Metodologias ágeis: Utilizando *scrum* e *extreme programming* para o desenvolvimento de *software* nas empresas

Miguel Ângelo Pinheiro Fernandes
Instituto Federal do Goiás
Campus Luziânia
Bacharelado em Sistema de Informação
Email: miguel.f@estudantes.ifg.edu.br

Maria Laura Roriz
Instituto Federal do Goiás
Campus Luziânia
Bacharelado em Sistema de Informação
Email: mlauraifg@gmail.com

Marco Filipe
Instituto Federal do Goiás
Campus Luziânia
Bacharelado em Sistema de Informação
Email: marcomabony@hotmail.co

Resumo

A utilização de *softwares* nas empresas vem da necessidade da utilização dessas aplicações que são fundamentais para a viabilidade das operações e rotinas das empresas. Com a evolução da área e com o aumento da demanda é natural que práticas consideradas tradicionais evoluam e deem espaço para outras consideradas modernas. A importância desse assunto fica evidente quando se observa a existência de uma área de pesquisa somente para ela a engenharia de *software* voltada à especificação, desenvolvimento, manutenção e criação de *software*, com a aplicação de tecnologias e práticas de gerência de projetos e outras disciplinas, visando organização, produtividade e qualidade. Por fim, é notório que a utilização de metodologias como *Scrum*, *Extreme Programming* (XP), vieram para revolucionar a forma de fazer *software*.

Index Terms— Metodologias ágeis, Metodologias tradicionais, *Scrum*, *Extreme Programming* (XP).

1. Introdução

O desenvolvimento ágil foi concebido para produzir, rapidamente, *softwares* úteis. O *software* não é desenvolvido como uma única unidade, mas como uma série de incrementos e cada incremento inclui uma nova funcionalidade do sistema. Os processos de especificação, projeto e implementação são intercalados. Eles acontecem em paralelo. Não há especificação detalhada do sistema e a documentação do projeto é minimizada. O foco na metodologia ágil é finalizar o produto, então não há muita documentação (SOMMERVILLE, 2011).

Entre essas metodologias ágeis duas se destacam, de acordo com Schwaber e Sutherland (2011), o Scrum foi criado para auxiliar no desenvolvimento de projetos de *software* na década de 90 mas só ficou popular nos anos 2000, atualmente é um método ágil muito utilizado em ambiente empresarial na organização de grandes projetos. Outros autores também conceituam o Scrum, Cho (2010), o relaciona a um processo ágil de desenvolvimento de *software* utilizado para gerenciar, controlar *softwares* complexos e desenvolver produtos.

Segundo esse raciocínio Savoine (2009), apresenta a metodologia a XP como importante característica sua habilidade adaptativa, sendo principalmente utilizada em ambientes nos quais os requisitos funcionais do sistema não são totalmente claros, ou não são conhecidos pela equipe de desenvolvimento. Mas também quando eles são mutáveis ao longo do ciclo do projeto.

Os requisitos, por completo, são apresentados como cenários (chamados histórias do usuário), que são implementados como uma série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes da escrita do código. Todos os testes devem ser realizados com sucesso quando um novo código é integrado ao sistema (SOMMERVILLE. 2009).

Sendo assim, este artigo será dividido em duas etapas: na primeira buscará identificar os pontos positivos e negativos das metodologias ágeis e das metodologias tradicionais para o desenvolvimento de *software*. E posteriormente um estudo mais completo sobre as metodologias *Scrum*

e *Extreme Programming* (XP), buscando identificar se existe ou não um ganho de produtividade nas empresas que utilizam essas duas metodologias. A pesquisa utiliza a metodologia de revisão sistemática, sendo utilizado uma grande análise bibliográfica para a resolução dos questionamentos acima citados.

