 80% do custo total do ciclo de vida de um produto de software (PRESSMAN, 2011; AZIZ; AHMED; LAGHARI, 2009). Pfleeger e Bohner (1990) acrescentam ainda que o retrabalho chega a representar 40% do custo de manutenção em projetos de software.

Os motivos da alta complexidade e dos elevados custos da fase de manutenção do software têm sido objeto de estudo de muitos pesquisadores. Pfleeger e Bohner (1990) sugerem

que o problema ocorre porque o desenvolvimento de software muitas vezes é executado sem considerar como futuras mudanças serão implementadas.

Segundo Moreira (2015), durante o processo de manutenção pode acontecer a insatisfação dos usuários por diversos motivos, tais como: entregas feitas com falhas, estouro de cronograma e requisições de mudança entregues diferente do que estava definido no requisito de mudança. Estudos sugerem que práticas ágeis trazem muitos benefícios para a manutenção de software (AHMAD et al., 2016) e, portanto, muitos destes problemas já poderiam ser mitigados ou mesmo eliminados com o uso dos métodos ágeis de desenvolvimento de software.

Embora os benefícios advindos da adoção desses métodos ágeis pareçam se aplicar perfeitamente como solução para os problemas enfrentados pela manutenção de software, vários autores afirmam que a aplicação deles nesse contexto ainda não foi estudada em detalhes (HEEAGER; ROSE; SUMAN, 2010).

3.3 MÉTODOS ÁGEIS

O Manifesto Ágil (FOWLER e HIGHSMITH, 2001), em seus mais de 20 anos de existência deu origem a cerca de 50 métodos ágeis (SALEH, 2021), alguns desses derivados de outros já existentes, a exemplo do XP ou métodos híbridos, como Crystal, *Lean* e ScrumBan (FOTIADIS, 2021), sendo todos estes métodos formados por um conjunto de técnicas e estratégias que juntas buscam empoderar os times de desenvolvimento, priorizar as pessoas ao processo e entregar valor em cada versão do software desenvolvido. A busca por entregas em intervalos de tempo cada vez mais curtos e em menores incrementos de código tem como principal objetivo entregar valor com agilidade aos clientes finais, aproximando os times aos clientes e promovendo uma comunicação “olho-no-olho” (AHMAD et al., 2016; PINO et al., 2012; HEEAGER e ROSE, 2015). Neste trabalho, focaremos nos métodos ágeis Scrum (AHMAD et al., 2016; PINO et al., 2012; HEEAGER e ROSE, 2015), Kanban (AHMAD et al., 2016) e XP (CHOUDHARI e SUMAN, 2012; CAPILUPPI et al., 2007), pois são os mais utilizados pelas indústrias segundo os artigos encontrados com a execução do MSL. Além desses métodos, também foi incluído o OKR (NIVEN e LAMORTE, 2016) na adaptação do Scrum com objetivo de dar uma visão estratégica e permitir um canal de comunicação entre os times de negócio e de desenvolvimento.

Os métodos ágeis surgiram como uma alternativa aos processos mais pesados de desenvolvimento de software, conhecidos como processos tradicionais, os quais são baseados no modelo em cascata (MELO et al., 2013). A finalidade dos métodos ágeis foi trazer uma dinâmica menos burocrática aos ambientes de desenvolvimento de software (CHAU; MAURER, 2004) com o objetivo principal de abraçar mudanças (FOWLER e HIGHSMITH, 2001). Entre os benefícios do uso de métodos ágeis, citados pelo *10th Annual State of Agile Report* (VERSION ONE, 2016), estão a aceleração na entrega do produto, aumento da qualidade do software, além da melhora significativa da gestão de mudanças. O *15th Annual State of Agile Report* (VERSION ONE, 2021) aponta que a adoção de métodos ágeis está em sua maioria - 64% - focada na busca de manejar mudanças de prioridades. Segundo Ahmad et al. (2016), a visibilidade do projeto e a satisfação do cliente também são áreas que podem ser melhoradas com o uso dos métodos ágeis.

O Manifesto Ágil (FOWLER e HIGHSMITH, 2001) é a declaração dos princípios ágeis e propõe uma série de quatro valores e doze princípios que deveriam ser seguidos a fim de consolidar um ambiente realmente ágil. A tabela 1 traz os quatro valores propostos pelo manifesto, acompanhados de uma breve justificativa.

Tabela 1 - Valores do Manifesto Ágil

	Valor	**Justificativa**
[1]	Interações entre indivíduos é mais importante que processos e ferramentas	Embora a importância de processos e ferramentas seja reconhecida, a interação entre indivíduos é ainda mais importante.
[2]	Software funcionando é mais importante que documentação abrangente	Documentação bem detalhada acerca do projeto não é necessariamente ruim, mas o foco principal do time deve se manter no produto final. A documentação mantida deve ser apenas aquela que é essencial.
[3]	Colaboração do cliente é mais importante que a negociação de contratos	Muitas vezes os contratos fazem parte de uma regulação do projeto, mas deve estar claro para todos que apenas a colaboração contínua pode ajudar os desenvolvedores compreender e entregar aquilo que o cliente quer.
[4]	Responder a mudanças é mais importante que seguir um plano	Seguir um plano é uma boa ideia, no entanto, poucos são projetos entregam exatamente aquilo que foi planejado. Por isso, é mais importante que o time seja capaz de responder a mudanças externas quantas vezes forem necessárias.

Fonte: Adaptado de Fowler e Highsmith (2001)

Além dos quatro valores supracitados, os Métodos Ágeis possuem também doze princípios que os norteiam. O Quadro 2 traz um mapeamento entre os doze princípios ágeis e os quatro valores citados anteriormente ([1], [2], [3] e [4]), adaptado de Singh, Singh e Sharma

(2013). Esses princípios estão presentes nas metodologias exploradas nesse estudo, a exemplo do primeiro que versa sobre entrega funcional, com valor e de forma contínua, que é um dos cernes do Kanban e XP possuindo como valores, todos os citados anteriormente.

Quadro 2: Cruzamento entre Princípios e Valores Ágeis

Princípios	**Valores**			
	[1]	**[2]**	**[3]**	**[4]**
Garantir a satisfação do cliente através da entrega contínua de software funcionando	X	X	X	X
Mudanças nos requisitos são bem-vindas, ainda que tardias, para garantir vantagens competitivas aos clientes			X	X
Entregas frequentes aos clientes		X		X
Colocar pessoas de negócio e desenvolvedores para trabalhar juntos diariamente durante todo o projeto	X		X	
Construir projetos em torno de indivíduos motivados, fornecendo um ambiente de confiança que eles farão o seu trabalho	X			
A maneira mais eficiente de trocar informações dentro de um time de desenvolvimento é através de comunicação face a face	X			
Software funcionando é a principal medida de progresso		X		X
Processos ágeis devem promover desenvolvimento sustentável, onde patrocinadores e desenvolvedores devem ser capazes de manter um ritmo constante indefinidamente	X		X	
Atenção contínua a excelência técnica e bom design melhoram a agilidade		X		
Simplicidade é essencial		X		
As principais arquiteturas, requisitos e projetos emergem de times auto-organizáveis		X		

Em intervalos regulares o time se reúne para refletir como se tornar mais efetivo e, apropriadamente, ajustar comportamento	X			X

Fonte: Adaptado de A. Singh, K. Singh e Sharma (2013)

Além de citados na MSL, a pesquisa de Digital.ai (2021) afirma que os métodos ágeis mais populares são, em ordem: Scrum (SCHW ABER e SUTHERLAND, 2020), Kanban (AHMAD; MARKKULA; OIVO, 2013), XP (BECK, 2000), Lean (POPPENDIECK e POPPENDIECK, 2003), entre outros como Crystal (COCKBURN, 2006) e FDD (PALMER, 2011).

Além disso, a popularidade dos métodos ágeis, desde seu surgimento em idos da década de 2000, tem crescido ainda mais ao longo dos últimos anos. Isto já era afirmado pela Version One (2016) no relatório 10th Annual State of Agile Report, pesquisa esta realizada anualmente com empresas de todo o mundo, a fim de diagnosticar o estado atual dos métodos ágeis no mercado. Sendo essa tendência confirmada por Digital.ai (2021), com 94% das companhias pesquisadas, afirmando utilizar ágil.

A evolução desta tendência de adoção fica clara pela quantidade de participantes que de maneira espontânea participaram das pesquisas ao longo dos anos. O relatório 2016, faz um comparativo, mostrando 3.880 responderam as questões, um aumento de quase 400% sobre as 1000 pessoas que em 2006 (primeiro ano de pesquisa). Saltando em 2021 para mais de 4.100 respostas. Essas pesquisas buscaram indicar os motivos para adoção dos métodos ágeis, a maturidade das empresas e os desafios encontrados nessa adoção. Os motivos citados para adoção foram:

- Capacidade de gerenciar as mudanças (DIGITAL.AI, 2021) para 64% dos pesquisados
- Aceleração da entrega do produto (DIGITAL.AI, 2021) para 64% dos participantes;
- Aumento da produtividade (47%);
- Melhora no alinhamento entre TI e negócio (47%);
- Aumento da qualidade do software (42%);
- Melhora na previsibilidade de entregas (41%).

É válido frisar que, de acordo com a pesquisa de 2021 (DIGITAL.AI, 2021), o principal fator para a adoção de métodos ágeis nas companhias foi o principal fator para a adoção de métodos ágeis nas companhias. Tal fatordispara 10 pontos percentuais à frente da pesquisa da Version One (2016), onde 54% consideraram essa capacidade importante, mas não a principal.

Essa tendência evidencia a busca dos times por responder as mudanças de negócio com mais naturalidade e praticidade.

3.3.1 Scrum

O Scrum é um framework para desenvolvimento de produtos complexos por pequenos times. Ele define claramente um conjunto de papéis para os atores dentro dos times, sendo estes: *product owner*, *scrum-master* e *dev-team*. Para execução de um projeto existe uma lista de tarefas chamada de *backlog*, que guia as atividades da equipe, de acordo com as prioridades. A partir dessa lista, as demandas são selecionadas em um *sub-backlog* para serem trabalhadas dentro de prazos de tempo pré-definidos, chamados *sprints*. Vale ressaltar que durante uma *sprint* não são esperadas mudanças no *backlog* ou no prazo de entrega acordado. A possibilidade de alterar os escopos é sempre tratada como uma exceção e executada através da cerimônia de Cancelamento de Sprint. Este cancelamento gera então uma nova abertura de Sprint, com todo o planejamento devendo ser novamente executado. Essa dinâmica conflita com o modelo de negócio que abraça mudança e deve reagir de forma mais imediata possível.

Esse cenário é nebuloso e não é bem descrito nem mesmo por Schwaber e Sutherland (2020) no guia do framework Scrum. Esse é um sinal de que o Scrum não está preparado para reagir a mudanças de escopo inerentes ao ambiente de SaaS, pois não considera plenamente e nem versa sobre como tais cancelamentos devem ser conduzidos.

Por definição, o “Scrum é um framework leve que ajuda pessoas, times e organizações a gerar valor por meio de soluções adaptativas para problemas complexos” (SCHWABER e SUTHERLAND, 2020). Ele surgiu na década de 90 por meio de desenvolvedores de software, com o intuito de capacitar times a responder a mudanças nos requisitos de forma cada vez mais rápida, além de focar no software em si (SOMMERVILLE, 2019), abandonando grandes massas de documentação e focando nas pessoas. São três elementos chaves que o compõem:

- Papéis: Scrum Master, *Product Owner* (P.O.) e *DevTeam*;
- Eventos: *Sprint, Daily Meeting, Sprint Planning, Spring Review e Sprint Retrospective*;
- Artefatos: *Backlog* do produto, *Backlog* da Sprint e Incremento.

A partir de uma *Sprint*, se desenvolvem os incrementos pelo *DevTeam* como uma unidade de planejamento onde o que foi feito é avaliado, recursos são selecionados para produção de um novo incremento e o requisito é codificado (SOMMERVILLE, 2019).

Os papéis são elementos fundamentais na aplicação do framework sobre um projeto de desenvolvimento de software, pois equilibram as forças entre clientes e o fornecedor do serviço, além de construir um sentimento de pertencimento aos desenvolvedores de software. Começando pelo Product Owner - PO), seu papel é representar a empresa e organizar as prioridades a serem desenvolvidas e incluídas no *backlog* do produto. O PO é sempre representado por uma pessoa que deve ser capaz de otimizar o trabalho e entrega do valor do time de desenvolvimento ao cliente (SCHWABER e SUTHERLAND, 2020).

O Scrum Master tem como principal responsabilidade defender e remover impedimentos do time de desenvolvimento, coordenando as cerimônias de retrospectivas e revisando as entregas em conjunto com o PO, planejando a aplicação do Scrum dentro da organização (SCHWABER e SUTHERLAND, 2020).

Por fim, o *DevTeam* é responsável exclusivamente por gerar código e entregas (incrementos) ao software alvo dos ciclos de desenvolvimento, sendo composto por membros que juntos promovem um time multifuncional.

O *framework* define também os eventos do Scrum, caracterizados por cinco cerimônias: *Sprint*, *Sprint Planning*, *Sprint Review, Sprint Retrospective e Daily Metting* (SCHWABER e SUTHERLAND, 2020).

Ao início da *Sprint* é executada a cerimônia de planejamento (*Planning*), em que o backlog é selecionado e gera o escopo a ser trabalhado. Diariamente o *DevTeam* reporta a todos o status das demandas e se há impedimentos *(Daily Scrum).* E ao final de uma *Sprint*, a equipe se reúne para discutir e avaliar o que foi entregue no evento chamado *Sprint Review*. E por fim, na *Sprint Retrospective*, para avaliação do processo, o time faz uma retrospectiva de tudo que ocorreu de positivo e negativo (SCHWABER e SUTHERLAND, 2020) e dá sugestões de como melhorar para o próximo ciclo. Além destes eventos, diariamente o time reporta o que foi feito e o que haverá de ser trabalhado, indicando se houver impedimentos. Esse evento é monitorado pelo *ScrumMaster*, cuja maior responsabilidade é remover os impedimentos.

3.3.2 Extreme Programming (XP)

O Extreme Programming é um conjunto de técnicas para desenvolvimento ágil, composta por um conjunto de boas práticas e ferramentas a serem seguidas por times que desejam entregar código de forma ágil com consistência. Surgiu no início do século XXI quando Beck (2000, p. 3) o definiu como um “método leve para times pequenos e médios que desenvolvem software e enfrentam requisitos que se alteram ou são vagos”. XP possui um

conjunto reduzido de práticas que devem ser seguidas pelo time para suportar entrega de valor e código de qualidade técnica refinada. Para isso, promete (BECK, 2000):

- Para desenvolvedores: desenvolver o que realmente importa, tomando decisões técnicas para entregar o melhor software e não tomando decisões que sua alçada não permite;
- Para gerentes e clientes: receber o maior valor possível sobre o trabalho executado em uma semana, e não mais que uma semana, permitindo que falhas sejam corrigidas rapidamente sem impactarem exorbitantemente os custos.

O XP é composto por 12 práticas (BECK, 2000): jogo do planejamento, pequenas entregas (*small releases* no texto original), conceitos compartilhados (metáforas), design simples, testes, refatoração, programação em par, posse coletiva de código, entrega contínua de funcionalidades, um máximo de 40h por semana de trabalho (Ritmo Sustentável), clientes incluídos no time e padrões no estilo de código. Sommerville (2019), afirma que todas as práticas sintetizam o desenvolvimento iterativo e o levam a níveis extremos. A seguir é apresentada a descrição detalhada de cada uma destas doze práticas:

- *Planning Poker* (Jogo do Planejamento): desenvolvedores se reúnem e avaliam as demandas, cruzando prioridades de negócio e gerando um plano de desenvolvimento. Caso o plano não seja exequível à realidade do time, este deve ser alterado (BECK, 2000);
- Pequenas Entregas: em curtos espaços de tempo, são liberadas versões simples que são colocadas em produção rapidamente (BECK, 2000);
- Metáfora: é um guia, escrito como uma estória, que explica como funciona o sistema (BECK, 2000);
- Design Simples: o sistema deve ser feito o mais simples possível para o momento do desenvolvimento e, caso no futuro seja verificado uma forma mais simples, a complexidade é removida (BECK, 2000);
- Testes: os desenvolvedores devem sempre desenvolver orientados a testes ou com testes unitários enquanto os clientes devem testar o software entregue (BECK, 2000);
- Refatoração: desenvolvedores devem dar manutenção no código removendo duplicidades, isolando responsabilidades, tornando mais flexível e melhorando a comunicação entre os módulos (BECK, 2000);

- Programação em par: os códigos a serem entregues devem ser desenvolvidos por dois desenvolvedores em uma única máquina, onde um inspecionará o código produzido pelo outro (BECK, 2000);
- Posse coletiva: qualquer um deve ser capaz de alterar o código a qualquer momento (BECK, 2000), sendo possível pela rotação dos desenvolvedores entre diferentes áreas do sistema, de modo que não se criem concentração de conhecimento no time (SOMMERVILLE, 2019);
- Entrega contínua: o sistema é compilado, construído e integrado sempre que uma tarefa é cumprida (BECK, 2000). Essa prática apoia a *small release*, vista anteriormente, e é garantida pelos testes unitários, que ao serem rodados devem funcionar (SOMMERVILLE, 2019). Esta prática é operacionalizada através das ferramentas de DevOPS;
- Ritmo Sustentável: as equipes devem trabalhar apenas nos dias de semana, preferencialmente, e sem horas extras (SOMMERVILLE, 2019). Portanto, horas extras não devem ser um padrão (BECK, 2000) e devem ser monitoradas;
- Cliente presente no time: o representante ou o próprio cliente deve estar no time por todo o tempo do desenvolvimento, tirando dúvidas e interagindo com o time (BECK, 2000);
- Padrão de código: a organização define os padrões de codificação e os desenvolvedores devem seguí-los, facilitando a documentação e comunicação da solução via código (BECK, 2000).

