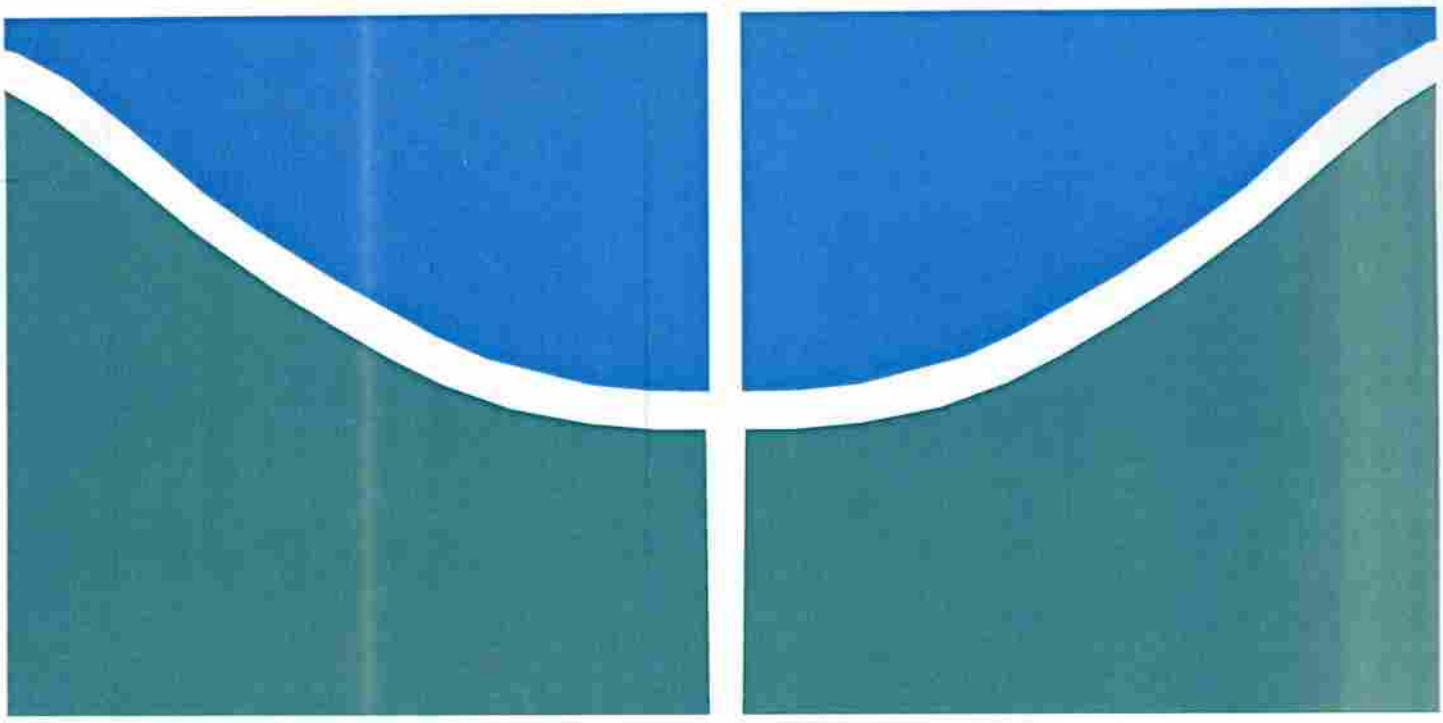

METOLOGOGIAS ÁGEIS: *EXTREME PROGRAMMING* E *SCRUM*

PAULO ADRIANO NUNES FERRAZ

MONOGRAFIA PARA PÓS-GRADUAÇÃO EM GESTÃO DE TI

FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METOLOGOGIAS ÁGEIS: *EXTREME PROGRAMMING* E *SCRUM*

PAULO ADRIANO NUNES FERRAZ

ORIENTADOR: Dra. EDNA DIAS CANEDO

MONOGRAFIA PARA PÓS-GRADUAÇÃO EM GESTÃO DE TI

PUBLICAÇÃO:

BRASÍLIA, DF: AGOSTO / 2017.

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METOLOGOGIAS ÁGEIS: *EXTREME PROGRAMMING* E *SCRUM*

PAULO ADRIANO NUNES FERRAZ

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE PÓS-GRADUAÇÃO EM GESTÃO DE TI.