2. Engenharia de *Software*

É importante ressaltar que na conceituação sobre a engenharia de *software* dois autores se destacam, sendo Pressman (2016), definindo o processo de *software* como uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um *software* de alta qualidade. Não se diferenciando de Sommerville (2011), que conceitua como: um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*. Os Processos de *Software* são complexos e dependem de vários fatores para Pressman (2016), essas são as camadas da Engenharia de *Software*:

Imagem 1: Camadas da Engenharia de *Software*

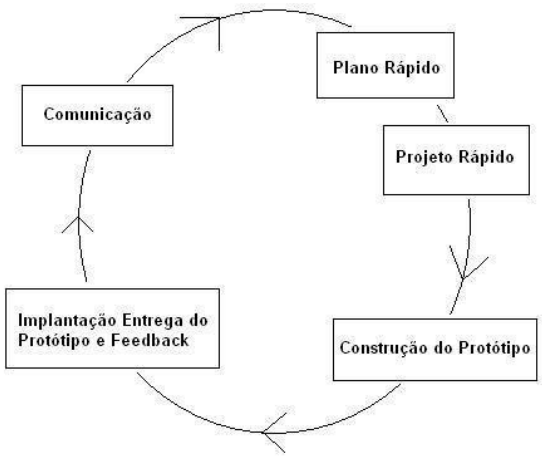
Fonte: Elaboração própria retirada do livro Pressman (2016).

Sendo assim, um modelo de processo fornece um guia específico para o trabalho de engenharia de *software*. Ele define o fluxo de todas as atividades, ações e tarefas, o grau de interação, os artefatos e a organização do trabalho a ser feito Pressman (2016). Os modelos de processos são prescritivos, ou seja, pensa-se no todo antes de começar o desenvolvimento. Os modelos de processo não prescritivos são denominados "ágeis", pois são dotados de uma maior facilidade para adaptação à mudança (PONTES, 2008).

Segundo Dos Santos (2004), uma das características das metodologias tradicionais está no fato que para que haja um melhor aproveitamento em sua utilização deve ser adotada em ambientes estáveis e com requisitos previsíveis. Portanto, o foco é nos processos e no desenvolvimento e uma caracterização da metodologia tradicional a utilização do modelo Clássico ou Sequencial que será utilizado como exemplo já que são esses dois modelos mais conhecidos e utilizados pelas empresas. Esse processo em que ocorre a construção de um *software* é baseado em várias etapas, porém não existe uma receita de bolo que dará certo para todas as empresas (DOS SANTOS, 2004).

Os modelos mais comuns de metodologias tradicionais são: a prototipação, o modelo Espiral e o modelo em cascata. A prototipação é um protótipo é uma versão inicial de um sistema de *software*, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções. O desenvolvimento rápido e interativo com custos controlados, sendo possível sua utilização como um modelo de processo isolado (PONTES, 2008).

Imagem 2: Etapas da prototipagem

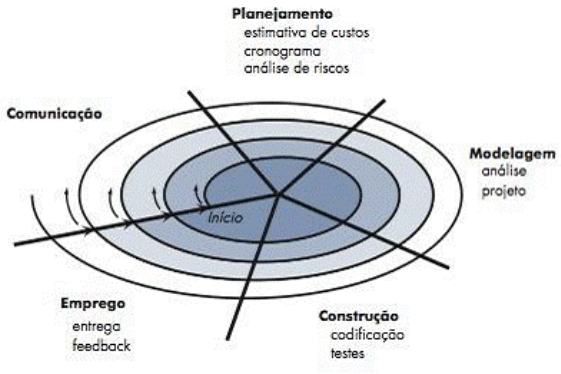


Fonte: Elaboração própria retirada do site (researchgate, 2008).

Como modelo de processo, a prototipação prega que o protótipo (modelo) deve ser descartado. Contudo, na prática, muitos acabam aproveitando esse código. O problema é que isso gera a necessidade de se incrementar esse *software*, o que pode acarretar um grande retrabalho. Hoje em dia a prototipação é mais utilizada como técnica dentro de outros modelos (SOMMERVILLE, 2011).

Segundo Sommerville (2011), o Modelo Espiral é um modelo de processo de *software* revolucionário que une a natureza iterativa da prototipação aos aspectos sistemáticos e controlados do modelo cascata. A principal diferença entre o modelo espiral e outros modelos de processo de *software* é seu reconhecimento explícito do risco. As versões iniciais podem ser um modelo de papel ou protótipo, as últimas são cada vez mais completas do sistema, diferentemente de outros modelos de processo, que terminam quando o *software* é entregue, o modelo espiral pode ser adaptado para ser aplicado ao longo da vida do *software*.