3.3.3 Kanban

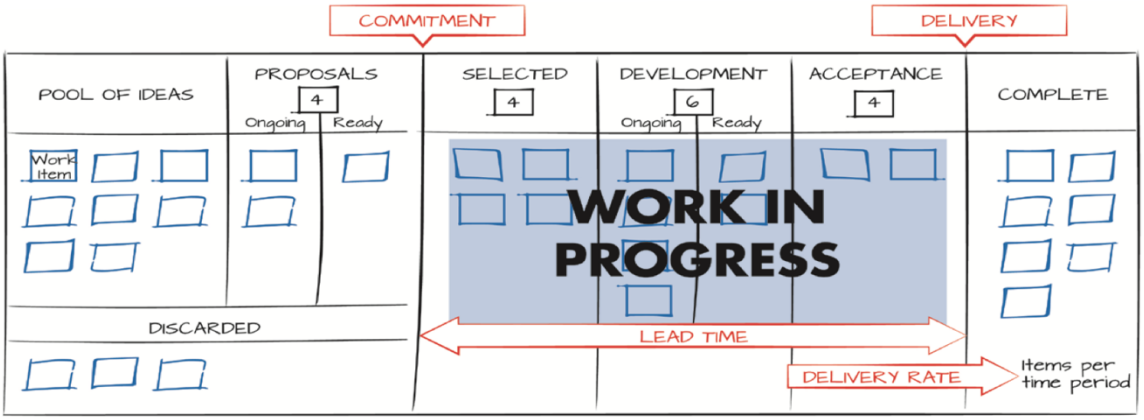
O Kanban surgiu a partir de uma pesquisa conduzida na Toyota, em meados da década de 70, e tem como objetivo conectar a linha de produção externa (clientes) e a linha de produção interna (recursos) (SUGIMORI et al., 1977). Vale salientar que o Work-in-progress - WIP pleno desejado em equipes de desenvolvimento de software é associado à produção just-in-time, onde possíveis flutuações de demanda e a capacidade de se adaptar às mudanças são desejadas e administradas (SUGIMORI et al., 1977).

Em desenvolvimento de software, o Kanban é uma técnica de acompanhamento das atividades de uma equipe através de um quadro composto por três etapas: “a fazer”, “em progresso” e “feito”. Os princípios em torno dessa abordagem são: visualizar o fluxo de trabalho, limitar a quantidade de trabalho em execução (*work in progress* – WIP), medir e

controlar o fluxo de trabalho, tornar explícito o processo de trabalho e usar modelos de referência para medir e gerenciar oportunidades (AHMAD, 2016). Na última década, o Kanban se tornou uma das principais técnicas dentre as metodologias ágeis (AHMAD *et al.*, 2013).

A principal ferramenta dessa técnica é o quadro Kanban, onde as colunas já citadas podem ser subdivididas conforme a necessidade de organização do time. Segundo Anderson (2016), o quadro Kanban mostra o fluxo de trabalho entre diferentes etapas do processo, da esquerda para direita, limitando sempre o WIP ao máximo que o time é capaz de desenvolver (sem atividades paralelas). Para exemplificar, a Figura 5 a seguir apresenta um exemplo de um quadro Kanban.

Figura 5 - Exemplo Quadro Kanban



Fonte: Anderson (2016)

3.4 DevOps

O uso cada vez mais extenso do SaaS, com o incremento de 35% entre 2018 e 2020 na oferta de soluções nesse modelo, chegando a uma previsão de receita de 116 bilhões de dólares em 2020 (GARTNER, 2019), tornou o processo de entrega de uma nova versão de software ainda mais importante. Essa importância se dá devido ao fato de que em SaaS os usuários não devem perceber a existência de novas versões - por vezes entregues em frequências diárias - presentes nas aplicações que sustentam seus serviços (MICCO, 2016). Para tornar isso possível, as empresas passaram a se valer de um conjunto de ferramentas e processos chamado DevOps para entregar de forma contínua, segura e escalável os serviços aos seus usuários.

Diversas empresas de *Plataform as a Service*, como a Amazon Web Services (AWS®), dispõem de soluções que auxiliam a implantação de novas versões de software de empresas

provedoras de SaaS entregando um fluxo básico de *deploy,* com etapas de compilação, testes e implantação. A Figura 6 demonstra esse fluxo entre o desenvolvimento e a entrega para o consumidor.

Figura 6 - Fluxo DevOPS Padrão

Fonte: O autor (2021)

A implicação prática do uso do modelo DevOps é a fusão entre os times de operações e desenvolvimento que antes da computação em nuvem e do modelo "*as a Service"* eram separados. Isso implica dizer que o time ao desenvolver o software está comprometido, por meio de testes, técnicas de mesclagem de código e rotinas de publicação a tornar possível um fluxo de entrega programado em script - chamado de *pipeline* -, possibilitando entrega de forma automatizada em produção. Dessa maneira, aquilo que antes era percebido como operações está fundido ao trabalho de desenvolvimento, incluindo todo o ciclo de vida da aplicação, do desenvolvimento aos testes e da implantação (AMAZON, 2020).

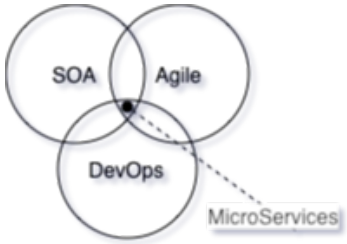
As principais técnicas que compõem o DevOps são as ligadas à entrega de solução, como a Integração Contínua (CI) e Entrega Contínua (CD). Além disso, para escalar a cultura DevOps, a arquitetura implementada deve estar em consonância a uma infraestrutura e código escalável onde existe idealmente:

- Infraestrutura é implementada e entregue como código;
- Software é entregue por meio de micro serviços;
- Registro das aplicações (*logs*) por meio de instrumentação (internamente ao código);
- Monitoramento de infraestrutura;

Para melhor entendimento do que são Microserviços (MS), é preciso conceituar a arquitetura *Service Oriented Architecture* (SOA). SOA é uma arquitetura muito utilizada pelas prestadoras de serviço de SaaS como forma de entregar valor em software distribuído aos seus clientes. Segundo Hao-He (2003), uma arquitetura orientada a serviço possui um conjunto de interfaces bem definidas que são públicas e disponíveis para consumidores e produtores; além disso, essas interfaces são contratos que indicam minimamente como o sistema se comportará em cada versão da aplicação. Dragoni (2018) aponta que os principais aspectos buscados ao se

adotar uma arquitetura MS é a escalabilidade. A figura 7 busca ilustrar a relação entre os MS, entrega de SaaS e metodologias ágeis, discutidas anteriormente. Os MS são a concretização da junção entre os princípios de uma arquitetura modular do SOA, com as ferramentas de entrega DevOPS e os princípios de entrega de valor ao cliente do Agile.

Figura 7 - MicroServiços e sua relação com o Agile, SOA e DevOps



Fonte: O autor (2021)

Como afirma Fox (2015), antes do SaaS, os lançamentos de software eram acontecimentos raros e importantes, e hoje as empresas que utilizam métodos ágeis lançam diversas versões por dia em casos extremos. Para que haja DevOps instituído em qualquer projeto ou empresa de desenvolvimento, se faz necessário um conjunto de ferramentas e processos implementados de tal maneira que o processo de entrega seja contínuo e ágil. Tais ferramentas e processos são divididos em dois grupos:

- Implantação e Integração Contínua (do inglês *Continous Deployment and Integration* - CD e CI): é um meio de disponibilização de nova versão do software de forma automatizada. CI é a composição de todas as etapas, sendo elas: compilação (*merge* e *build*), execução de testes unitários, instalação de binário e publicação de uma nova versão de software distribuído. O CD pode ser executado separadamente e diz respeito apenas as duas últimas etapas de instalação e publicação. Os requisitos para que a integração contínua seja bem-sucedida, segundo Fox (2015), são: haver testes unitários, de integração, de performance e de segurança, além de garantia de que não haja a quebra de contratos de API por parte do sistema que foram integrados anteriormente.
- Logging/Monitoring: monitorar a aplicação é a forma mais eficiente de indicar um mal funcionamento, indisponibilidade ou baixa performance de um sistema. Para isso, registra-se os eventos do sistema que podem ser processados e filtrados por meio de ferramentas como a pilha *Elasticsearch, Logstash e Kibana* (ELK). Além disso,

ferramentas de *Remote Performance Monitoring* (RPM), como o *NewRelic* têm obtido forte adesão da comunidade por fornecer instrumentação em tempo de execução para monitorar performance das aplicações, utilizando para isso bibliotecas incluídas diretamente no código antes da implantação.

3.5 OBJECTIVE AND KEY RESULTS (OKR)

Muito foi discutido a respeito de ferramentas operacionais para desenvolvimento de software como serviço, mas é necessário deixar claro o plano estratégico de evolução e desenvolvimento de um produto e isto pode ser feito através do uso de métricas. Para isso, metodologias orientadas a objetivos são muito úteis, permitindo quantificar os resultados esperados com base em métricas definidas. Dentro do contexto das metodologias ágeis, surgiu então o OKR como forma de definir e rastrear objetivos (*Objectives*), utilizando métricas (*Key Results*) que podem avaliar o alcance destes objetivos em nível estratégico. Dentro da adaptação proposta do Scrum, o OKR possui papel fundamental na conexão entre a camada tática (gerentes e líderes) e a estratégica (diretores e conselho) da companhia desenvolvedora de SaaS. Isto ajuda a mitigar as dificuldades já citadas quanto a conexão entre o time de negócio e times de desenvolvimento. A seguir, serão detalhados a história, funcionamento e benefícios do OKR.

Advinda de teorias clássicas como o Gerenciamento por Objetivos (MBO) (DRUCKER, 1954; WODTEK, 2016), foi adotada pelo Google em 1999 para ser uma ferramenta para gestão de projetos, criando indicadores e meios de rastreá-los. O *framework* OKR foi uma evolução do MBO de Drucker pelas mãos do CEO do Google, Andy Grove (NIVEN e LAMORTE, 2016). Andy tornou o processo de gestão por objetivos mais simples, respondendo a duas perguntas que deram origem ao OKR e incluindo, de forma leve, a alta gestão e o corpo operacional na definição dos objetivos. Atualmente, a técnica é aplicada por diversas empresas, no mundo todo, como Twitter, Linkedin, Intel e o próprio Google.

A execução do OKR exige engajamento entre o gestor e o executor, de tal modo que todos os objetivos sejam definidos em comum acordo. Essas definições permitem que em nível estratégico as metas e as formas como serão medidas sejam claras, instigantes e simples de serem entendidas pelos times. A transparência é um dos aspetos principais do OKR, sendo fundamental que tais objetivos sejam conhecidos por todos da empresa.

Para definir um OKR, segundo Doerr (2018), pioneiro no tema e ex-funcionário da Intel, é preciso atender a uma proposição básica que é definir os objetivos e como estes serão medidos. Dessa forma, uma descrição simples e bem definida de um objetivo em conjunto a resultados chave bem definidos e possíveis de serem quantificados, definem um OKR completo. A figura 8 apresenta um ciclo básico de OKR dentro de um time:

Figura 8 - Ciclo OKR

Fonte: Castro (2020)

A figura mostra que além de operacionalizar o OKR, é preciso engajar o time, alinhando e tornando público todos os objetivos, tornando assim possível a busca dos resultados para ao final atingir o objetivo definido. Para isso, o rastreio dos objetivos e seu monitoramento em torno dos resultados chaves é importante e fundamental para o sucesso do plano definido. Segundo Wodtek (2016), existem duas formas de medir o progresso, sendo uma delas totalmente orientada a índices de atingimento do OKR e outra quanto à confiança em relação ao atingimento do OKR. Na primeira, também chamada de *graded* OKR, o progresso é medido numa escala de 0 (não cumprido) a 1 (totalmente cumprido), sendo dessa forma público e claro o progresso para todos da empresa; na segunda, a comunicação sobre o progresso é o nível de confiança do time em atingir seu objetivo, sendo essa confiança atualizada constantemente para indicar uma mudança da percepção do time, mantendo todos sempre engajados e atentos aos OKR definidos.

Diante do exposto, o OKR se mostra uma ferramenta poderosa para a gestão indicar de forma clara as metas aos times, além de tornar ágil a comunicação e transparente as formas como todos os objetivos serão avaliados. Assim, a adaptação do Scrum proposta nessa pesquisa e voltada a SaaS é definida, definindo as metas utilizando-se o OKR, gerenciando as execuções

através de partes do Scrum em conjunto ao Kanban, implementando o código com as boas práticas XP e, por fim, entregando o software através das ferramentas de DevOps.

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo, foram trazidas como referencial para construção de uma metodologia voltada ao SaaS, todas as ferramentas e metodologias isoladas que hoje são utilizadas em ambientes de desenvolvimento ágil. Desta maneira, se constrói uma abordagem que busque mitigar os pontos fracos na utilização de métodos ágeis discutidos e que isolados, como o Scrum, trazem dificuldades ao processo de entrega de SaaS.

A adaptação proposta trata as diferentes esferas dos negócios, sendo OKR em nível estratégico, Scrum e Kanban em nível tático e, por fim, XP e DevOPS em nível operacional. Espera-se com esse referencial constituir uma base necessária à discussão e proposta de mudanças no estado da arte dos métodos ágeis com algo concreto e voltado ao SaaS, que é hoje o principal modelo de entrega de software.

4 MAPEAMENTO SISTEMÁTICO DA LITERATURA

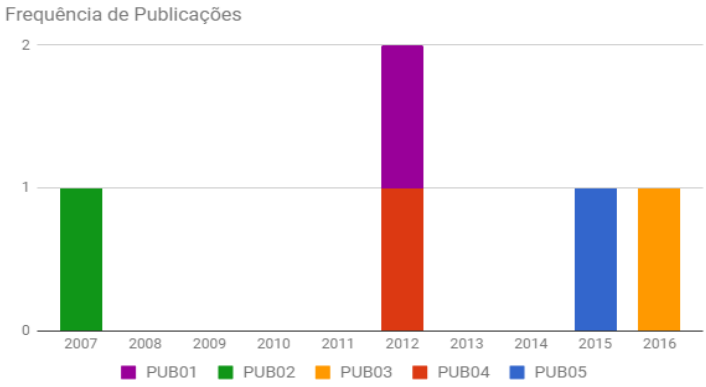
Neste capítulo, são apresentados os resultados obtidos após a análise de 5 estudos selecionados a partir da execução de um mapeamento sistemático da literatura. O protocolo detalhado está contido no Apêndice A. Inicialmente, é apresentado um sumário acerca da distribuição de frequência de publicações relacionadas ao tema metodologia ágil para software as a servisse - SaaS. Em seguida, são apresentadas as definições utilizadas pelos autores dos estudos selecionados para descrever Manutenção de Software e o uso das Metodologias Ágeis. Por fim, as demais subseções têm por objetivo contribuir para construção das respostas a cada uma das questões de pesquisa deste trabalho.

Vale ressaltar que o autor flexibilizou a questão chave do mapeamento para manutenção de software por não existir quantidade de resultados suficientes incluindo às questões de pesquisa do termo SaaS. Essa constatação deu ainda mais confiança na relevância de se pesquisar tal temática.

4.1 VISÃO GERAL DOS ESTUDOS

A Figura 9 ilustra um gráfico com a distribuição das publicações ao longo dos anos. Todos os estudos foram publicados em um intervalo de 10 anos, sendo o primeiro em 2007 e último em 2016. Apesar da lacuna de 10 anos, observa-se que 4 dos 5 estudos incluídos neste mapeamento foram publicados nos últimos 5 anos, entre a publicação do estudo de Capiluppi et al. (2007) e dos estudos de Choudhari e Suman (2012) e Pino et al. (2012). Esse dado pode ser um indicador do crescente interesse científico acerca do uso de métodos ágeis no processo de manutenção de software.

Gráfico 2 - Distribuição das publicações ao longo dos anos



Fonte: O autor (2021)

4.1.1 Manutenção de Software

A forma de definir manutenção de software foi diversificada entre os estudos selecionados: enquanto alguns autores adotaram definições formais, outros optaram por fazer classificações próprias do termo. De início, deve-se compreender que a manutenção de software é uma atividade muito importante e que deve ser tratada de maneira dissociada do desenvolvimento de software devido à natureza e características diferenciadas de cada atividade (PINO et al., 2012). Muitas das técnicas, ferramentas, processos e modelos utilizados no desenvolvimento de software não são diretamente aplicáveis à manutenção.

Segundo Choudhari e Suman (2012), por simplicidade, manutenção pode ser considerada um sinônimo de evolução, por ambas as atividades serem desenvolvidas após a primeira *release* do software. Os autores também citam que esta é a fase do desenvolvimento que demanda a maior parte do esforço dos desenvolvedores.