APROVADO POR:

_____

EDNA DIAS CANEDO, ORIENTADORA
DOUTORA, FGA/UNB

_____

ELIANE CARNEIRO SOARES
MESTRE, SEEDF (EXAMINADOR EXTERNO)

_____

RAFAEL TIMOTEO DE SOUSA JUNIOR
DOUTOR, FGA/UNB (EXAMINADOR INTERNO)

BRASÍLIA, DF, 15 DE AGOSTO DE 2017.

FICHA CATALOGRÁFICA

Ferraz, Paulo Adriano Nunes.
Metodologias Ágeis: *Extreme Programming* e *Scrum* [Distrito Federal], 2017.
49p., 210 x 297mm (ENE/FT/UnB, Especialista, Engenharia Elétrica, 2017).
Monografia de especialização – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica.

1. Metodologias Ágeis
2. Manifesto Ágil
3. *Scrum*
4. *Extreme Programming*
5. Projetos de Software
I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Ferraz, Paulo Adriano Nunes. (2017). Metodologia Ágeis: *Extreme Programming* e *Scrum*. Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF.

CESSÃO DE DIREITOS

AUTOR: Paulo Adriano Nunes Ferraz
TITULO DA TESE: Metodologias Ágeis: *Extreme Programming* e *Scrum*.
GRAU / ANO: Pós-graduado / 2017

Paulo Adriano Nunes Ferraz
SCRN 714/715 Bloco G Entrada 7 Apartamento 202
Asa Norte
CEP: 70.761-670 - Brasília - DF
Tel. 55 – 61 – 983779366 / tecpauloadriano@gmail.com

AGRADECIMENTOS

Agradeço a Deus por todas as oportunidades que tem me concedido.

Agradeço a minha família pelo apoio e compreensão nos dias que foram necessários para dedicação.

Agradeço a equipe do LabRedes e aos professores pelo suporte no desenvolvimento e construção deste trabalho.

RESUMO

METOLOGOGIAS ÁGEIS: *EXTREME PROGRAMMING* E *SCRUM.*

Autor: Paulo Adriano Nunes Ferraz
Orientador: Professora Dra. Edna Dias Canedo
Programa de Pós-graduação em Gestão de TI
Brasília, 15 de agosto de 2017.

Os projetos de *software* estão em constante evolução e adequação para atender as necessidades do mercado. Nos projetos atuais de *software* é possível observar cada vez mais a flexibilidade e dinamismo nas definições, devido a muitos fatores internos e externos das organizações. Esses problemas têm dificultando ainda mais a conclusão e assertividade das entregas e finalização com sucesso dos projetos. As metodologias ágeis, em respostas a essas necessidades, conseguem absorver melhor essas mudanças revertendo em oportunidade de melhoria na qualidade de entrega. Neste trabalho, é apresentado uma revisão dos conceitos do manifesto ágil apresentando as metodologias ágeis *Scrum* e *Extreme Programming*. Através da revisão conceitual das metodologias, busca analisar e apresentar um comparativo entre o *Scrum* e *Extreme Programming*. Os resultados obtidos não visão uma comparação qualitativa das duas metodologias e provar qual seria melhor, pois apesar de implementarem os princípios do manifesto ágil, cada uma dessas metodologias pode ser aplicadas e utilizada em ambientes diferentes e resultando em ganhos diferentes. O comparativo apresenta os detalhes de como executar e aplicar a metodologias, para que seja possível identificar qual se adequaria a necessidade e problemas das empresas.

Palavras-chave: Projetos de Desenvolvimento de *Software*. Metodologias Ágeis. Manifesto Ágil. *Scrum*. *Extreme Programming*.

ABSTRACT

AGILE METHOLOGIES: EXTREME PROGRAMMING AND *SCRUM*
Author: Paulo Adriano Nunes Ferraz
Supervisor: Dra. Edna Dias Canedo
Post-Graduation Program in Electrical Engineering
Brasilia, 15 August 2017

Software projects are constantly evolving and adapting to meet the needs of the market. In today's software projects, it is possible to observe more and more the flexibility and dynamism in the definitions, due to many internal and external factors of the organizations. These problems have made it even more difficult to complete and assert deliveries and successfully complete projects. Agile methodologies, in response to these needs, are able to absorb these changes better, reversing opportunities for improvement in delivery quality. In this work, a review of the agile manifesto concepts is presented, presenting agile Scrum and Extreme Programming methodologies. Through the conceptual review of methodologies, it seeks to analyze and present a comparison between Scrum and Extreme Programming. The results obtained do not see a qualitative comparison of the two methodologies and prove which would be better, because although they implement the principles of the agile manifesto, each of these methodologies can be applied and used in different environments and resulting in different gains. The comparative presents the details of how to execute and apply methodologies, so that it is possible to identify which would suit the need and problems of the companies.