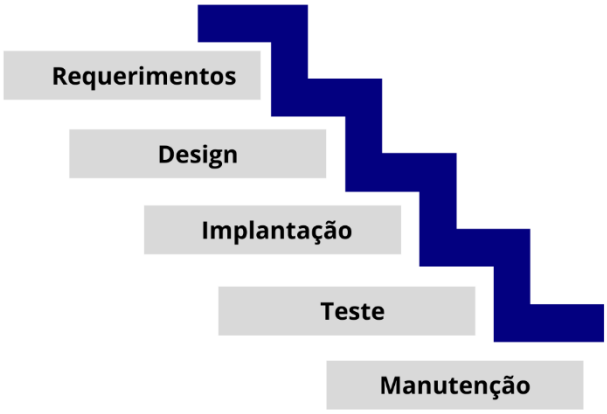
Imagem 3: Etapas do Modelo Espiral



Fonte: Elaboração própria retirada do livro, (SOMMERVILLE, 2011).

O modelo em cascata apresenta um fluxo linear, O modelo cascata — também conhecido como processo *Waterfall* — é uma metodologia de desenvolvimento de *software* surgida na década de 1970, criada por Winston Walker Royce. Sua principal característica é a divisão das tarefas em etapas predeterminadas, que são executadas de forma sequencial. Isso quer dizer que é preciso finalizar todas as tarefas de uma etapa para que seja possível passar para a seguinte. Ao cumprir todas as etapas, o resultado será um produto de *software* funcional, pronto para ser entregue ao cliente (DOS SANTOS, 2004).

Imagem 3: modelo em cascata



Fonte: Elaboração própria retirada do blog (betrybe, 2020).

Já que quando falamos de metodologia ágil é importante destacar apesar de temos esse conceito introdutório cada empresa irá escolher ou elaborar uma técnica que se adaptar melhor às suas rotinas e necessidades, porque ágil é muito mais que uma metodologia, ágil é um conjunto de valores e princípios a serem seguidos para evitar que surjam problemas típicos de desenvolvimento de *software* (DOS SANTOS, 2004).

A metodologia ágil surgiu da insatisfação com o modelo tradicional que era muito burocrático e se preocupava mais com a

documentação do que com a entrega do *software* e a satisfação do cliente. É importante destacar que a metodologia ágil tem mais efetividade quando implementada em pequenas empresas (PONTES, 2008).

Modelos tradicionais ou dirigidos a planos (classificação feita por Sommerville, onde há um plano e ele é seguido. É necessário conhecer todo o contexto do *software* para depois desenvolvê-lo) são interessantes, por exemplo, no caso de sistemas críticos de controle de segurança, em que uma análise completa dos sistemas é essencial. Mas no mundo atual em um ambiente de negócio caracterizado por mudanças rápidas, seria recomendado usar essas abordagens tradicionais? As metodologias possuem contextos e em alguns contextos, em que é preciso conhecer o todo, não será utilizada a metodologia ágil (DOS SANTOS, 2004).

2.1 Manifesto ágil

Com a evolução da engenharia de *software* e os constantes fracassos dos projetos de desenvolvimento utilizando abordagens tradicionais, em 2001 diversos profissionais compilaram as melhores maneiras de desenvolver *softwares*. Essa foi a origem do manifesto ágil. Há estudos que mostram que as abordagens tradicionais de desenvolver *softwares* não são simples. O maior percentual de desenvolvimento de *softwares* é considerado fracassado, pois é comum alguns problemas, como estouro de orçamento, não atendimento de requisitos de usuários, prazo não cumprido, resultando nos seguintes princípios (DA SILVA, 2014).

Tabela 01: Princípios do Manifesto Ágil

Princípios do Manifesto Ágil
1. Satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento.
3. Entregar frequentemente *software* funcionando.
4. Pessoas de negócio e desenvolvedores devem trabalhar em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados
6. O método mais eficiente e eficaz de transmitir informações é através de conversa face a face.
7. *Software* funcionando é a medida primária de progresso.
8. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante.
9. Contínua atenção à excelência técnica e bom design
10. A simplicidade é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina seu comportamento de acordo.