Capiluppi et al. (2007) se limitaram a classificar a manutenção como uma tarefa muito desafiadora, embora os desenvolvedores não estejam muito preocupados com essa fase. Os motivos para esse desinteresse é que, segundo os autores, a manutenção é tratada como uma tarefa rigorosa e que não representa ganho de experiência relevante para os desenvolvedores. Apesar disso, Pino et al. (2012) cita que desenvolvedores dedicam até 61% da vida profissional à manutenção de software.

No estudo realizado por Ahmad et al. (2016) foi utilizada a definição formal dada pelo IEEE, citando que manutenção de software pode ser o processo, componente ou sistema, que após a sua entrega corrige erros, melhorara a performance ou outros atributos ou adapta um software à um ambiente em mudança. Em Heeager e Rose (2015), manutenção de software é definida como toda e qualquer intervenção aplicada a um sistema após a sua entrega.

Os estudos primários também citaram diversos tipos de classificação das atividades de manutenção. De acordo com Pino et al. (2012) e Heeager e Rose (2015), por exemplo, são citados os seguintes tipos:

- Corretiva: envolve a solução de erros encontrados após a entrega.
- Adaptativa: responde a mudanças nos requisitos do software.
- Preventiva: utilizada para mitigar problemas futuros e corrigir limitações conhecidas.
- Perfectiva: trata da inclusão de novos requisitos de usuários.

Os conceitos apresentados anteriormente são compartilhados entre os estudos, variando somente o nível de detalhe dado a cada tipo de manutenção. Em Pino et al. (2012), por exemplo, as correções são ainda classificadas como urgentes ou não-urgentes.

4.1.2 Métodos Ágeis

Por se tratar de um tema recorrente, a maioria dos estudos resultantes das fases preliminares deste trabalho consideraram o senso comum quando definiram métodos ágeis. Apesar disso, aqueles que buscaram definições na bibliografia, o fizeram recorrendo ao Manifesto Ágil (FOWLER e HIGHSMITH, 2001) de forma direta.

O senso comum foi de que os métodos ágeis definem processos iterativos, com documentação "ausente", comunicação direta entre desenvolvedores e clientes, foco em código de qualidade e no negócio (AHMAD et al., 2016). Ainda foram definidas como características essenciais das metodologias ágeis: pequenos times, testes frequentes e cultura de ritmo de trabalho baseado em pontos de estória definidas como esforço a cada *sprint* (HEEAGER e ROSE, 2015).

Os autores como Choudhari e Suman (2012), destacam que os proponentes dos métodos ágeis defendem que tais processos devem ser capazes de responder às mudanças e suportar melhor as pressões envolvidas na evolução e manutenção do software. Além disso, é citado que a criação de um ambiente sustentável para o desenvolvimento e evolução do software é um objetivo explícito dos times ágeis, como também a prática de refatoração, esta última encarada como uma atividade necessária e positiva.

Ahmad et al. (2016) demonstraram a dificuldade junto a entrevistados que não possuíam conhecimento prévio dos conceitos ágeis. Essa constatação destacou tal estudo como aquele que melhor contextualizou o cenário de carência de trabalhos concluídos ou pesquisas acerca do tema manutenção de software e o uso de metodologias ágeis.

4.2 MÉTODO DE PESQUISA

Com o intuito de elucidar a forma como o tema deste mapeamento tem sido abordado pelos pesquisadores nos estudos primários encontrados, buscou-se responder a seguinte pergunta de pesquisa nesta seção:

RQ1.1*: Quais os métodos de pesquisa utilizados para estudar a manutenção de software em um contexto ágil?*

Durante a definição do protocolo de pesquisa, notou-se a ausência de trabalhos voltados à Saas. Dessa forma, “manutenção de software e o uso de metodologias ágeis” foi o tema mais

próximo a ser pesquisado devido a sua similaridade ao conceito de serviço e constante evolução e manutenção do software oferecido como serviço.

De maneira sintética, há dois tipos de metodologia utilizada nos 5 estudos incluídos neste mapeamento: estudo de caso e pesquisa-ação. No total, 3 estudos utilizaram estudos de caso (CHOUDHARI e SUMAN, 2012; CAPILUPPI et al., 2007; AHMAD et al., 2016) e 2 utilizaram pesquisa-ação (PINO et al., 2012; HEEAGER e ROSE, 2015). A seguir é apresentada descrição mais detalhada acerca do método utilizado por cada estudo.

O estudo de caso conduzido pelos autores Choudhari e Suman (2012) se valeu da aplicação de um questionário que obteve 200 respostas. Dentre os respondentes, 50 eram provenientes da indústria de software, não sendo especificado pelo autor o número de indústrias nem o segmento das mesmas. Os demais 150 participantes eram alunos de pós-graduação envolvidos em projetos de manutenção de software. O objetivo dos autores era avaliar uma técnica ágil proposta para auxiliar os times de manutenção a estimar o esforço das suas tarefas.

Capiluppi et al. (2007) foram mais detalhistas quanto ao método utilizado para o seu estudo de caso. A pesquisa foi realizada em uma pequena empresa de TI, a fim de descrever a evolução de um software comercial de recomendação de publicidade de maneira inteligente na web. O time estudado era composto por 12 integrantes, sendo 10 desenvolvedores, 1 *designer* e 1 responsável pela infraestrutura. Ao todo, o estudo durou 2 anos e meio e a coleta de dados foi feita utilizando métricas extraídas do próprio código fonte da aplicação. Este estudo se destacou por ser o único, dentre os artigos incluídos neste mapeamento, a usar uma abordagem qualitativa de maneira complementar a quantitativa, como atas de reuniões, observações e comentários dos membros do time para compor seus resultados.

Capiluppi et al. (2007) realizaram dois estudos de caso com o objetivo de analisar a migração de times de manutenção que utilizavam Scrum para o Kanban. A escolha por múltiplos estudos casos, segundo os próprios autores, teve como objetivo prover múltiplas fontes de evidências que serviriam para fortalecer as suas descobertas. A amostra foi composta por 2 times de 2 grandes empresas de TI, sendo os times compostos por 6-8 integrantes com cada time possuindo um gerente de projetos responsável. A coleta de dados se deu por meio de entrevistas semiestruturadas. Ao longo dos 2 anos de estudo foram realizadas 17 entrevistas, sendo 9 pessoalmente e 8, via *Skype*. Os entrevistados desempenhavam variados papéis: 2 *product owner*s, 3 *coaches* (3), 3 *scrum masters*, 3 gerentes de projeto (3) e 6 desenvolvedores.

A pesquisa-ação feita por Pino et al. (2012) foi realizada em 2 pequenas organizações uruguaias do ramo de TI. O objetivo era avaliar uma metodologia ágil voltada para times de manutenção em pequenas empresas. Ao todo, 71 profissionais de variados cargos dentro das

organizações participaram do estudo. A coleta de dados foi feita através do registro da observação dos autores. Além disso, os participantes foram estimulados a fornecer um *feedback* acerca da metodologia proposta, servindo também de entrada de dados para o estudo.

Por fim, Heeager e Rose (2015) também lançaram mão de uma pesquisa-ação. Ao contrário de Pino et al. (2012), que executaram a intervenção em 2 pequenas empresas, os autores focaram em apenas uma organização de grande porte, representante da indústria naval. Apesar disso, o número de participantes foi menor: 9 profissionais, incluindo os gerentes da empresa. O objetivo do estudo era aferir até que ponto o uso de uma metodologia ágil pode ser adequado para o processo de manutenção de software e para fornecer diretrizes para adoção na forma de técnicas aplicáveis ao ambiente da pesquisa. O estudo foi executado ao longo de 1 ano e a coleta de dados foi realizada por meio de relatos das observações dos autores e de entrevistas com os representantes da organização que participaram do estudo.

4.3 METODOLOGIAS ÁGEIS UTILIZADAS

Os estudos primários utilizados neste mapeamento foram realizados em diferentes contextos, utilizando metodologias ágeis diferentes. Nesta seção busca-se responder a seguinte questão de pesquisa:

RQ1.2: *Que metodologias ágeis têm sido usadas no processo de manutenção de software?*

Métodos ágeis são compostos por um conjunto de técnicas e estratégias que buscam empoderar os times de desenvolvimento, compartilhando a propriedade do código entre os desenvolvedores (PINO et al., 2012). A busca por entregas em intervalos de tempo cada vez mais curtos e em menores incrementos de código, tem como objetivo principal entregar valor com agilidade aos clientes finais, aproximando times e clientes, promovendo uma comunicação "olho-no-olho" (AHMAD et al., 2016; PINO et al., 2012; HEEAGER e ROSE, 2015). O Scrum (AHMAD et al., 2016; PINO et al., 2012; HEEAGER e ROSE, 2015), o Kanban (AHMAD et al., 2016) e o XP (CHOUDHARI e SUMAN, 2012; CAPILUPPI et al., 2007) foram as metodologias ágeis mais utilizadas nas indústrias que constituíram a pesquisa dos artigos selecionados.

Dois dos estudos utilizaram os princípios de *eXtreme Programming* (XP), buscando alta produtividade e qualidade de código através do desenvolvimento de código em pares, cultura de testes, design eficiente e entrega contínua de código. Ao todo, XP é composta por 12 práticas (BECK, 2000) que sintetizam a metodologia: jogo do planejamento, pequenas versões,

conceitos compartilhados (metáforas), design simples, testes unitários, refatoração, programação em pares, posse coletiva de código, entrega contínua de funcionalidades, um máximo de 40h por semana de trabalho, clientes incluídos no time e padrões no estilo de código.

As informações extraídas dos estudos selecionados também nos permitem traçar um paralelo entre as diferentes metodologias. Por exemplo, o Kanban foi indicado como uma metodologia mais adequada frente às outras (AHMAD et al., 2016), uma vez que atividades de manutenção, sobretudo, manutenção corretiva, precisam ser resolvidas com rapidez e serem disponibilizadas no formato de entrega contínua e não com *sprints* com intervalos de tempo bem definidos, variando de 2 até 4 semanas.

Indo de encontro a essa constatação sobre metodologias distintas como o Kanban, os resultados do estudo de Pino et al. (2012) demonstram que uma abordagem utilizando o Scrum está alinhada com a forma com que times ágeis enxergam o processo de manutenção. E ainda mais: para que propostas de gerenciamento ágil do processo de manutenção, como o Agile_MANTEMA, prosperem, é essencial que as atividades das *sprints* sejam monitoradas de perto pela gerência e que haja integração com outros processos auxiliares, gerando-se um *overhead* no gerenciamento de tarefas.

4.4 MELHORIAS SOBRE O PROCESSO DE MANUTENÇÃO DECORRENTE DO USO DE MÉTODOS ÁGEIS

Um dos principais focos durante a leitura dos artigos era a descoberta de evidências que sugerissem melhorias no processo de manutenção causada pelo uso de métodos ágeis. Esta seção traz informações reunidas durante o mapeamento que ajudam a responder a seguinte questão de pesquisa:

***RQ1.3**: Que benefícios os métodos ágeis trazem ao processo de manutenção de software?*

A visibilidade do projeto é uma das áreas mais beneficiadas pelo uso de métodos ágeis no processo de manutenção, tanto para o próprio time como para a organização como um todo. De acordo com os resultados apresentados por Ahmad et al. (2016), o uso do Kanban fez com que os times de manutenção tivessem uma visão geral de todas as tarefas, identificando gargalos causados por muitas atividades com *status* “em progresso”. Para resolver esse problema, os times adotaram políticas que limitavam a quantidade de tarefas em progresso, impossibilitando que novas tarefas fossem iniciadas até que o trabalho em progresso tivesse sido concluído.

As melhorias na visibilidade do projeto se estendem até a gerência do projeto. Os gerentes são capazes de identificar atividades bloqueadas ou que estão em progresso há muito tempo e alocar especialistas para trabalhar de forma colaborativa com outros integrantes do time na resolução da tarefa (PINO et al., 2012). Além disso, o aumento da visibilidade favorece o agrupamento de requisições associadas a um mesmo conjunto de mudanças, impactando diretamente na eficiência e produtividade, permitindo que os testes de regressão sejam executados sobre o grupo de tarefas e não a cada uma isoladamente.

Uma abordagem ágil também auxilia em atividades como a estimativa do esforço de tarefas de manutenção (CHOUDHARI e SUMAN, 2012). Ao utilizar técnicas de metodologias ágeis para este fim, os times de manutenção passaram a ser 91% mais assertivos com relação à utilização de meios convencionais para estimar esforço.

Outro aspecto relevante foi reportado devido ao uso da refatoração constante do código-fonte. Isso auxilia no controle da entropia do código produzido, gerando uma ausência quase que completa de itens de alta complexidade (CAPILUPPI et al., 2007).

A gestão de mudanças e demais requisições dos clientes também evoluem com o uso das metodologias ágeis. De acordo com Ahmad et al. (2016), a partir do momento que o time passa a acompanhar as tarefas diariamente, a substituição de tarefas ou mesmo a troca de prioridades passa a ser mais bem gerenciada. Isso permite que o time possa, de fato, focar nas principais necessidades do cliente, oferecendo valor o mais rápido possível. O reflexo disso pode ser percebido pela melhoria na satisfação do cliente nos projetos de manutenção (CAPILUPPI et al., 2007).

4.5 DESAFIOS NO PROCESSO DE MANUTENÇÃO DECORRENTE DO USO DE MÉTODOS ÁGEIS

Outro objetivo deste mapeamento era identificar possíveis problemas ocasionados pelo uso de metodologias ágeis de desenvolvimento de software no processo de manutenção. Esta seção tenta responder a seguinte questão de pesquisa:

RQ1.4*: Quais os desafios conhecidos pelo uso de métodos ágeis na manutenção de software?*

Dos 5 estudos selecionados, apenas 2 citaram alguns desafios, indo de encontro aos benefícios citados na seção anterior. O estudo de Ahmad et al. (2016) aponta que Scrum não cabe no agitado processo de manutenção.

Entre os problemas indicados está, por exemplo, a chegada de tarefas que necessitam de atenção imediata no meio de uma *sprint* corrente. Com relação a este problema, Heeager e Rose (2015) afirmam que frequentemente *sprints* de manutenção são sujeitas à interrupção devido a demandas urgentes dos clientes.

A inclusão de tarefas adicionais àquelas que o time havia se comprometido no início da *sprint* pode levar todo o ciclo à falência, impedindo que o time entregue aquilo que havia combinado com o cliente. Isso aumenta a pressão sobre o time, gera trabalho extra, desmotivação, além de causar insatisfação do cliente.

Outro desafio apontado se refere à falta de comunicação entre times diferentes que trabalham no mesmo projeto. Apesar das reuniões diárias fomentarem a comunicação constante e proverem informações atualizadas acerca do andamento das tarefas, torna-se difícil reunir todos os integrantes de times distintos na mesma reunião. Além disso, Heeager e Rose (2015) pontuam que como as tarefas de manutenção são, geralmente, compostas por requisições pequenas e específicas, nem sempre é produtivo reunir todos os integrantes para discutir e compartilhar conhecimento dada a natureza diferente das suas tarefas.

Além disso, segundo tais autores, a redução considerável de documentação proposta pelos métodos ágeis pode ser danosa para a manutenção do sistema. Isso se dá porque muitas vezes é necessário documentar o sistema para auxiliar em decisões de projeto e manutenções no futuro. Uma vez iniciado o processo de manutenção, também se faz necessário que o time tenha posse de documentos que auxiliem na compreensão do sistema e no impacto gerado pelas mudanças requisitadas.

4.6 DISCUSSÃO

O objetivo deste mapeamento era auxiliar na construção do conhecimento acerca do que se sabe do uso de metodologias ágeis no processo de manutenção de software. Nesta seção são discutidos os resultados obtidos no mapeamento, destacando as implicações para a pesquisa na área, bem como são discutidas as limitações do presente trabalho.

4.6.1 Metodologia de Pesquisa e Métodos Ágeis Utilizados

- RQ1.1:

De acordo com a resposta à pergunta RQ1.1, foi observado que os estudos analisados possuem predominantemente tempos de pesquisa longos, longitudinais em alguns casos com

anos de dados analisados, o que ajuda o entendimento de escolhas de estudos de caso ou pesquisa-ação como predominantes. A duração das pesquisas (cerca de 1 a 2 anos e meio – embora os estudos de (CHOUDHARI e SUMAN, 2012) e (PINO et al., 2012) não explicitaram a duração da pesquisa) é um indicativo que os autores buscam uma melhor compreensão da motivação e dos efeitos de se usar um método ágil para governar o processo de manutenção de software.

Estudos como o de Ahmad et al. (2016) evidenciam ainda a busca por soluções adequadas para o contexto estudado e pesquisas de caráter etnográfico, ao incluir o pesquisador como agente da mudança observada, muito embora, como já destacado, os autores não tenham tornado explícito o tempo de duração do trabalho.

É possível também observar o pequeno número de indústrias participantes, embora os estudos tenham demonstrado variância entre o porte das organizações e o tamanho dos times estudados. O baixo número de empresas participantes pode indicar duas coisas: (i) poucas empresas estão utilizando métodos ágeis no processo de manutenção de seus softwares; e/ou ainda (ii) a dificuldade de gerenciar estudos longos e de caráter cultural (envolvendo pessoas como principal agente ambiental) sobre metodologias de desenvolvimento em organizações de diferentes portes.

Pode-se constatar o caráter construtivista dos estudos primários utilizados por este mapeamento sistemático. Destacam-se os métodos de inferência de resultados baseado em questionários (CHOUDHARI e SUMAN, 2012), entrevistas semiestruturadas (AHMAD et al., 2016; HEEAGER e ROSE, 2015) e observações (AHMAD et al., 2016; PINO et al., 2012). Entre esses métodos, destacam-se técnicas que permitem a coleta de dados de teor qualitativo, visando a construção de conhecimento (apesar dos riscos de validade externa, sobretudo) que ajude a compreender de forma geral os resultados frente à intervenção aplicada.

Apenas o estudo de Capiluppi et al. (2007) utilizou métricas extraídas do próprio software, uma abordagem totalmente quantitativa, para chegar a resultados que permitissem tirar conclusões acerca dos efeitos do uso de metodologias ágeis sobre a manutenção do software estudado.

- *RQ1.2:*

Apesar dos inúmeros métodos ágeis propostos, apenas três se fizeram presentes nos estudos incluídos no mapeamento: XP (CHOUDHARI e SUMAN, 2012; CAPILUPPI et al., 2007), Scrum (AHMAD et al., 2016; PINO et al., 2012; HEEAGER e ROSE, 2015) e Kanban (AHMAD et al., 2016).

Embora não seja novidade a presença de XP e Scrum entre as metodologias utilizadas, haja vista que estão entre os métodos ágeis mais utilizados no mundo (VERSION ONE, 2016), a escolha das organizações por estas metodologias nos levam a algumas inferências. Entre elas estão a escolha por métodos mais maduros no mercado a fim de evitar problemas como a falta de conhecimento dos colaboradores da organização. Outra possível justificativa é o reaproveitamento de técnicas e estratégias utilizadas por times de desenvolvimento também na execução das tarefas de manutenção.

Apesar disso, é possível também perceber que a falta de sucesso utilizando uma determinada abordagem pode levar as organizações a pivotar sua estratégia e migrar para um método que pode ser mais adequado ao seu contexto. Este caso é reportado no estudo de Ahmad et al. (2016), que traz relatos da migração de times de manutenção do Scrum para o Kanban. Além de pôr em evidência possíveis problemas na utilização do Scrum em atividades de manutenção, o estudo sugere uma preocupação das organizações com uma maior satisfação do cliente utilizando conceitos do Kanban, como a entrega contínua e uma maior abertura a chegada de mudanças mesmo que “urgentes”.

4.7 METODOLOGIA DE PESQUISA E MÉTODOS ÁGEIS UTILIZADOS

- *RQ1.1* e *RQ1.2:*

Essa investigação demonstrou discordância entre os autores nas distintas conclusões a que chegaram. Alguns indicando que a utilização de metodologias ágeis é adequada ao processo de manutenção (PINO et al., 2012) e outros apontando que existem fatores que impedem o uso pleno de métodos ágeis (HEEAGER e ROSE, 2015). Além disso, estudos como o realizado por Ahmad et al. (2016) se contradizem ao citar que, dependendo da metodologia escolhida, metodologias ágeis podem tanto solucionar quanto ser o motivo de problemas relacionados a tópicos como a visibilidade do projeto e a comunicação entre a equipe.

De maneira geral, os métodos ágeis possuem mecanismos que tendem a melhorar o processo de manutenção com entregas contínuas, boa aceitação de mudanças de última hora, compartilhamento de conhecimento acerca do domínio da aplicação, entre outros. A utilização de quadros, tornando o espaço de trabalho mais colaborativo e visual tende a oferecer benefícios a todos, inclusive à gerência que pode tomar atitudes como alocação de recursos de acordo com o andamento do cronograma.

Para usufruir dos benefícios do mundo ágil é necessário ter consciência de que manutenção e desenvolvimento não são a mesma coisa e, devido a isso, devem ser tratadas de maneira diferente. Como citado por Capiluppi et al. (2007), muitas técnicas, ferramentas e estratégias ágeis bem-sucedidas no desenvolvimento de software não serão, necessariamente, adequadas para atividades de manutenção. Apesar disso, técnicas ágeis voltadas para atividades de desenvolvimento podem ser adaptadas para o contexto de manutenção, como reportado no estudo realizado por Choudhari e Suman (2012).

É válido observar também que alguns preceitos ágeis vão de encontro às necessidades do processo de manutenção, como o uso de documentação, por exemplo. Como metodologias ágeis focam em comunicação informal durante o ciclo de desenvolvimento, é possível que importantes decisões arquiteturais não sejam documentadas e gerem enormes problemas para o time de manutenção no futuro. Isso leva a problemas como a dependência de pessoas que estiveram presentes no desenvolvimento do sistema, que podem já ter deixado a organização. Outro problema pode ser a alteração de partes sensíveis do sistema que causam efeitos colaterais no sistema como um todo.

O uso de *sprints* também parece não ser adequado para o processo de manutenção, sobretudo a manutenção corretiva. Ao se manifestar um erro em ambientes de produção é esperado que a correção seja implantada o quanto antes. Nestes cenários uma abordagem de entrega contínua, como preconizado pelo Kanban, parece ser mais adequado do que as *sprints* com escopo fechado de metodologias como o Scrum.

Um bom guia para a implantação bem-sucedida de um método ágil para controlar efetivamente o processo de manutenção são as heurísticas criadas por Heeager e Rose (2015). Nelas é possível encontrar uma série de adaptações e balanceamentos que podem ser aplicados aos valores e princípios ágeis para que estes caibam melhor no processo de manutenção. Métodos ágeis voltados inteiramente para a gestão do processo de manutenção, como o Agile_MANTEMA também possui resultados positivos e pode ser utilizado para integrar estes dois mundos.

4.8 LIMITAÇÕES

Assim como qualquer estudo científico, este mapeamento está sujeito a certas limitações. A primeira delas é a cobertura de estudos que tratam sobre o tema. Eventualmente algum estudo relevante para o contexto da pesquisa pode não ter sido alcançado ou mesmo ter

sido excluído pelos autores devidos aos processos utilizados. Para mitigar o problema de cobertura, lançamos mão de uma estratégia de busca combinada, complementando a busca automática em bibliotecas digitais com uma busca manual em eventos e periódicos relevantes que abordam os temas desse mapeamento. Esta estratégia teve por objetivo a seleção de estudos que eventualmente não utilizassem uma das palavras-chave da *string* de busca utilizada na busca automática ou ainda estudos que não estivessem indexados por nenhuma das bibliotecas digitais.

O número pequeno de estudos encontrados também é uma limitação do presente trabalho. Apenas 5 artigos puderam ser incluídos de acordo com os critérios de inclusão e exclusão utilizados. Potenciais estudos relevantes foram deixados de fora pelos autores não terem acesso aos mesmos, o que certamente empobrece os resultados reportados pelo mapeamento. O baixo número de estudos analisados aliado a amostras limitadas utilizadas pelos autores dos mesmos, também impossibilitam que os resultados do mapeamento possam ser generalizados. A tentativa de ampliar a cobertura do mapeamento através de *snowballing* se mostrou ineficaz pois os trabalhos continham poucas referências externas, além de distintas do tema da pesquisa.

5 ADAPTAÇÃO DO SCRUM AO SAAS – "SAAS PROCESS"

A empresa alvo desta pesquisa-ação está presente no mercado desde os anos 2000 e possui como principal modelo de negócios a entrega de software como serviço, sendo pioneira nessa área quanto se trata do mercado paraibano. A empresa surgiu como um setor interno de uma outra empresa focada na captura de transações, especificamente recebimento de contas (tecnologia para sistemas bancários), e teve como seu principal projeto a criação de um sistema para autorização, conciliação financeira e gestão online (*real-time*) dos pontos de atendimento.

Ao longo dos seus primeiros cinco anos, foi construída uma plataforma sólida de captura e processamento de transações que lhe permitiu sua emancipação em meados de 2006, tornando-se uma provedora de soluções para o mercado corporativo. Dessa maneira, o modelo de oferecimento de serviços impulsionou a adaptação do Scrum para que entregasse valor sem rótulos de tempo (*Sprints*), atendendo a disponibilidade da solução sem pausas, ou seja, vinte quatro horas por dia, sete dias por semana, além de abraçar com naturalidade novos requisitos de mercado, sem burocracia ou impedimentos prescritivos/processuais. Esses desafios juntamente à pressão por evolução de requisitos, por implementação e pela entrega da solução de forma ágil e flexível devem ser garantidos pelo processo de desenvolvimento de software utilizado pela empresa. Por esse motivo tamanha é a importância do processo de desenvolvimento ser adaptado ao negócio da empresa.

Conforme relato dos sócios, desde a criação da empresa, o processo de desenvolvimento de software foi definido com times pequenos, executando projetos de pequeno porte, tornando exequível uma abordagem *ad-hoc* de desenvolvimento de software, sem prejuízos quanto à ausência de um processo formal de software. Em meados dos anos 2010, a empresa passou a contar com um time de cerca de uma dezena de desenvolvedores e obteve clientes de médio porte, que demandavam requisitos de serviço mais sofisticados e maior profissionalismo na entrega do software. Nesta ocasião, foram relatadas que as principais dificuldades eram o controle de qualidade (*quality-assurance)* (ISO, 2020), a entrega contínua e a gestão de prazos e do *backlog* de desenvolvimento das demandas. Dessa forma, houve a necessidade latente de evoluir e estruturar o processo de desenvolvimento de software.

Como forma de contornar as limitações presentes no modelo *ad-hoc*, em 2012, foi contratada uma consultoria de processo de software com intuito de implantar uma metodologia que trouxesse:

- Gestão de prazos;
- Acompanhamento das demandas;

- Monitoramento do processo de desenvolvimento;
- Divisão de responsabilidades entre os componentes do time.

À época, havia um cenário semelhante ao descrito no ScrumGuide (SCHWABER e SUTHERLAND, 2020) que descreve o framework Scrum como indicado a contextos em que existe a necessidade de:

- "Gerar valor por meio de soluções adaptativas para problemas complexos", ou seja, desenvolvimento de um produto complexo;
- Auxiliar os times a serem capazes de se adaptar em situações em que "o processo que está sendo aplicado ou os materiais que estão sendo produzidos devem ser ajustados";
- Ter um time "pequeno o suficiente para permanecer ágil e grande o suficiente para concluir um trabalho significativo dentro de uma Sprint, normalmente 10 ou menos pessoas.".

A implantação do Scrum trouxe aos times uma melhor interlocução com as áreas de negócio - através do P.O. - remoção de impedimentos e representatividade do time através do Scrum Master, dando a todos uma confiança nos requisitos trazidos e no andamento das atividades, sem bloqueios não resolvidos.

Além disso, a gestão de prazos tornou-se mais assertiva com o planejamento das entregas ao final de cada *Sprint*, com controle de qualidade mínima na revisão da entrega pelo DEVTeam e P.O., além suporte aos times sobre aspecto de melhorias no processo utilizando-se da retrospectiva de Sprint. Estes foram os pontos positivos em torno da implementação do Scrum que trouxeram um maior profissionalismo à empresa por meio de sua Melhoria de Processo de Software, gerando a versão 1.0 do processo de desenvolvimento, composta puramente pelo Scrum.

Ao longo de aproximadamente quatro anos, a equipe de desenvolvimento mobilizou-se para entregar valor por meio de *Sprints* de quinze dias. O papel do coordenador de desenvolvimento foi transformado em Scrum Master, o gerente comercial passou a se tornar um *product owner* e o time de 6 desenvolvedores passou a revisar o produto ao final de cada ciclo, além de pontuar o que poderia ser melhorado, o que havia de bom e sugestões de melhoria. No entanto, esta implantação do Scrum puro trouxe alguns problemas devido à sua aderência parcial ao modelo de negócio de software como serviço, visto que no Scrum não está prevista a entrega contínua. Essa inadequação gerou conflitos entre o P.O e o *DevTeam,* pois,

segundo as regras do Scrum, a liberação deve ocorrer apenas ao final da Sprint. Assim, se tornou inviável compatibilizar a necessidade de liberação de versões em meio a *Sprint* com a descrição do *framework* que indica que é durante o evento de Sprint Review que "o Scrum Team apresenta os resultados de seu trabalho para os principais stakeholders" (SUTHERLAND e SCHWABER, p.11, 2020), deixando explícito não caber entregas antes desta cerimônia.

As pressões comerciais por entregas contínuas aliadas à necessidade de se obter uma melhor qualidade na codificação do software, levaram ao desenvolvimento por intermédio dos coordenadores de desenvolvimento da Melhoria de Processo de Software - MPS 2.0. Essa transição ocorreu no ano de 2016, momento em que o fluxo de demandas e o controle de *Work-in-progress - WIP* se voltou ao uso do Kanban, garantindo que demandas finalizadas fossem entregues sem que houvesse conclusão de um ciclo de 15 dias de desenvolvimento.

No entanto, apesar da nova forma de gerenciar as demandas, a qualidade das entregas sem a finalização e revisão técnica devida, se refletia em *bugs*, insatisfação dos clientes e pressões sobre entregas de emergência numa frequencia que não era saudável nem às pessoas e nem ao negócio. Isso acontecia porque nem o Scrum puro e nem o Kanban exigem ou definem como controlar qualidade técnica das entregas e, além disso, não havia definição no processo de desenvolvimento para atividades em torno de controle de qualidade.

Para complementar a adaptação e preencher as lacunas abertas ao se renunciar ao ciclo *time-boxed* do Scrum, foram usadas algumas boas práticas da eXtreme Programming. Dessa maneira, demandas passaram a ser definidas na presença de ao menos um desenvolvedor, além daquele que iria implementar e do P.O., havendo sempre revisão de código para promover transferência de conhecimento, posse coletiva de código e, ainda, Q.A. na etapa de codificação.

Além disso, o processo de entrega contínua passou a ser visto pelo time como parte integrante do processo de desenvolvimento da empresa. Os papéis do Scrum e as reuniões diárias foram mantidas, assim como as retrospectivas de *Sprint,* com intervalos agora semanais.

A fim de complementar o processo de desenvolvimento de software e integrar o time de negócio ao de desenvolvimento, em 2018, foi incluído o OKR como ferramenta de planejamento e execução de demandas estratégicas, comunicando a todo o time o *roadmap* do produto que está sendo desenvolvido.

Para desenvolver os MPS foi preciso uma composição de técnicas, ferramentas e frameworks, a fim de compor uma adaptação do Scrum o mais aderente possível ao modelo de negócio SaaS. A seguir são apresentados cada um dos elementos incorporados à cada versão da MPS.

5.1 O USO DO SCRUM NOS PROCESSOS DE SOFTWARE DA EMPRESA

Na empresa alvo do estudo, os valores de comprometimento, coragem, foco, abertura e respeito (SUTHERLAND e SCHWABER, 2020) são mantidos e tornam a entrega contínua algo possível, porém, para que isso aconteça, os papéis devem estar bem definidos. Por isso, o framework Scrum presume a presença de três atores durante o ciclo de vida de um projeto: *Product owner (P.O.), Devteam e Scrum master*. Os três em conjunto se complementam e permitem a realização dos valores citados.

O *Product owner* tem como propósito defender os interesses do cliente frente ao time de desenvolvimento de soluções, escrevendo as estórias de usuário e servindo de interlocutor para validações, negociações ou mesmo discussões de interesses entre todos os atores, incluindo o Scrum Master. Além disso, ele é o único responsável por definir o *backlog* do produto, responsabilidade essa que é definida e respeitada pelos times comerciais, sendo esse papel fundamental para que a priorização dos itens presentes no *to-do list* sejam sempre ordenados conforme interesses dos clientes, assim otimizando o valor do trabalho que o time de desenvolvimento realiza (SUTHERLAND e SCHWABER, 2020). No processo desenvolvido nesse trabalho, a presença do P.O. independente do time de desenvolvimento é fundamental para que as demandas estejam livres de discussões técnicas e se aproximem ao máximo da percepção abstrata do cliente.

O *Scrum master* tem como intuito, definido no próprio Scrum, fazer uma defesa do time de desenvolvimento e garantir que a entrega definida será executada, removendo impedimentos e servindo como ponte entre P.O. e *DevTeam* (SUTHERLAND e SCHWABER, 2020). Dessa maneira, o S.M. se torna um líder no dia a dia do desenvolvimento, mas, além disso, promove as melhorias necessárias de estrutura para que os times não tenham dificuldades em trabalhar as demandas conforme o P.O. as define. Esse compartilhamento entre diferentes DevTeam torna-se particularmente importante para empresas que possuem mais de um time ágil trabalhando simultaneamente, devendo o S.M. deve estar presente em mais de um projeto, servindo assim de ponte entre o negócio representado pelo P.O., a organização e o time de desenvolvimento.

Por fim, em termos dos atores, o *Devteam* é a mola propulsora do desenvolvimento das soluções, consumindo o *backlog* definido pelo P.O. e contando com até 9 integrantes multidisciplinares. Esse ator possui a prerrogativa de se comportar como um só, apesar de ser composto por várias pessoas, sendo assim uma equipe, em sua essência, sem hierarquia ou

submissão entre membros, bem como possuindo responsabilidade solidária (SUTHERLAND e SCHWABER, 2020). Além disso, o *Devteam* nunca é subdividido, sendo considerado assim, indissociável.