Fonte: Elaboração própria retirada da SILVA (2014).

3. *Scrum*

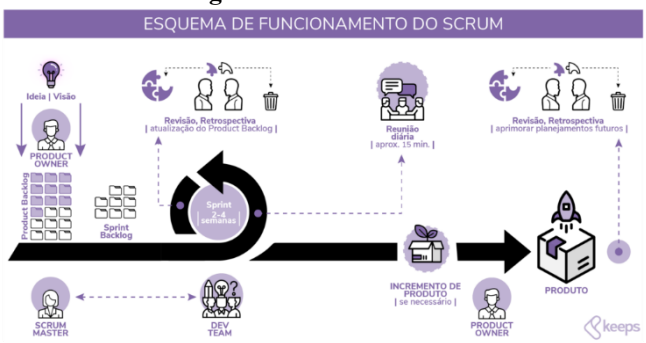
Sendo um aprimorado método iterativo e incremental para entregar *softwares* orientados a objetos, gerenciando projetos leves e ágeis, baseado em pequenas equipes capacitadas e auto-organizadas, com visibilidade e adaptação eficiente. Somado a isso muitas analogias podem ser feitas sobre a utilização do Scrum:

> *O método de desenvolvimento de software Scrum é um processo ágil que pode ser usado para gerenciar e controlar software complexo e desenvolvimento de produtos usando iterativo e práticas incrementais. Três estratégias do rugby, incluindo uma abordagem de equipe holística, constante interação entre os membros da equipe, e membros da equipe principal imutável, são adotados em Processos de gerenciamento e controle do Scrum (CHO, pág. 24, 2010).*

Como pode ser observado no exemplo acima um dos pilares do *Scrum* dentro das rotinas da empresa e o trabalho em equipe e apesar de ser

criado como modelo de desenvolvimento de *software*, com estrutura simples e um conjunto pequeno de regras, o *Scrum* pode ser utilizado em diferentes tipos de organizações públicas ou privadas, instituições de ensino, saúde, entre outros, contudo sempre adaptando as necessidades (SOUZA, 2021).

Imagem 4: Funcionamento do *Scrum*



Fonte: Elaboração própria retirada do site (keeps, 2022).

3.1 *Framework* e o *Time Scrum*

Segundo Schwaber e Sutherland (2011), o *Framework Scrum* foi definido no Guia do *Scrum* e apresentado ao mundo pela primeira vez em 1995 como uma forma melhor de colaboração em equipe para resolver problemas complexos, associando papéis, eventos, artefatos e regras há um time *Scrum*.

O *Scrum* é baseado em um pequeno time altamente flexível e adaptativo, composto por três papéis principais: o *Product Owner*, o *Scrum Master* e o Time de Desenvolvimento. O *Product Owner* é responsável por maximizar o retorno do investimento (ROI) do produto, gerenciando o *backlog* do produto e sempre atento às necessidades do cliente e do mercado. O *Scrum Master* é responsável por garantir que o *Scrum* seja compreendido e seguido por todos envolvidos no processo. O Time de Desenvolvimento é formado por profissionais que realizam o trabalho necessário para entregar um incremento do produto ao final de cada *Sprint* (SOUZA, 2021).

3.2 *Sprint*

Segundo Pereira, Torreão e Marçal (2007), o ciclo do *Scrum* tem o seu progresso baseado em uma série de iterações bem definidas, cada uma com duração de 2 a 4 semanas, chamadas *Sprints*. Antes de cada *Sprint*, realiza-se uma Reunião de planejamento (*Sprint Planning Meeting*) onde o time (equipe) de desenvolvedores tem contato com o cliente (*Product Owner*) para priorizar o trabalho que precisa ser feito, selecionar e estimar as tarefas que o time pode realizar dentro da *Sprint*.

Imagem 5: Detalhamento da fase *Sprint*

Fonte: Elaboração própria retirada do site (laboneconsultoria, 2020).