No que diz respeito à empresa alvo, durante a primeira melhoria do processo de software, a MPS foi executada com o apoio de uma consultoria externa e, durante o período de dois anos, foi aplicado o Scrum puro, com todos os papéis, eventos e valores. Neste intervalo de tempo o controle sobre demandas foi melhorado, resultando em previsibilidade quanto ao prazo de entrega de demandas para P.O. e clientes, tratamento de impedimentos na atividade de desenvolvimento e diminuição de volatilidade de entrega dos times, aumento no ritmo e de demandas entregues. No entanto, foram observadas também algumas dificuldades no uso do Scrum na empresa alvo. São elas:

- Papéis sobrepostos: *ScrumMaster* incluso dentro do DevTeam e em algumas situações, ocupando o papel de P.O., havendo confusão de responsabilidades e conflito de interesses dentro do processo de desenvolvimento (ex. defender o time e concomitantemente o cliente);
- Cerimônias e atividades negligenciadas: o Planejamento de Sprint deixou de ser importante pois as demandas eram empurradas da lista *to-do* em fluxo contínuo. Além disto, as demandas não eram mais validadas em cerimônia de encerramento de *sprint*, mas sim pelos times de qualidade que liberavam a demanda ainda antes do final da Sprint, por pipelines de integração contínua;
- Sprint não definida: com entregas parciais, e sem cerimonias de review, deixou de haver um compromisso com o intervalo das *sprints* para entregas;
- Trabalho em progresso (WIP) limitado a *Sprint*: em caso de finalização das atividades do *backlog* da *Sprints*, presume-se que os times estarão livres das demandas negociadas e serão liberados de suas atividades, o que pode gerar um impacto comercial e financeiro negativo. No entanto, o que se espera no modelo SaaS é um consumo constante e ininterrupto das demandas, não se limitando ao *backlog timeboxed*.

Durante o segundo ciclo MPS, o Scrum passou a ser utilizado parcialmente, estando todos os atores previstos no *framework* presentes no processo de desenvolvimento. Porém, foram eliminados três eventos previstos na definição formal:

- *Sprint Review:* tornou-se desnecessária, haja vista que o processo de desenvolvimento de software como serviço presume entrega contínua e reação imediata ao mercado em caso de alterações de escopo, exigindo que as equipes entreguem valor de forma imediata com a qualidade validada pelo time de Q.A.
- *Sprint Planning:* foi descartada, levando em consideração que o processo envolve o uso de Kanban e integração contínua, sem a janela de desenvolvimento e entrega de demandas com intervalo fechado, sendo o escopo de entregas livre e contínuo alimentado pelo *backlog* do produto.
- *Sprint:* foi eliminada, tendo em vista que o Kanban oferece fluxo contínuo de demandas, com *Work-in-progress* pleno e entrega contínua.

No entanto, foram mantidas as reuniões de retrospectiva quinzenais para avaliar melhorias em torno do processo com apontamentos de aspectos relativos ao sistema, além da reunião diária para relato dos impedimentos, de tal maneira que o Scrum Master garantisse a fluidez das demandas sem percalços.

5.2 KANBAN

Como forma de gerenciar as demandas no seu fluxo de execução e preencher a lacuna aberta pela ausência das *Sprints* foi adotado o Kanban, tornando o fluxo de demandas contínuo e não limitado à janela de tempo negociada. Dessa maneira, os P.O. puderam contar com flexibilidade para compor as entregas da maneira que o mercado e os serviço exigiam, sem obrigatoriamente executar eventos de planejamento, cancelamento e fechamento de entregas.

A principal diferença ao aplicar o Kanban frente ao Scrum no processo implementado foi oficialmente permitir a entrada de novas demandas na esteira de desenvolvimento, ou seja, a inclusão ou exclusão de itens do *backlog* para a coluna de “em execução” tornou-se possível a qualquer momento. Dessa maneira, o P.O. mantém as demandas em ordem de importância para que o fluxo contínuo ocorra sempre orientado ao valor a ser entregue pelo *Devteam* ao negócio.

O uso do Kanban, como afirma Ahmad et al. (2016), torna o processo mais leve sem destoar dos princípios ágeis, no aspecto de ser orientado a mudanças rápidas. Na tabela a seguir apresenta-se o resumo das diferenças entre o Scrum e Kanban e observações acerca de como cada uma delas foi agregada ao SaaS Process.

Tabela 2 - Kanban vs Scrum

Kanban	Scrum	Aplicação do SaaS Process
Não há papéis definidos	P.O., S.M. e Devteam	Papéis do Scrum mantidos para eleger responsabilidades nos times.
Entrega contínua	*Sprint* com tempo fechado	Para adequação do processo ao modelo de negócio SaaS, a entrega contínua foi definida como regra sem que houvesse necessidade de se esperar o final da *Sprint* para lançamento de features.
Itens são consumidos do *backlog* um a um	Itens são consumidos do *backlog* em blocos para cada Sprint	Os itens de backlog passam a ser continuamente consumidos pelo fluxo de atividades do Kanban buscando-se o WIP pleno.
Mudanças podem ocorrer a qualquer tempo	Não há mudanças em meio à Sprint	As mudanças são aceitas para que o cliente/usuário seja atendido dentro do processo de forma aderente ao modelo de negócio.
Usa entrega de valor ao cliente como métrica de planejamento e melhoria	Usa velocidade como métrica de planejamento e melhoria	Em se tratando de SaaS, o valor agregado ao serviço é mais importante que a quantidade de pontos trabalhados pelo time.
Adequado a ambientes com mudança de prioridades constantes	Adequado a ambientes com mudança de demandas a serem entregues em blocos e que não mudam	Com um comportamento de consumo volátil, as necessidades dos usuários mudam constantemente e devem ser acolhidas em forma de alteração de prioridades, como o Kanban permite.
Compromisso com as demandas espontâneas	Compromisso do time firmado pela Sprint	O compromisso do time existe sobre todo o backlog e não apenas sobre aquela parcela definida em uma Sprint, sendo possível entregar software espontaneamente, sem delimitar escopo dentro de um intervalo de tempo fixo.
WIP pleno e constante garantido pelo fluxo	WIP negociado por meio da Sprint	Como artifício para produção perene e sem interrupções, foi adotado o quadro Kanban que dá fluidez ao WIP, tornando-o aderente ao modelo SaaS.
Quadro de demandas constante e contínuo	Quadro reiniciado a cada Sprint	Assim como o modelo de negócio é contínuo e fluido na entrega de solução e competitividade, assim também é o Kanban, com seu quadro que define um fluxo constante de demandas.

Fonte: Adaptado de Kniberg apud Ahmad (2016)

As diferenças citadas entre os dois *frameworks* permitiram sobrepor as dificuldades encontradas junto ao *Devteam* na forma como as demandas eram geridas e entregues. Com uma metodologia menos preditiva que o Scrum puro e mais orientada ao workflow natural de entrega – análise, desenvolvimento, testes e *deploy* –, os conflitos se tornaram menos constantes entre o P.O. e o time, tornando também a experiência do cliente com a entrega de valor mais positiva.

Além disso, a flexibilidade que o *backlog* fluído oferece é fundamental para a entrega de serviço, no qual o mercado é o indicador de mudanças, especialmente se considerando aspectos como legislação, tendências de consumo e negociações que impõem uma condição de mudança de requisitos constante.

Dessa forma, o uso do Kanban trouxe ao processo a agilidade necessária às mudanças de *backlog*, bem como a fluidez para entrega contínua e, ainda, adequou o trabalho em progresso - WIP - para impor o maior ritmo possível aos times de desenvolvimento. Além disso, como efeito colateral, trouxe o compromisso aos P.O. em manter os *backlogs* constantemente atualizados.

5.3 XTREME PROGRAMMING (XP)

O uso de XP, como meio para introduzir ao time boas práticas de desenvolvimento, tem como origem um conjunto de princípios que buscam a transferência de conhecimento, cultura de equipe, entrega contínua e qualidade de código, por meio da participação de mais de um desenvolvedor em cada etapa do processo de codificação, como revisão de código e testes unitários. Assim, os times passaram a ter em seu processo de desenvolvimento algumas etapas incluídas com base nos princípios do XP: *code review, planning poker, pair programming,* comunicação informal, refatoração e código coletivo.

A adaptação do Scrum foi completada e pôde suportar o desenvolvimento de software para soluções SaaS, com gestão de atividades utilizando-se Kanban, divisão de responsabilidades através de Scrum e da execução das atividades com boas práticas do XP, buscando melhor qualidade de codificação.

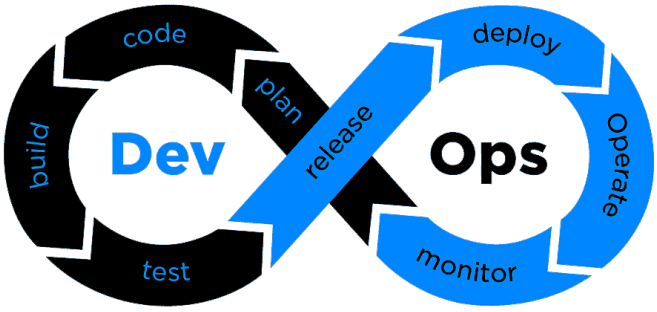
5.4 OKR e DevOPS

Além dos frameworks e das ferramentas apresentadas, o processo passou a contar com o apoio do OKR na fase macro de planejamento do *backlog,* com o intuito que todos os times possuíssem a visão estratégica da empresa, com objetivos e métricas de avaliação bem

definidos, tornando assim os esforços consonantes e direcionados às demandas do negócio, com Kanban, Scrum e XP sendo ferramentas para a entrega do resultado esperado.

Por fim, para complementar o conjunto de métodos, ferramentas e técnicas do SaaS *Process*, foi utilizado DevOPS e seus conceitos para a ponta operacional e de entrega da solução, como microsserviços, *cloud computing* e entrega contínua. Assim, o processo passou a suportar a cultura de agilidade imposta pelo negócio. Essas práticas de arquitetura e entrega contínua se unem ao desenvolvimento – DEV e às operações - OPS. A figura 9, a seguir, exemplifica o ciclo de vida de software orientado a DevOPS:

Figura 9- Ciclo de Vida DevOPS



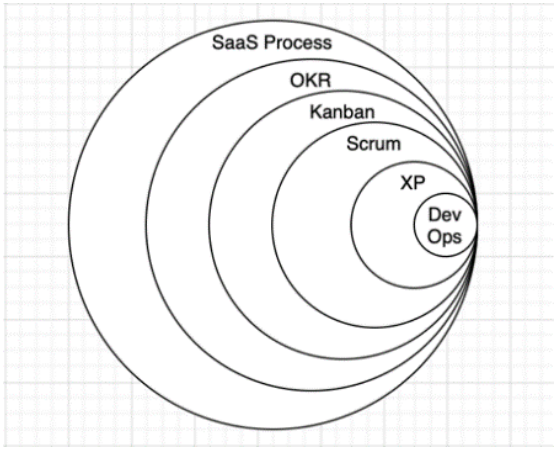
Fonte: Adaptado de Gunja (2021)

A Figura 9 ilustra as etapas de codificação com análise (*plan*), programação (*code*), compilação (*build*), testes (*test*) e sua conexão com operações a partir da liberação de versão (*release*), entrega (*deploy*), implantação e gerência (*operate*) e monitoramento.

5.5 RESULTADO DO SCRUM ADAPTADO

Como consolidação e nova versão da metodologia de desenvolvimento de software, foi compilado um conjunto de métodos, técnicas e ferramentas que definem concretamente o SaaS Process. Assim, o processo de desenvolvimento SaaS foi definido de ponta a ponta, incluindo desde a gestão estratégica até a entrega do SW ao usuário final. A seguir, a figura 10 apresenta uma representação gráfica do processo com os componentes mais estratégicos em maior evidência e os operacionais em menor escala:

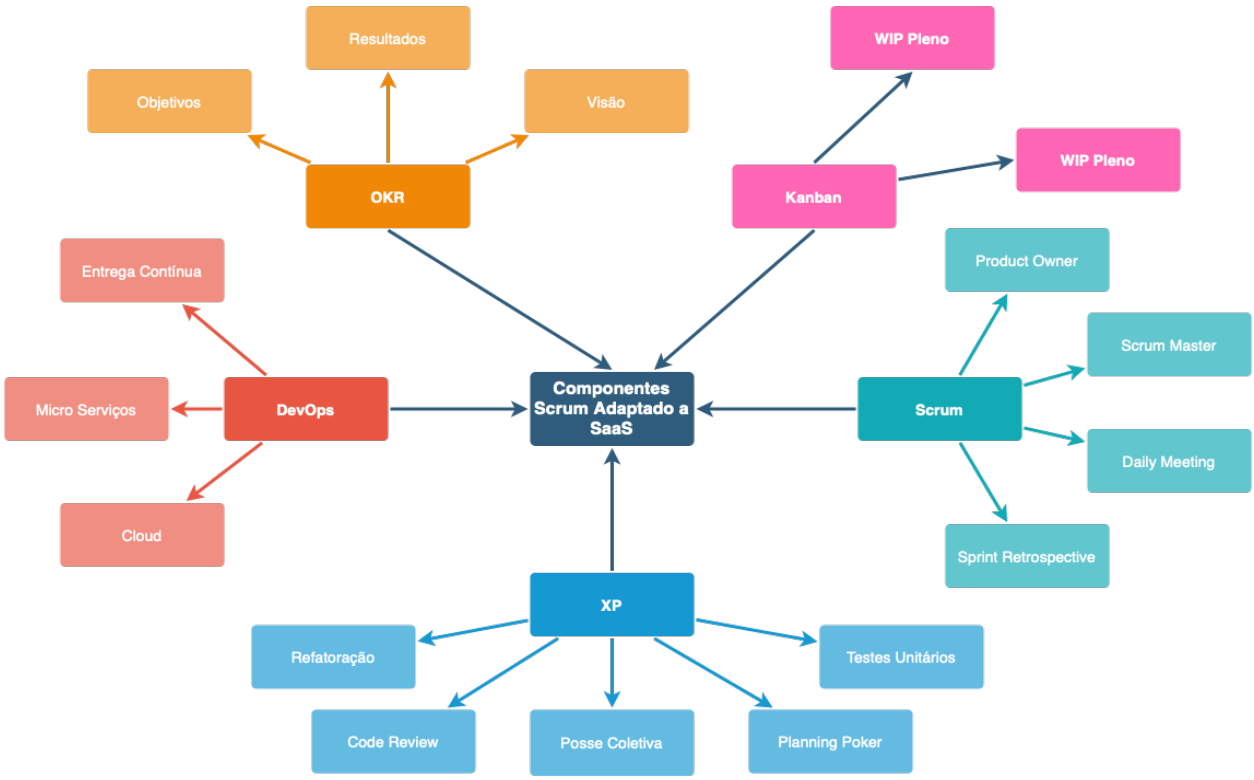
Figura 10 – Adaptação do Scrum - “SaaS Process”



Fonte: O autor (2021)

Assim, a empresa pôde avançar na entrega de soluções SaaS para diferentes mercados, estando comprometida com as mudanças que os seus clientes presumem serem atendidas em consonância com a rapidez com que seus concorrentes se mobilizam no atendimento de seus usuários. Em sequeência, a figura 11 contém um mapa mental dos elementos presentes em cada framework e as técnicas adotadas.

Figura 11 - Componentes do SaaS Process



Fonte: O autor (2021)

A figura 11 esboçada acima demonstra como cada um dos componentes contribuiu para a adaptação do *framework* ao SaaS. Observa-se que o Scrum cedeu seus papéis e cerimonia de retrospectiva, entregando responsabilidades e capacidade de avaliação para o processo na busca de melhoria contínua. O XP, com suas técnicas de análise de esforço, codificação e cultura de desenvolvimento, contribuiu para a entrega de software de maior qualidade, construído e mantido coletivamente. Além disso, o Kanban tornou-se a ferramenta para gestão do backlog e ordem de priorização dos trabalhos, permitindo institucionalmente que atividades fossem substituídas ou reordenadas conforme a demanda do negócio, de forma prática e natural. Aliado a isso, para entregar com agilidade, usou-se arquitetura e estrutura escalável suportada pelos princípios de DevOPS, com destaque para Entrega Contínua (CD) que engloba a automatização da execução de testes prevista pela Integração Contínua (CI) (HUMBLE e FARLEY, 2010). Para suportar a estratégia, prover transparência e manter todos os times com mesmos objetivos, aplicou-se o OKR - Objective and Key Results.

5.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A busca por elementos em diferentes ferramentas, métodos e técnicas ágeis, teve como objetivo uma adaptação do Scrum aderente ao modelo de negócio SaaS, buscando maior performance (mais entregas) e estabilidade (menos volatilidade) para o desenvolvimento e manutenção de um software provido como serviço. Bem como, ultrapassando o nível de gestão de tarefas e englobando níveis estratégicos e operacionais, com OKR e XP, por exemplo.

Essa união formaliza o que ocorre em grande parte da indústria e já é apontando como tendência pela literatura. Tal adaptação é uma busca em formalizar e dar um impulso na melhor definição de responsabilidades, atividades e resultados esperados de cada um dos métodos ágeis no contexto de desenvolvimento de software as a service.

O próximo capítulo discorre sobre a aplicação e avaliação da adaptação proposta, em ambiente real.

6 PESQUISA-AÇÃO – APLICAÇÃO DO SAAS PROCESS

Após observações do processo de software baseado no Scrum implantado na empresa alvo da pesquisa entre 2012-2014, em seu primeiro MPS, definiu-se a hibridização do processo, como discorrido no capítulo anterior, utilizando-se elementos de frameworks ágeis, como *eXtremeProgramming*, Kanban e OKR. Essa composição entre processos ocorreu por três anos, a constar entre 2015 a 2018, de forma *ad-hoc* e, entre 2019 e 2020, como pesquisa acadêmica, considerando como métricas os aspectos de performance: volatilidade, ritmo e média de pontuação.

6.1 RECORTES DE TEMPO

Durante o primeiro recorte de tempo, ao implantar a metodologia Scrum, foi definido duas semanas como intervalo de Sprint. Dessa forma, para também manter a consistencia na comparação entre recortes, manteve-se o intervalo de comparação para os demais recortes. Por isso, foi comparado o efeito da inclusão do XP e Kanban avaliando-se em recortes de quinze dias. Dessa maneira, pode-se verificar as melhoras e pioras sob o aspecto de performance obtidas no time.

O intervalo de tempo entre janeiro de 2019 a novembro de 2020 forneceu 46 *sprints* para comparação. Essas sprints foram comparadas a outros dois intervalos: janeiro de 2015 a novembro de 2016 e janeiro de 2017 a novembro de 2018. A comparação entre três diferentes intervalos de tempo objetivou avaliação dos MPS dentro da empresa sob dois critérios: mesmo tempo entre os recortes e tempo suficiente para haver mudanças perenes e não apenas imediatas. Os intervalos de aplicação de cada MPS foram:

- MPS 1.0: entre 2013 e 2015, após a implantação total do Scrum como framework do primeiro processo de desenvolvimento;
- MPS 2.0*:* entre 2016 e 2018, aplicando-se o XP e o Kanban;
- MPS 3.0: entre 2019 e 2020, aplicando-se OKR.

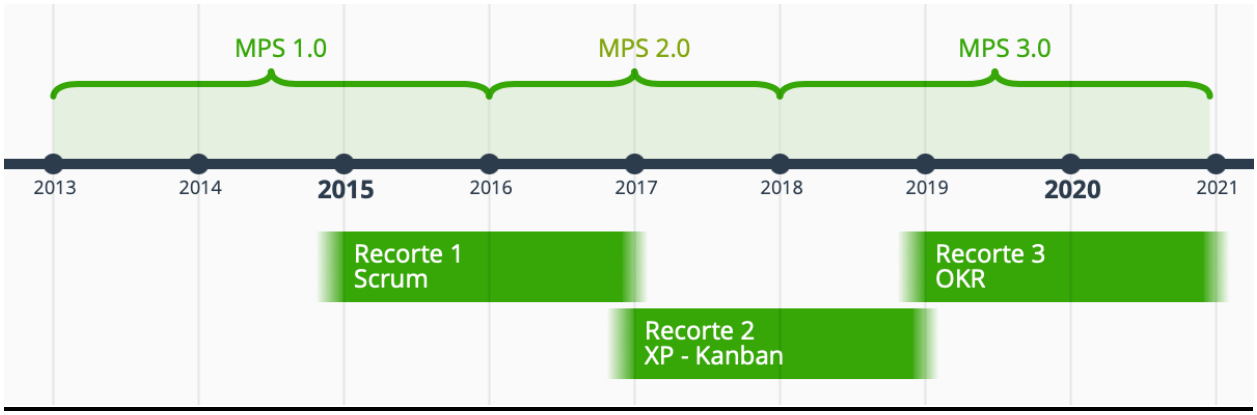
Somando os períodos pôde-se visualizar a evolução entre o Scrum e a adoção de novas técnicas e frameworks em três períodos distintos, com a mesma quantidade de recortes (46 *sprints*):

- Entre 2015 e 2016: a fim de avaliar a performance do Scrum e melhorias necessárias;

- Entre 2017 e 2018: objetivando monitorar performance do uso de XP e Kanban combinados ao Scrum;
- Entre 2019 e 2020: visualizando o impacto na performance do uso de OKR.

A diferença de anos entre os recortes de aplicação dos MPS e de avaliação ocorre pela necessidade de se avaliar o resultado da melhoria de processo somente após sua finalização. Dessa forma, a MPS aplicada entre os anos de 2013 e 2015 só pôde ser avaliada entre 2015 e 2016. O gráfico abaixo visa ilustrar essa sobreposição de períodos:

Gráfico 3- Recortes em timeline



Fonte: O autor (2021)

6.2 MÉTRICAS DE PERFORMANCE

Foram analisados aspectos voltados à performance das entregas, essencialmente esforço em Story Points, sem medições quanto à qualidade, dívida técnica ou aspectos subjetivos relacionados a cumprimentos de requisitos. Dessa maneira, toda a pesquisa-ação conduzida no último período comparado, de 2019 a 2020, foi avaliada sob os mesmos aspectos, sendo eles:

- Velocidade: a partir da técnica de mensuração ágil *Planning Poker*, os desenvolvedores em conjunto definem o esforço em *Story Points,* sem grandeza definida, sendo passível de ser comparada entre si e derivada em horas por meio de base histórica. Quanto maior a velocidade, melhor a performance.
- Média de velocidade: a partir dos pontos desenvolvidos individualmente é obtida a média do time ou do próprio desenvolvedor ao longo do tempo, com o intuito de comparar a performance em diferentes recortes de tempo. Quanto maior a média, melhor a performance do time.

- Média do *Lead Time*: a partir do início do desenvolvimento de uma demanda até seu final, pode-se comparar a eficiência do processo como um todo quanto à entrega de um produto pronto dentro da Definição de Pronto (*Definition of Done – DoD)* do time e do cliente. Espera-se que o *Lead Time – tempo entre o início da demanda e sua entrega* - seja sempre o menor possível. Importante frisar que essa métrica não se confunde com o esforço, pois pode haver interrupções em meio ao desenvolvimento, aumentando o Lead Time sem alterar o esforço da demanda.
- Quantidade de itens desenvolvidos: naturalmente inverso em quantidade ao Lead Time, essa métrica é utilizada para avaliação de times que utilizam Kanban. Quanto maior quando o *Lead Time* é menor a quantidade de itens entregues. Nesse quesito, quanto mais itens desenvolvidos, melhor.
- Volatilidade: flutuação entre a quantidade de tarefas entregues entre as *sprints* comparadas. A volatilidade deve ser controlada para que os clientes possam ter previsibilidade sobre as entregas ao longo do tempo. Quanto menor a volatilidade, melhor o ritmo e constância de entregas.

6.3 MPS 1.0

A implantação da primeira melhoria de processo de software foi executada por uma consultoria externa, ao longo de doze meses de trabalho, entre os anos de 2012 e 2013. Nesse período, foi indicado à diretoria, colaboradores e estagiários das áreas de desenvolvimento de software os elementos constituintes do Scrum e introduzido o uso da ferramenta PivotalTracker ® para rastreamento, controle de *sprints* e monitoramento de performance dos desenvolvedores.

Durante o período de implantação melhoria, diversos comportamentos do desenvolvimento *ad-hoc* de software foram combatidos, a exemplo da mudança constante de prioridades, falta de comunicação das necessidades dos clientes sob a ótica do levantamento de requisitos, gestão de escopo e ausência de documentação das etapas do desenvolvimento executado – análise, desenvolvimento, testes, validação e entrega do software.

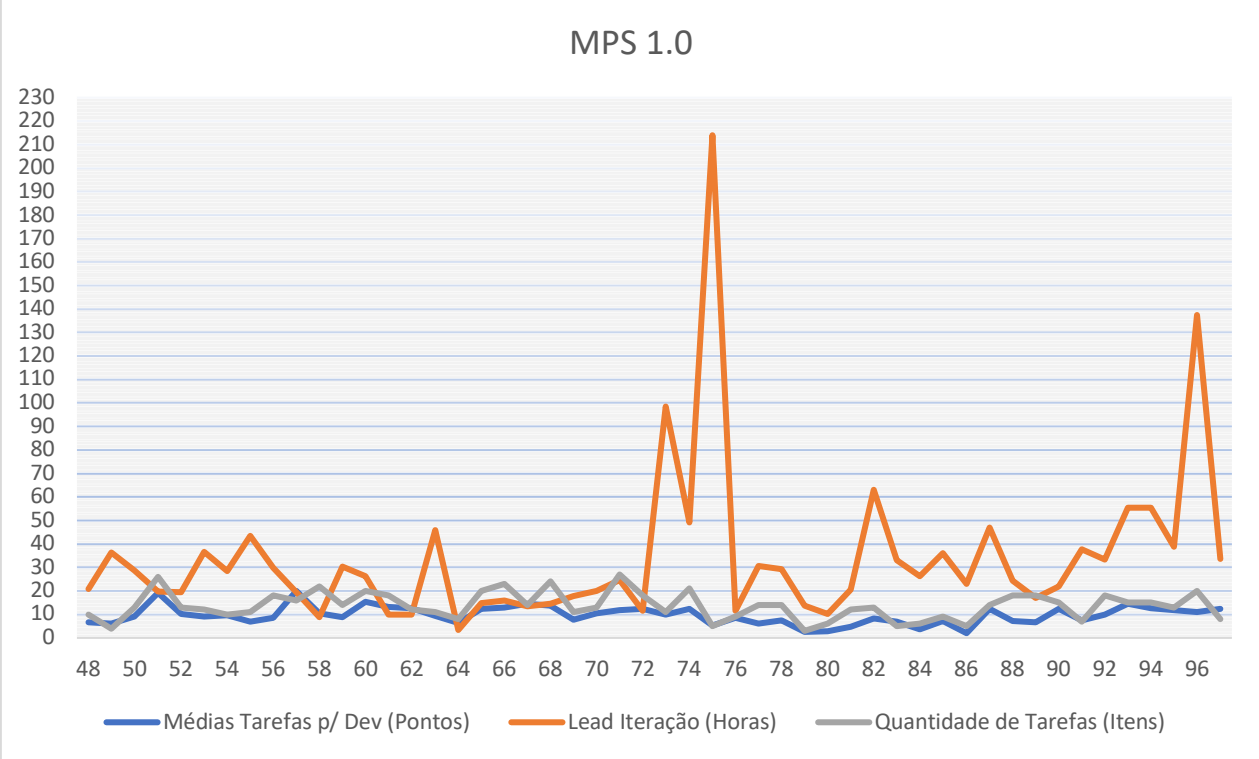
A transformação cultural ocorreu em dois anos, sendo ao longo do primeiro ano acompanhada pela consultoria, culminando com todas as etapas do desenvolvimento, eventos do Scrum e papéis aplicados plenamente. Desta forma, as *sprints* executadas baseavam-se em sua totalidade e foram utilizadas como ponto de partida para as melhorias subsequentes.

Ao longo de quarenta e seis *sprints,* cada uma com quinze dias, a performance dos times foi avaliada e se percebeu uma grande volatilidade entre os recortes, além de um *LeadTime* inconstante, reflexo da flutuação das velocidades atingidas pelos desenvolvedores na entrega dos itens dos *backlogs* de Sprint.

Em reuniões de retrospectiva entre gerentes e times durante o intervalo observado, percebeu-se a constante alteração de prioridades dos itens de *backlogs* por diversas vezes sem mudança formal na Sprint (desenvolvimento *ad-hoc*), além da falta de clareza em requisitos, a falha na estimativa de *StoryPoints* e a exigência do processo quanto a eventos de replanejamento de *Sprint* que, em conjunto consumiam tempo e energia dos times. Esses fatores foram determinantes na visão do autor para a ausência de resultados satisfatórios nas métricas analisadas.

Sendo assim, pode ser observado no gráfico 2 a flutuação constante de todas as métricas avaliadas, com mais de 100% de variação em quantidade de tarefas e incidência maior na oscilação do *LeadTime,* de mais de 2000%.

Gráfico 4- Performance time – Melhoria do processo de software 1.0



Fonte: O autor (2021)

6.4 MPS 2.0

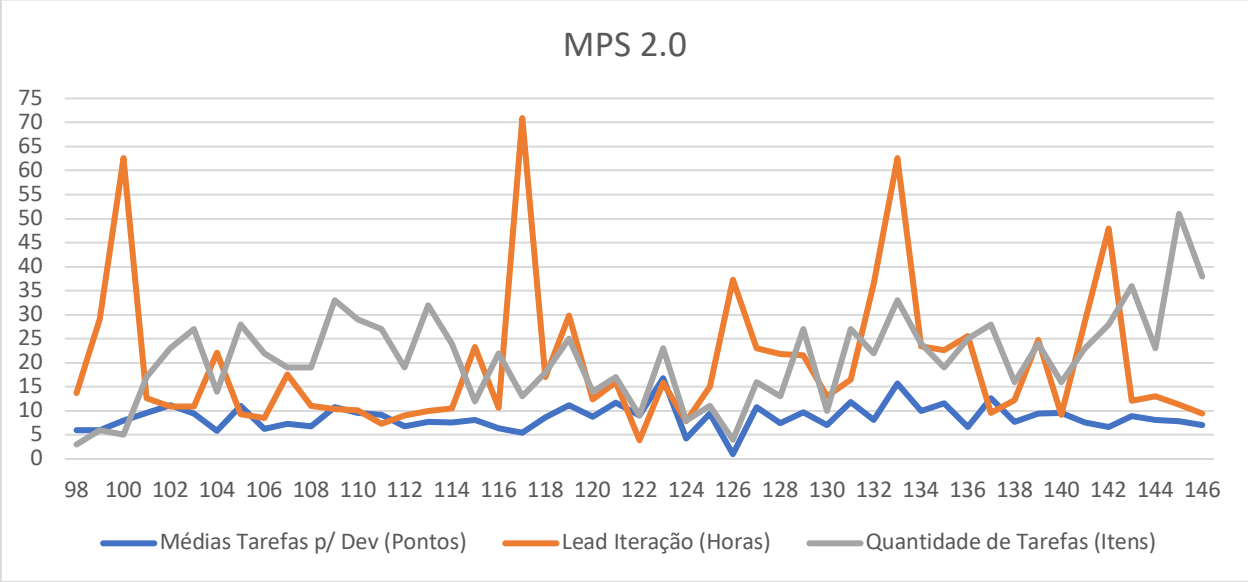
Após a avaliação de possíveis melhorias dentro do processo de desenvolvimento implementado, decidiu-se implantar o Kanban como ferramenta de controle de *Backlog*. Isto permitiu mudanças de inclusão, reposicionamento e exclusão de features do backlog como algo possível e esperado, sem estresses ao processo ou eventos de alteração de *sprints* para acolhimento de demandas, além de promover o *Work-in-process* pleno em busca de uma velocidade maior e mais constante por parte dos times.

A adoção do Kanban como mecanismo de gestão do *backlog* exigiu dos times o amadurecimento das técnicas de desenvolvimento, a fim de resolver impedimentos técnicos e diminuir o *LeadTime.* Como remédio, foram adotadas diversas técnicas de eXtreme Programming, tais como: Pair Programming, Metáforas, Integração Contínua (por meio do uso de ferramentas), Cliente no time e pequenas releases.

A junção do Kanban ao XP trouxe desafios em aspectos de consumo do *backlog,* como busca a ativa e constante de demandas na lista de *to-do* e na colaboração entre os membros para efetivamente entregar resultado em equipe. Esta modificação das atividades do processo e transformação das rotinas de trabalho ocorreu em 2015, tornando-se plena apenas no ano seguinte em que foi mensurada.

Após a inclusão das novas atividades, as reuniões de retrospectiva demonstraram a dificuldade dos times em perceber a ordem de prioridade dos itens de *backlog* e sua conexão com a estratégia da empresa. Para mitigar esse efeito foi introduzido o *Objectives and Key Results* (OKR) (NIVEN e LAMORTE, 2016) na iteração seguinte de MPS.

Gráfico 5- Performance time – Melhoria do processo de software 2.0



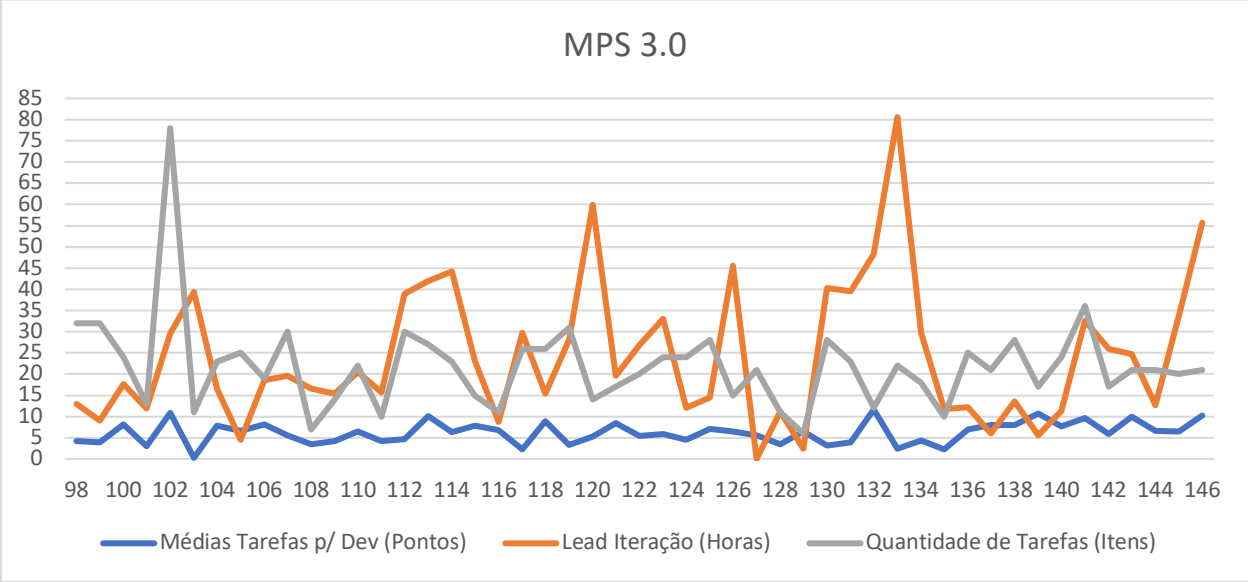
Fonte: O autor (2021)

6.5 MPS 3.0

Como última iteração na melhoria de processo de software, foi adotado o OKR para compor a pilha final do processo em vigor na empresa citada. A inclusão do OKR iniciou-se em 2019, passando a ser utilizada para definir e estimar visões de médio e longo prazo, como planejamentos semestrais e estratégia anual para áreas comerciais e de vendas.