Ainda segundo os autores Pereira, Torreão e Marçal (2007), a próxima fase é a Execução da *Sprint*. Durante a execução da *Sprint*, o time controla o andamento do desenvolvimento realizando Reuniões Diárias Rápidas (*Daily Meeting*), não mais que 15 minutos de duração, e observando o seu progresso usando um gráfico chamado *Sprint Burndown*. Ao final de cada *Sprint*, é feita uma revisão no produto entregue para verificar se tudo realmente foi implementado.

Também ao final de cada *Sprint*, é realizada a Retrospectiva da *Sprint*, um evento em que o time *Scrum* se reúne para fazer uma autoavaliação e reflexão sobre aspectos positivos e negativos que ocorreram durante a *Sprint*. Neste momento, é elaborado um plano de melhorias para a próxima iteração. A retrospectiva é o principal evento onde o *Scrum* aplica o conceito de melhoria contínua. É papel do *Scrum Master* garantir um ambiente positivo para que todos se sintam à vontade para expressar suas opiniões. O tempo alocado para este evento é limitado a 3 horas para *Sprints* de um mês, sendo proporcionalmente menor para *Sprints* mais curtas (SOUZA, 2021).

3.3 Artefatos

Conforme Souza (2021), os artefatos são elementos que representam o trabalho a ser executado ou o resultado produzido. O *Scrum* possui três artefatos principais: o *Backlog* do Produto, o *Backlog* da *Sprint* e o Incremento (ou Trabalho Feito). O *Backlog* do Produto é uma lista ordenada com todas as características, funções, requisitos e melhorias que o produto deve ter. Essa lista é dinâmica e pode ser adaptada às demandas do projeto. O *Backlog* da *Sprint* é uma sub-lista do *Backlog* do Produto, contendo as tarefas que serão trabalhadas na *Sprint* atual. O Incremento é o resultado produzido no final de cada *Sprint*, representando o progresso realizado em relação ao produto final.

O *Backlog* da *Sprint* é um conjunto de itens do *Backlog* do Produto que serão trabalhados durante a *Sprint* atual. Esses itens são selecionados e refinados durante o planejamento da *Sprint* para que o time de desenvolvimento possa compreender o trabalho futuro. No entanto, ainda podem ser descobertas novas necessidades durante a *Sprint*, que são adicionadas ao *Backlog* da *Sprint*. Assim como o *Backlog* do Produto, o *Backlog* da *Sprint* é dinâmico e permite que o time lide de forma flexível com as mudanças.

O Incremento ou Trabalho Feito é o resultado da soma de todos os itens do *Backlog* do Produto que foram completados durante a *Sprint* atual e que foram devidamente integrados aos incrementos das *Sprints* anteriores. O Incremento é inspecionado durante a Revisão da *Sprint* e é um passo na direção do objetivo final do projeto. Segundo o *Scrum*, o Incremento precisa ser funcional e potencialmente liberável, ou seja, os usuários devem ser capazes de utilizá-lo de forma completa caso seja lançado no mercado (SOUZA, 2021).

4. Extreme programming

A Ex*treme Programming* (XP) é provavelmente o mais conhecido e mais amplamente usado dos métodos ágeis (SOMMERVILLE. 2009).

Foi criada em 1998 por Kent Beck, que a define como “um método ágil para equipes pequenas e médias desenvolvendo *software* com requisitos vagos e em constante mudança”. Beck explica que o termo *extreme* (extremo) é aplicado a um conjunto de práticas de desenvolvimento já existentes e reconhecidas como “boas práticas” no desenvolvimento de *software*, que são reunidas pela XP e elevadas ao extremo, ao limite (COSTA, 2016).

É uma metodologia que encoraja a equipe a enfrentar as mudanças de forma natural, sendo necessário se ajustar os contratempos que acontecem no projeto, contornando assim as

adversidades da forma mais simples possível, buscando produzir um sistema de qualidade (SAVOINE, 2009).

A partir desta metodologia, busca-se produzir um sistema de qualidade por meio de um conjunto de valores, princípios e práticas que se diferem das medidas tradicionais. Os valores em que ela se sustenta são: a comunicação, que deve ser simples, porém eficiente; a simplicidade, tanto no design quanto no algoritmo e tecnologias utilizadas; o *feedback* em relação à qualidade do código e ao andamento do projeto e por último a coragem para aplicar mudanças que venham a surgir durante o desenvolvimento do *software* (MAGALHÃES, 2009).