Quando os times e a gerência estavam habituados à nova técnica, passou-se a utilizá-la nas definições de projetos de desenvolvimento, condensando os itens de um mesmo projeto sob um mesmo objetivo, claro e definido, além de fornecer métricas para avaliação do resultado. Assim como outras mudanças no processo de desenvolvimento, existiu uma janela de adaptação dos times e gerências às novas atividades. Durante as quarenta *sprints* avaliadas, o time se encontrava em processo de amadurecimento do MPS 3.0, podendo haver melhorias ou pioras na performance à medida que se habitua o uso do OKR. Essa é a versão atual e continua vigente até a finalização desse trabalho.

Gráfico 6- Performance time – Melhoria do processo de software 3.0

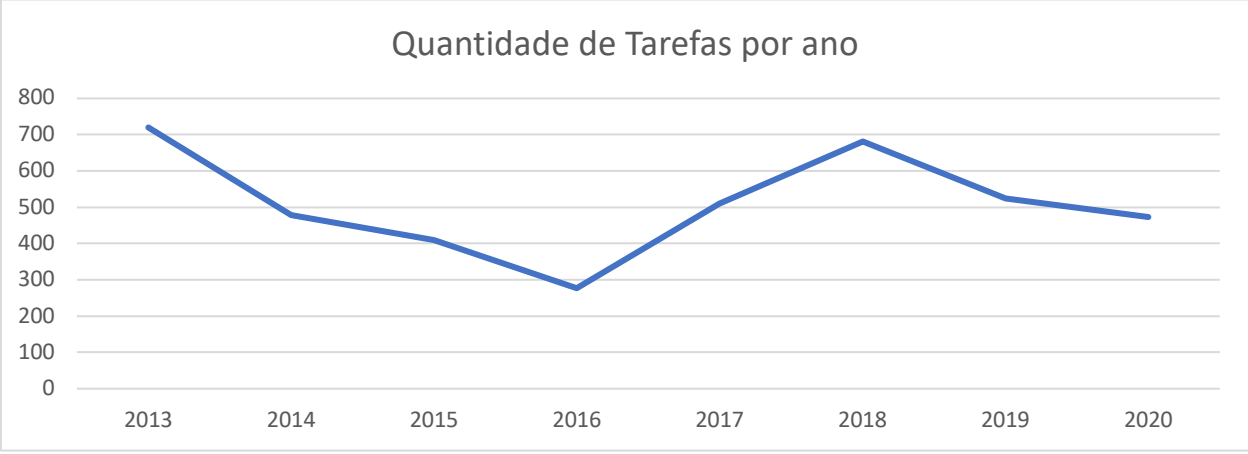


Fonte: O autor (2021)

6.6 RESULTADOS OBTIDOS

Durante as MPS executadas, foram monitorados e comparados os efeitos das alterações de processo frente a performance dos times. Em ocasiões de transição foram vistas oscilações que se refletiram em maior volatilidade nos pontos de estória produzidos, itens produzidos e *leadtime*. A justificativa está na adaptação dos times às novas atividades e na perda de energia produtiva envolvida nesse aprendizado e acomodação. A seguir, o gráfico 7 demonstra a volatilidade ao longo dos anos sobre a métrica de quantidade de tarefas executadas:

Gráfico 7- Performance time – Quantidade de tarefas executadas por ano



Fonte: O autor (2021)

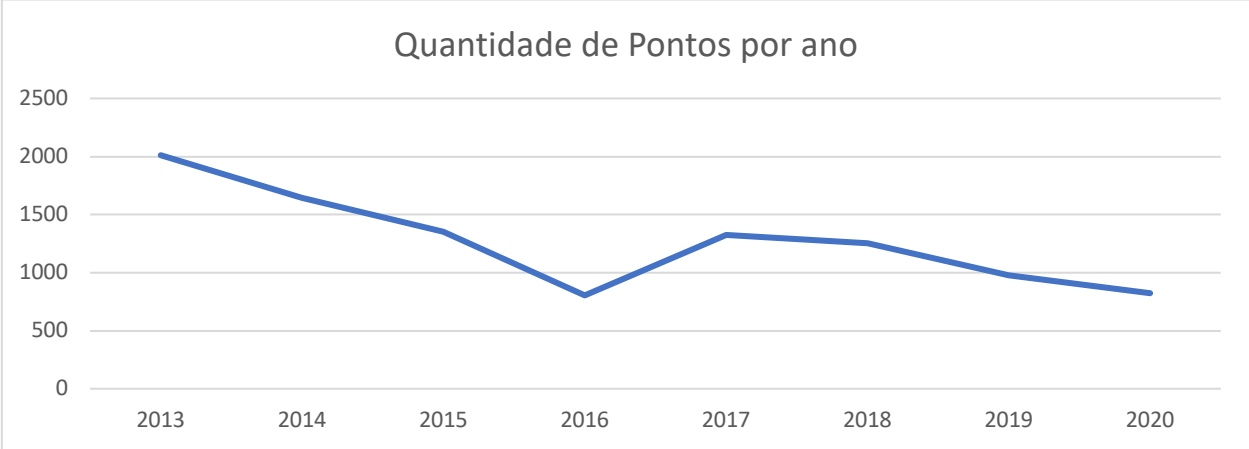
O reflexo mais contundente da mudança dos processos ocorreu no *LeadTime*, indicando uma possível associação entre as técnicas do XP, a transparência do Kanban e a visão dada pelo OKR à otimização da produção do software. O Gráfico 6 mostra o reflexo durante os períodos de mudança, resultando em uma maior quantidade de tarefas, mas com *LeadTime* estabilizado. Um exemplo é o período pós o 2017, com uma curva sem picos e vales, como visto nos anos anteriores. O resultado na quantidade de tarefas, vista no gráfico 7, num patamar médio mais alto durante períodos em que o *LeadTime* esteve mais estável é esperado e pode ser inferido comparando-se os gráficos.

Gráfico 8- Performance time – variação do Lead por ano

Fonte: O autor (2021)

Um efeito importante em termos de performance foi a quantidade de pontos de estória produzidos ao longo das iterações. O efeito foi a princípio inesperado, mas ao analisá-lo com cuidado, percebe-se que mesmo quando a quantidade de pontos diminuiu, a quantidade de tarefas entregues cresceu. Essa correlação entre ambos pode evidenciar a entrega de valor (quantidade de itens) com menor esforço (quantidade de pontos), aumentando concretamente o benefício versus o custo das demandas produzidas. As causas podem estar relacionadas ao amadurecimento dos processos e melhoria da qualidade dos requisitos. O gráfico 9 demonstra tal comportamento, sobrepondo-o ao gráfico 8:

Gráfico 9- Performance time – quantidade de pontos por ano



Fonte: O autor (2021)

6.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Concluiu-se através da observação das métricas monitoradas, que a mudança de processo de desenvolvimento tem, a princípio, um resultado negativo que é mitigado ao longo do processo de ajustamento das atividades. Os impactos positivos foram percebidos durante as intervenções como resultado do amadurecimento da empresa em perceber os pontos fracos de cada momento e aceitar resolvê-los com o uso das técnicas e ferramentas complementares aplicadas.

Vale ressalvar que a aplicação da pesquisa-ação, pode ter seu resultado influenciado pela presença do pesquisador e a ausência de aplicação em times em outros ambientes além do citado nesse estudo. Uma das maneiras buscadas para mitigar o possível comportamento enviesado foi a aplicação longitudinal (por um longo período) do estudo, com intuído das atividades aplicadas em cada um dos MPS fossem incorporadas e se tornassem naturais ao longo dos vinte e três meses de cada recorte.

7 CONCLUSÃO

Durante todo o curso do programa e escrita do documento, o autor esteve atento ao crescimento da adoção de modelos híbridos em empresas que promovem o uso de SaaS. Essa atenção ao tema no mercado de software teve como objetivo manter a relevância ao tema da pesquisa e estar atualizado quanto as diferentes abordagens. Durante tais incursões, percebeu-se uma corrente prática de adoção parcial do Scrum, devido a obrigatoriedade de seus eventos, que tem sido flexibilizado dentro do próprio framework pela Scrum Foundation, como a minimização da seção de cancelamento de Sprint (SUTHERLAND e SCHWABER, 2020).

A inclusão de diferentes níveis organizacionais no processo é considerado um ponto importante e que pode trazer uma contribuição sólida para construção de processos que avancem além do nível gerencial, incluindo pessoas estratégicas, bem como definindo tarefas operacionais de forma clara e incluindo-as no conjuntos de procedimentos, envolvendo concretamente toda a organização baseada em SaaS no desenvolvimento de software, desde a diretoria até os programadores, os analistas de infraestrutura e de sustentação.

Como conclusão, a pesquisa buscou compilar as necessidades e melhorias no processo de desenvolvimento de software de uma empresa local. Contudo foi constatada a necessidade de validação externa da pesquisa, o que se deu por meio de um questionário público enviado a empresas locais e filtrado pelo modelo de negócio SaaS. Assim, abre-se uma agenda de pesquisa voltada ao tema do trabalho, abrindo a discussão dentro da academia quanto a adequação dos métodos ágeis tradicionais além das fronteiras da indústria que já o faz de forma *ad-hoc*.

7.1 TRABALHOS FUTUROS

É importante a aplicação do SaaS Process em diferentes empresas de software provedoras de serviço como forma de avaliar a adaptação em diferentes ambientes e preencher vieses que invariavelmente tenham surgidos durante a pesquisa-ação, neste trabalho. Essa ação daria origem a novos MPS sobre o SaaS Process, tornando-o mais aderente a realidade de mercado e não apenas da empresa alvo ou empresas locais. Além disso, a etapa de entrevistas junto a gestores e times, deve ser executada, o que não foi feito, apesar de planejado em virtude do tempo reduzido para execução das demais etapas.

A validação do OKR como ferramenta para camada estratégica também é um ponto de atenção e, como sugestão de trabalhos futuros, deve ser melhor avaliado quanto ao seu impacto prático para times, empresa e demais etapas do processo.

7.1.1 Indicações de Trabalhos Futuros Relacionados ao MSL

Com base nos resultados obtidos neste mapeamento sistemático, algumas lacunas identificadas podem ser direções para condução de futuras pesquisas. Entre os principais destacamos:

- *Qual a relação entre os efeitos do uso de métodos ágeis sobre a manutenção de software no contexto acadêmico e também no mercado?*

Uma vez que esse mapeamento considerou apenas estudos conduzidos dentro das indústrias de software, faz-se necessária uma análise comparativa dos resultados reportados por pesquisas realizadas em contextos acadêmicos. As respostas para essa pergunta podem balizar o processo de tomada de decisão de usuários de métodos ágeis, que também estejam preocupados com os resultados obtidos em contextos onde pesquisadores tiveram algum controle sobre o ambiente. E portanto, ser possível replicar no mercado em cenários de imprevisibilidade no qual as empresas de software eventualmente estão inseridas.

- *Que motivos levam times de manutenção a optar por metodologias ágeis?*

Uma vez que ainda há um movimento tímido de times de manutenção que migram para o uso de métodos ágeis, cabe avaliar qual a motivação da escolha. Problemas recorrentes que times de manutenção enfrentam com métodos tradicionais e o que esperam com a mudança de paradigma podem auxiliar pesquisadores e praticantes a compreender o que pode ser melhorado nos métodos aplicados a manutenção.

- *Qual o efeito do uso de métodos ágeis sobre o tempo e/ou custo de manutenção de um produto de software?*

Dois dos principais problemas que motivaram o surgimento dos métodos ágeis foram projetos que recorrentemente eram entregues fora do prazo e do custo estabelecido com o cliente. Além disso, sob a ótica da manutenção, é esperado pelos clientes que as mudanças requisitadas sejam entregues com a maior brevidade possível. Estudar se os métodos ágeis

também são capazes de entregar valor aos clientes no contexto da manutenção de software pode ser um fator decisivo para a escolha desse paradigma pelos times de manutenção.

Além disso, projetos entregues dentro dos custos planejados são mais rentáveis para a empresa, podendo ser este um fator fundamental utilizado por gerentes e tomadores de decisão para a opção por métodos ágeis para os times de manutenção dentro da organização.

- *Que métodos ágeis trazem melhores resultados para o processo de manutenção de software?*

Os resultados obtidos por este mapeamento foram inconclusivos com relação a quais métodos ágeis melhor se adequam junto ao processo de manutenção de software. Auxiliando na migração de times que utilizem ciclos de vida tradicionais para adaptativos com métodos ágeis, indicando quais deles são mais adequados ao processo de manutenção e identificando quais elementos podem ser utilizados diretamente e quais precisam ser adaptados, incluídos ou removidos.

REFERÊNCIAS

AHMAD, M. Omair; MARKKULA, Jouni; OIVO, Markku. Kanban in software development: A systematic literature review. In: **Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference** on. IEEE, 2013. p. 9-16.

AHMAD, Muhammad Ovais et al. Transition of software maintenance teams from Scrum to Kanban. In: **System Sciences (HICSS), 2016 49th Hawaii International Conference on**. IEEE, p. 5427-5436, 2016.

AMAZON (Estados Unidos) (org.). **What is DevOps?** Disponível em: <https://aws.amazon.com/devops/what-is-devops/>. Acesso em: 27 set. 2020.

ANDERSON, David J.; CARMICHAEL, Andy. **Essential kanban condensed**. Blue Hole Press, 2016.

ARMBRUST, Michael et al. Above the Clouds: A Berkeley View of Cloud Computing. 2009. Disponível em: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28>. Acesso em: 17 jul. 2021.

AZIZ, Junaid; AHMED, Faheem; LAGHARI, Mohammad Shakeel. Empirical analysis of team and application size on software maintenance and support activities. In: **Information Management and Engineering, 2009. ICIME'09. International Conference on**. IEEE, p. 47-51, 2009.

BHATTACHARYYA, D. K. **Research Methodology.** Excel Books India, 2009.

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

CANFORA, G. *et al.* How changes affect software entropy: an empirical study. **Empirical Software Eng 19,** 1–38 (2014). https://doi.org/10.1007/s10664-012-9214-z

CAPILUPPI, Andrea et al. An empirical study of the evolution of an agile-developed software system. In: **Proceedings of the 29th international conference on Software Engineering**. IEEE Computer Society, p. 511-518, 2007.

CHAU, Thomas; MAURER, Frank. Knowledge sharing in agile software teams. In: **Logic versus approximation**. Springer Berlin Heidelberg, p. 173-183, 2004.

CHOUDHARI, Jitender; SUMAN, Ugrasen. Iterative maintenance life cycle using extreme programming. In: **Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on**. IEEE, p. 401-403, 2010.

CHO, V.; CHAN, A. An integrative framework of comparing SaaS adoption for core and non-core business operations: An empirical study on Hong Kong industries. **Information Systems Frontiers**, 17(3), p. 629-644, 2015.

CMS. **Selecting a Development Approach**. Department of Health & Human Services- USA, 2005. Disponível em: < https://www.academia.edu/22583615/SELECTING_A_DEVELOPMENT_APPROACH>. Acesso em: 17 set. 2020.

COCKBURN, Alistair. **Agile software development: the cooperative game**. Pearson Education, 2006.

COLEMAN, Don; LOWTHER, Bruce; OMAN, Paul. The application of software maintainability models in industrial software systems. **Journal of Systems and Software**, v. 29, n. 1, p. 3-16, 1995.

COREY, Ladas. **Scrumban:** Essays on Kanban Systems for Lean software development. Seattle, WA, USA: Modus Cooperandi Press, 2009.

DE MAGALHÃES, Cleyton VC et al. Investigations about replication of empirical studies in software engineering: A systematic mapping study. **Information and Software Technology**, v. 64, p. 76-101, 2015.

DIGITAL.AI. **15th Annual State of Agile Report**, 2021. Disponível em: <https://info.digital.ai/rs/981-LQX-968/images/RE-SA-15th-Annual-State-Of-Agile-Report.pdf>. Acesso em: 08 mar. 2022.

DOERR, John. **Measure what matters:** How Google, Bono, and the Gates Foundation rock the world with OKRs. Penguin, 2018.

DRAGGONI, Nicola et al. Microservices: How To Make Your Application Scale. A. K. Petrenko and A. **Voronkov** (Eds.): PSI 2017, LNCS 10742, p. 95–104, 2018.

DRUCKER, Peter F. "Management by objectives and self-control." *Practice of management.* Canada: Harper Collins Publishers, 1954.

DRUCKER, Peter F. "O futuro já chegou". ***Revista Exame***, ano 34, n.6, edição 710, p.112-126, janeiro/2000.

DYBÅ, Tore; DINGSØYR, Torgeir. Empirical studies of agile software development: A systematic review. **Information and software technology**, v. 50, n. 9, p. 833-859, 2008.

EASTERBROOK, S. et al. Selecting Empirical Methods for Software Engineering Research. In: SHULL, F.; SINGER, J.; SJøBERG, D. I. K. **Guide to advanced empirical software engineering**. [S.l.]: Springer-Verlag London, p. 285-311. 2008.

EETU, Kupiainen; MÄNTYLÄ, Mika V.;' ITKONEN, Juha. Using metrics in Agile and Lean Software Development: A systematic literature review of industrial studies. **Information and software technology 62**, p. 143-163, 2015.

ENGSTRÖM, Emelie; RUNESON, Per. Software product line testing–a systematic mapping study. **Information and Software Technology**, v. 53, n. 1, p. 2-13, 2011.

FITZSIMMONS, James A.; FITZSIMMONS, Mona J. **Administração de Serviços-: Operações, Estratégia e Tecnologia da Informação**. Amgh Editora, 2014.

FOTIADES, Georgios. **Agile project management approach applied in non-IT industry**. A manual to start adopting. Disponível em: <https://repository.ihu.edu.gr/xmlui/handle/11544/29884>. Acesso em: 17 jul. 2021.

FOX, Armando; PATTERSON, David A. **Construindo software como serviço (SaaS): uma abordagem ágil usando computação em nuvem**. Strawberry Canyon LLC, 2015.

FOWLER, Martin; HIGHSMITH, Jim. The agile manifesto. **Software Development**, v. 9, n. 8, p. 28-35, 2001.

GARTNER (org.). **Cloud Computing as Gartner Sees it.** Gartner's Application Architecture, Development & Integration Summit. 2009

GARTNER (org.). **Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17% in 2020**: iaas secures highest growth in 2020 due to data center consolidation. IaaS Secures Highest Growth in 2020 Due to Data Center Consolidation. 2019. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020>. Acesso em: 17 set. 2020.

GARTNER. **Predicts 2022**: SaaS Dominates Software Contracting by 2026 — and So Do Risks. 2021. Disponível em: < https://www.gartner.com/en/documents/4009001>. Acesso em: 17 mar. 2022.

GENE, Kim et al. *The* **DevOps Handbook**: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution, 2016.

GOTH, Greg. **Software-as-a-service**: The spark that will change software engineering? IEEE Distributed Systems Online 9.7, 2008.

HASSAN, Qusay F. "Demystifying cloud computing." **The Journal of Defense Software Engineering 1**, p. 16-21, 2011.

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

HE, Hao. "What is service-oriented architecture, 2003. **Disponível em:** <http://webservices.xml.com/pub/a/ws/2003/09/30/soa. Html>. >. Acesso em: 11 out. 2020.

HENDERSON, Fergus. **Software engineering at Google**: lessons learned from programming over time. Disponível em: <https://abseil.io/resources/swe_at_google.2.pdf >. Acesso em: 11 out. 2020.

HESSE-BIBER, S. N. **Mixed Methods Research**: Merging Theory with Practice, Guilford, New York, 2010.

HIGHSMITH, Jim. **Adaptive software development: a collaborative approach to managing complex systems**. Addison-Wesley, 2013.

HUMBLE, Jez; FARLEY, David. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**, 1st ed. Addison-Wesley Professional, 2010.

ISO (org.). **ISO 9000:2015 (en)**: quality management systems **.:** fundamentals and vocabulary. Quality management systems — Fundamentals and vocabulary. Disponível em: <https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en>. Acesso em: 11 out. 2020.

JAGLI, Dhanamma; YEDDU, Shireesha. CloudSDLC: Cloud Software Development Life Cycle. **International Journal of Computer Applications 168.8.** p. 6-10 2017.

JURIKRE, Amos O. et al. **Analysing the Obstacles in Agile Software Development Approach**: A Review. East African Journal of Information Technology. 5, 1 (Jan. 2022), 1-6. DOI:https://doi.org/10.37284/eajit.5.1.520.

KAN, Stephen H. **Metrics and models in software quality engineering**. Addison-Wesley Longman Publishing Co., Inc., 2002.

KEELE, Staffs et al. Guidelines for performing systematic literature reviews in software engineering. In: **Technical report, Ver. 2.3 EBSE Technical Report. EBSE**. sn, 2007.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1-26, 2004.

LAKATOS, Eva Maria; MARCONI Marina de Andrade. **Fundamentos de Metodologia Cientifica.** 7. ed. São Paulo: Atlas, 2010.

LI, Wei; HENRY, Sallie. Object-oriented metrics that predict maintainability. **Journal of systems and software**, v. 23, n. 2, p. 111-122, 1993.

MELO, Claudia de O. et al. The evolution of agile software development in Brazil. **Journal of the Brazilian Computer Society**, v. 19, n. 4, p. 523-552, 2013.

MERRIAM, S. B. **Qualitative research and case study applications in education**. São Francisco (CA): Jossey-Bass, 1998.

MICCO, John. **Continuous integration at google scale**: Developer Infrastructure. 2016. Disponível em: < https://www.slideshare.net/JohnMicco1/2016-0425-continuous-integration-at-google-scale>. Acesso em: 02 out. 2020.

MOREIRA, Renata Teles. **Um perfil de capacidade para a melhoria do processo em micro e pequenas organizações orientadas à manutenção e evolução de produtos de software**. 2015. Tese (Doutorado) - Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2015.

NBR ISO/IEC 9126-1:2003: **Engenharia de Software** – Qualidade de Produto. Parte 1: Modelo de Qualidade.

NIVEN, P. R.; LAMORTE, B. **Objectives and Key Results: Driving Focus, Alignment, and Engagement with OKRs**. Wiley Corporate F&A, 2016.

OLIVEIRA, T. et al. Understanding SaaS adoption: The moderating impact of the environment context. ***International Journal of Information Management***, *49*, 1-12, 2019

PALMER, Steve R.; FELSING, Mac. **A practical guide to feature-driven development**. Pearson Education, 2001.

PETERSEN, Kai et al. Systematic Mapping Studies in Software Engineering. In: **EASE**. 2008. p. 68-77.

PFLEEGER, Shari Lawrence; BOHNER, Shawn A. A framework for software maintenance metrics. Em: **Software Maintenance, 1990, Proceedings., Conference on**. IEEE, 1990. p. 320-327.

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

POPPENDIECK, Mary; POPPENDIECK, Tom. **Lean Software Development: An Agile Toolkit: An Agile Toolkit**. Addison-Wesley, 2003.

PRESSMAN, Roger S**. Engenharia de software**: uma abordagem profissional. 7 ed. Ed: McGraw Hill, 2011.

RIAZ, Mehwish; MENDES, Emilia; TEMPERO, Ewan. A systematic review of software maintainability prediction and metrics. Em: **Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement**. IEEE Computer Society, p. 367-377, 2009.

RUNESON, P.; HOST, M. Guidelines for conducting and reporting case study re- search in software engineering. Emp. Softw. Eng. Vol.14, No. 2, p. 131–164, 2008.

SANTOS, Mônica (ed.). **Saiba o que é e como implementar a metodologia OKR | Sebrae**. Disponível em: <https://www.sebrae.com.br/sites/PortalSebrae/artigos/gestao-de-metas-como-

implementar-a-metodologia-okr,a67875d380a9e410VgnVCM1000003b74010aRCRD>. Acesso em: 02 out. 2020.

SARAH, Saleh. **Everything you need to know about Agile methodologies.** Disponível em: **<**https://jexo.io/blog/everything-you-need-to-know-about-agile-methodologies/#:~:text=There%20are%20over%2050%20agile,are%20a%20few%20of%20them**>.** Acesso em: 02 Jul. 2021.

SINGH, A.; SINGH, K.; SHARMA, N. **Knowledge Management:** the agile way. Information and knowledge management, vol. 3, p. 143–152, 2013.

SOFTWARE ADVICE. **Cloud vs. On-Premise Deployment, 2008-2014 BuyerView.** Disponível em: < https://www.softwareadvice.com/buyerview/deployment-preference-report-2014/>. Acesso em: 02 Jul. 2021.

SOMMERVILLE, I. (2019). **Software Engineering**. 10ª edição. Pearson, 2019.

SUGIMORI, Y. et al. (1977). Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. THE **INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH**, 15:6, 553-564, DOI: 10.1080/00207547708943149.

SUSMAN, Gerald I.; EVERED, Roger D. An Assessment of the Scientific Merits of Action Research. **Administrative Science Quarterly**, Vol. 23, No. 4 (Dec., 1978), pp. 582-603.

SUTHERLAND, Jeff; SCHWABER, Ken. "The scrum guide." **The definitive guide to scrum: The rules of the game**. Scrum. org268, 2020.

STAPLETON, Jennifer. **DSDM, dynamic systems development method: the method in practice**. Cambridge University Press, 1997.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. (2002), Introdução à Engenharia de Software Experimental. COPPE/UFRJ, Rio de Janeiro, **Relatório técnico**: RT-ES-590/02. Disponível em: <https://www.cin.ufpe.br/~scbs/experimental/IntroducaoExperimentacao.pdf>. Acesso em: 02 jul. 2021.

TSAI, Weitek; BAI, Xiao Ying; HUANG, Tu. Software-as-a-service (SaaS): perspectives and challenges**. SCIENCE CHINA,** Berlin, Vol. 57, May 2014.

VERSION ONE. **10th Annual State of Agile Report**, 2016. Disponível em: <http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf>. Acesso em: 02 out. de 2020.

WODTKE, Christina. **Introduction to OKRs**. O'Reilly Media, 2016. Disponível em: <https://www.oreilly.com/content/introduction-to-okrs/>. Acesso em: 02 out. de 2020.

YIN, R. K. **Estudo de caso:** planejamento e métodos. Tradução de Daniel Grassi. 3a. ed. Porto Alegre: Bookman, 2005.

ZHOU, Yuming; LEUNG, Hareton. Predicting object-oriented software maintainability using multivariate adaptive regression splines. **Journal of Systems and Software**, v. 80, n. 8, p. 1349-1361, 2007.

APÊNDICE A – PROTOCOLO DO MAPEAMENTO SISTEMÁTICO DA LITERATURA

A fim de encontrar estudos relevantes para nossa pesquisa, foi utilizada uma estratégia de busca combinada, com busca automática e manual. Para execução da busca automática foram consideradas as seguintes bibliotecas digitais:

- ACM http://dl.acm.org/
- Scopus http://www.scopus.com
- IEEE Xplore http://ieeexplore.ieee.org
- ScienceDirect http://www.sciencedirect.com/

Para executar a busca nos engenhos supracitados nós definimos uma *string* de busca. No primeiro momento, definimos uma *string* robusta, considerando também os sinônimos dos principais termos da questão de pesquisa. Contudo, o número extremamente reduzido ou mesmo a ausência de resultados na execução da busca, nos motivou a tornar a *string* tão simples quanto possível.

Com essa simplificação, o objetivo foi mitigar os riscos de estudos relevantes para pesquisa não serem encontrados devido à complexidade da *string* e a forma particular de cada biblioteca digital interpretar os termos da busca. Assim, a string resultante, considerando apenas os termos em inglês, foi a seguinte: "*agile*" AND ("*software maintenance*" OR "*software maintainability*").

A string supracitada foi aplicada considerando os metadados (isto é, título, resumo e palavras-chave) dos estudos em todos os engenhos de busca utilizados.

De forma complementar à busca automática, lançamos mão da busca manual a fim de identificar quaisquer estudos relevantes que não puderam ser alcançados por meio da busca automática. A busca manual foi feita em duas frentes: em anais de conferências e jornais e nas referências de estudos selecionados.

No primeiro caso, foram considerados os três últimos anos (2014-2016) dos seguintes jornais e conferências:

- *IEEE International Conference on Software Maintenance*;
- *European Conference on Software Maintenance and Reengineering*;
- *Journal of Software Maintenance and Evolution*;
- *Agile Development Conference*.

Critérios de inclusão e exclusão

Para seleção dos estudos primários, definimos que seriam elegíveis aqueles que, obrigatoriamente, atendessem aos seguintes critérios de inclusão:

- estudos abordando os efeitos do uso de metodologias ágeis de desenvolvimento no processo de manutenção de software;
- estudos realizados dentro do contexto da indústria de software.

Após a análise de título e resumo dos trabalhos e considerando os critérios de inclusão, foram identificados 47 potenciais estudos relevantes. Após esta pré-seleção, estabelecemos que seriam descartados estudos que obedecessem a pelo menos um dos seguintes critérios de exclusão:

- estudos realizados antes de 2001;
- estudos escritos em outro idioma que não fosse o inglês;
- estudos não acessíveis na web;
- teses, dissertações, slides, notas de aula, resumos estendidos e estudos incompletos;
- estudos secundários;
- estudos duplicados;
- estudos realizados majoritariamente ou completamente em contexto acadêmico;
- estudos com resultados não discutidos ou citados de maneira superficial sobre a influência do uso de métodos ágeis no processo de manutenção de software.

A restrição de estudos a partir de 2001 (primeiro critério de exclusão) se dá apenas pelo fato do Manifesto Ágil (FOWLER e HIGHSMITH, 2001) ter sido publicado neste ano. Portanto, para o contexto do presente trabalho, estudos publicados anteriormente a esta data não foram considerados. Ressaltamos ainda que nesta etapa da seleção, os autores fizeram uma leitura mais aprofundada dos estudos, passando a considerar também as seções de introdução, metodologia, resultados e conclusões. Finalmente, após a aplicação dos critérios de exclusão, foram excluídos 40 estudos, resultando em um conjunto final de 7.

Análise de qualidade

Após a fase de seleção, os 7 estudos remanescentes foram submetidos aos seguintes critérios de qualidade, baseados nos critérios utilizados por Dybå e Dingsøyr (2008):

i. O estudo reporta dados de uma pesquisa empírica.
ii. A definição dos objetivos da pesquisa está posta de forma clara.
iii. O contexto em que foi conduzido a pesquisa é descrito de forma clara.
iv. A metodologia utilizada está alinhada com os objetivos da pesquisa.
v. A estratégia de recrutamento utilizada foi adequada.
vi. Os impactos da pesquisa no tocante aos resultados obtidos foram discutidos pelos autores.

Além destes critérios, incluímos também mais um relacionado ao tema específico deste mapeamento:

vii. Os resultados obtidos foram discutem de maneira clara e objetiva a influência do uso de métodos ágeis no processo de manutenção considerando o contexto da pesquisa.

Cada autor avaliou os estudos selecionados de maneira independente e possíveis discordâncias foram discutidas a fim de que um consenso fosse alcançado. Usando uma escala de três pontos que indicavam o grau de aderência aos critérios de qualidade. A escala utilizada buscava responder se o estudo atendia a cada critério de qualidade e cada resposta recebeu um valor numérico: “Sim” (1 ponto), “Parcialmente” (0,5 ponto) e “Não” (0 ponto).

Os critérios de qualidade (i) e (iii) foram utilizados para excluir quaisquer estudos não-empíricos e/ou com contexto não descrito. Com base nisso, 1 estudo foi excluído por não ter atendido a ambos os critérios. Além disso, definimos um limiar mínimo de 3 pontos, de forma que qualquer estudo com pontuação inferior a tal, fosse desconsiderado. Desta forma, 1 estudo, com pontuação final 2,5, foi excluído.

Por fim, tivemos um conjunto final de 5 estudos alinhados com o objetivo deste trabalho para extração e análise de dados.

Extração de dados e síntese

Para conduzir a extração de dados, lançamos mão de um formulário de extração, de forma que fosse possível coletar informações detalhadas de como cada estudo abordava as

questões de pesquisa deste mapeamento sistemático. Cada autor executou a extração de maneira independente e potenciais discordâncias foram resolvidas em comum acordo nos marcos semanais de acompanhamento.

Para suportar esta etapa, foram utilizadas planilhas no Google Sheets, além da ferramenta NVivo. Após finalizar a extração nas planilhas, os trabalhos foram importados para o NVivo a fim de que o processo de codificação de conceitos, relações, resultados, amostras e metodologias utilizadas fosse realizado de maneira mais assistida. Os dados extraídos de cada artigo foram comparados por meio de tabelas de conclusões de cada um dos estudos e seu posicionamento frente ao uso de metodologias ágeis junto ao processo de manutenção de software.

O formulário de extração de dados utilizado tinha os seguintes tópicos:

- ID;
- Título;
- Ano de publicação;
- Autores;
- Tipo de estudo;
- Objetivos
- Método de coleta de dados;
- Segmento da indústria.
- Tamanho da indústria;
- Tamanho da amostra de participantes;
- Duração da pesquisa.
- Definição de manutenção de software;
- Definição de métodos ágeis;
- Métodos ágeis utilizados;
- Melhorias resultantes do uso de métodos ágeis;
- Desafios identificados devido ao uso de métodos ágeis.

As informações foram sintetizadas levando em os principais temas abordados pelos estudos selecionados. A partir disso, foram feitas classificações descritivas do conteúdo dos artigos selecionados com respeito às intervenções utilizadas, contexto das pesquisas, tamanho das amostras e resultados reportados pelos autores. Os resultados da síntese são apresentados

na próxima seção com o intuito de expor as evidências que levam às respostas das questões de pesquisa deste trabalho.