De acordo com Soares (2004), a XP enfatiza o uso de doze práticas que devem ser aplicadas totalmente, ou a maior parte possível. Podendo ser adaptadas à realidade das organizações, sendo utilizadas de forma harmônica e complementar para a entrega de um *software* de alta qualidade, que são:

1. Planejamento: que consiste em decidir o que vai ser feito e o que pode ser adiado.
2. Entregas frequentes: aumenta a probabilidade de o *software* final estar de acordo com o que foi pedido pelo cliente.
3. Metáfora: descrição simples com intuito de orientar no desenvolvimento.
4. Projeto simples: visa à construção simples para satisfazer requisitos atuais. Eventuais requisitos futuros devem ser adicionados assim que eles realmente existirem.
5. Testes: validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o *software* criando primeiramente os testes.
6. Refatoração: simplificar o código, sem alterar sua funcionalidade.
7. Programação em pares: a implementação do código é feita em dupla.
8. Propriedade coletiva: todos são responsáveis e aptos a realizarem melhorias. Sendo uma vantagem, pois caso um membro da equipe deixe o projeto, a equipe consegue continuar o projeto sem muita dificuldade já que todos conhecem todas as partes do *software*.
9. Integração contínua: integração diária do código produzido.
10. Trabalho semanal de 40h: evita sobrecarregar a equipe. Esta prática procura ratificar o foco nas pessoas e não em processos e planejamentos. Caso seja necessário, os planos devem ser alterados, em vez de sobrecarregar as pessoas.
11. Cliente presente: ele deve estar disponível para responder às dúvidas de requisitos, evitando atrasos e até mesmo construções erradas. O cliente deve ser mantido como parte integrante da equipe de desenvolvimento.
12. Código padrão: padronização na arquitetura do código para que o mesmo possa ser compartilhado entre todos.

Uma das características da XP é assegurar que o cliente receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento (PAINKA, 2013). De modo que, a satisfação do cliente é uma preocupação constante na XP, pois o objetivo principal é entregar um produto de qualidade em tempo hábil que satisfaça as exigências e expectativas tanto do cliente como da equipe de desenvolvimento (SAVOINE, 2009).

Com isso, a XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas (SOARES, 2004).

Por fim Soares (2004), cita alguns pontos positivos da metodologia XP, como as entregas constantes de partes operacionais do *software*, de

modo que o cliente não precisa esperar a finalização do projeto para ver o *software* funcionando. E com a integração e o teste contínuo, que possibilita que eventuais problemas sejam resolvidos constantemente. Sendo ideal para projetos em que os *stakeholders*[1] não sabem exatamente o que desejam e podem constantemente mudar de opinião durante o desenvolvimento do projeto.

5. Metodologia

A pesquisa utiliza a metodologia de revisão sistemática, elaborada por Siddaway, Wood e Hedges (2019), sendo utilizado uma análise bibliográfica para a resolução dos questionamentos acima citados. Levando em consideração os principais artigos e publicações científicas no que diz respeito às metodologias ágeis com foco no *Scrum* e XP trazendo questionamentos de diversos autores sobre o tema. Sendo assim, a pesquisa observou as seguintes etapas na construção do artigo começando com o estudo inicial do tema e fazendo o levantamento dos artigos que futuramente seriam utilizados na pesquisa.

- Na 1ª etapa, a pesquisa foi realizada empregando as palavras 'metodologias ágeis' associada com as palavras '*Scrum* + XP. Em princípio, foram selecionados 18 artigos, que contemplavam individualmente ou em conjunto esses temas.
- Na 2ª etapa, após a verificação do que estaria ou não dentro do campo de estudo selecionado, foram excluídos três artigos que em todo ou em parte tinham como objetivo áreas diferentes. Após essa seleção foram feitos fichamentos para facilitar a revisão dos artigos.

6. Considerações finais

O mercado de *software* está cada vez mais competitivo, exigindo constantes mudanças e novas formas de adaptação. Sendo necessário uma metodologia que possa se adequar nesse espaço dinâmico, proporcionando produtos de qualidade.

As metodologias ágeis apresentaram resultados positivos e mudanças para a engenharia de *software*, alcançando a exigência de mercado e excelência dos produtos desenvolvidos. Porém elas não são aplicáveis em todas as situações e requerem adaptação e treinamento específicos.

Como apresentado neste trabalho, as duas metodologias exemplificadas foram o *Scrum* e a XP que compartilham semelhanças como, equipes pequenas trabalhando em requisitos instáveis ou desconhecidos. Entretanto, elas apresentam também diferenças, como na maneira que o código deve ser integrado, a questão dos testes, entre outros. Afirma-se que enquanto a XP é voltada para as práticas de implementação, o *Scrum* foca no gerenciamento e planejamento. Diante disso, acredita-se que a melhor maneira de utilizar esses dois métodos é de forma complementar.

REFERÊNCIA

CHO, Juyun Joey. An exploratory study on issues and challenges of agile software development with scrum. All Graduate theses and dissertations, p. 599, 2010.

COSTA, Daiane Morandi da. Customização de processo de desenvolvimento de software baseado na modelagem ágil, p. 40-46, 2016.

DA SILVA, ERAYLSON Galdino; manifesto, ágil; de curso, trabalho de conclusão. Universidade de Pernambuco-upe campus Garanhuns licenciatura em computação. 2014.

DE OLIVEIRA, Rodrigo Alberto et al. Desafios no uso de metodologias ágeis de gestão de projetos em

[1]Stakeholders (partes interessadas) são as pessoas e as organizações que podem ser afetadas por um projeto ou empresa, de forma direta ou indireta, positiva ou negativamente.

órgãos públicos: um estudo de caso da Receita Estadual do Paraná. Gestão e Projetos: GeP, v. 11, n. 2, p. 12-36, 2020.

DOS SANTOS SOARES, Michel. Comparação entre metodologias Ágeis e tradicionais para o desenvolvimento de software. INFOCOMP Journal of Computer Science, v. 3, n. 2, p. 8-13, 2004.

MAGALHÃES, Cleyton Vanut Cordeiro de et al. XP e Scrum sob uma Abordagem Comparativa. IX Jornada de Ensino, Pesquisa e Extensão da UFRPE. Anais do JEPEX, 2009.

PAINKA, Marcelo Augusto Lima; DA COSTA MARCHI, Késsia Rita. Utilização das Metodologias Ágeis XP e Scrum para o Desenvolvimento Rápido de Aplicações. 2013.

PEREIRA, Paulo; TORREÃO, Paula; MARÇAL, Ana Sofia. Entendendo Scrum para gerenciar projetos de forma ágil. Mundo PM, v. 1, p. 3-11, 2007.

PONTES, Thiago Bessa; ARTHAUD, Daniel Dias Branco. Metodologias ágeis para o desenvolvimento de softwares. Ciência E Sustentabilidade, v. 4, n. 2, p. 173-213, 2018.

SAVOINE, Márcia et al. Análise de Gerenciamento de Projeto de Software Utilizando Metodologia Ágil XP e Scrum: Um Estudo de Caso Prático. XI Encontro de Estudantes de Informática do Tocantins, p. 93-102, 2009.

SCHWABER, Ken; SUTHERLAND, Jeff. The scrum guide. Scrum Alliance, v. 21, n. 19, p. 1, 2011.

SOARES, Michel dos Santos. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. Revista Eletrônica de Sistemas de Informação, v. 3, n. 1, 2004.

SOMMERVILLE, Ian. Engenharia de Software. São Paulo: Pearson Addison Wesley, p. 259-268, 2009.

SOUZA, Beatriz Lopes. Metodologias ágeis: análise e comparação do Scrum, Kanban e Lean aplicados ao desenvolvimento de software. 2021.

VACARI, Isaque; PRIKLADNICKI, Rafael. Metodologias ágeis na administração pública: uma revisão sistemática da literatura. In: Embrapa Informática Agropecuária-Artigo em anais de congresso (ALICE). In: WORKSHOP BRASILEIRO DE MÉTODOS ÁGEIS, 5., 2014, Florianópolis. Resumos... São José dos Campos: INPE, 2014., 2014.