Keywords: *Software* Development Projects. Agile Methodologies. Agile Manifesto. *Scrum*. *Extreme Programming*.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

ABES	Associação Brasileira das Empresas de *Software*
PMI	*Project Management Institute*
PMI-ACP	*Project Management Institute – Agile Certified Practitioner*
TI	Tecnologia da Informação
XP	*Extreme Programming*

1 - INTRODUÇÃO

O gerenciamento de projetos, assim como as tecnologias, vem passando por crescentes adaptações e atualizações. No cenário gestão de desenvolvimento de *software*, cada vez mais o mercado consumidor tem suprimido as empresas com exigências de sistemas com qualidade, sem deixar de buscar pelo melhor custo benefício financeiro. Essas exigências têm causado mudanças significativas no modo de gestão em um projeto de desenvolvimento de *software*.

Em busca da aceleração do desenvolvimento econômico as mudanças, sejam elas em negócio ou estratégia, estão gerando impacto no modelo de desenvolvimento de *software*. O produto constantemente está sofrendo mudanças, durante quase todo o ciclo de desenvolvimento para atender as diversas necessidades dos clientes (Rising & Janoff, 2000). Essas alterações nem sempre implicam em novos acordos de extensão dos prazos e ajustes nos orçamentos, consequentemente a equipe acaba se desgastando mais para cumprir uma meta anteriormente estabelecida. A assertividade na estimativa de prazo e orçamento para a construção de soluções, é algo fundamental para satisfação do cliente e sucesso em qualquer projeto.

Quando ocorrem essas alterações no produto, independentemente do nível de complexidade que sofrerá o produto, gera uma instabilidade na gestão do projeto que possivelmente será discutido em inúmeras reuniões. Abaixo do nível gerencial do projeto, as equipes envolvidas na construção do produto, obrigatoriamente precisam-se readequar e iniciar novamente os ciclos da construção, quando utilizam de métodos tradicionais para gestão.

Quando conhecemos o produto e compreendemos do ambiente em que ele é inserido, podemos criar os requisitos com precisão. Com os requisitos bem definidos, é possível alcançar um nível de maturidade para construção da solução em menor tempo com as equipes da construção da tecnologia. Segundo Pressman (1995), para que um projeto de *software* seja bem-sucedido, é necessário que alguns parâmetros sejam corretamente analisados, como por exemplo, o escopo do *software*, os riscos envolvidos, os recursos necessários, as tarefas a serem realizadas, os indicadores a serem acompanhados, os esforços e custos aplicados e a sistemática a ser seguida.

O primeiro modelo de gerenciamento de *software*, surgiu na década de 70, foi o modelo em Cascata. Mesmo sendo um modelo antigo, o processo Cascata ainda é utilizado atualmente. Os modelos para a produção de *software* produzidos utilizando métodos sequenciais são denominados de métodos tradicionais. A construção de sistemas através desses métodos, são feitos de modo linear seguido de uma ordem sequencial de desenvolvimento de fases e dependentes das fases anteriores (Teles, 2006).

Para obter maior sucesso na utilização das metodologias tradicionais, essas metodologias devem ser aplicadas em situações em que os requisitos do *software* são estáveis e possivelmente não tenha alterações. Para os atuais cenários de TI (Tecnologia da Informação, estas situações são difíceis de serem atingidas, uma vez que os requisitos para o desenvolvimento de um *software* estão cada vez mais estáveis e dificilmente são passados de forma detalhada (Soares, 2004). Isso é consequências de vários fatores como: mercado dinâmico, possíveis alterações nas leis, novas solicitações feitas pelos clientes dentre muitas outras que aumentam a instabilidade dos requisitos.

Em 2001, houve uma reunião realizada por 17 especialistas na criação de *software*. Desta reunião emergiu o Manifesto Ágil e através dele muitas mudanças inéditas ocorreram para a área de engenharia de *software* (Lacerda et al., 2004). A transformação iniciada pelo Manifesto Ágil foi um marco que deve ser destacado, pois vários métodos, ferramentas, técnicas e melhores práticas foram criadas e definidas desde então. A criação desses métodos foi motivada pela necessidade de superar os problemas dos métodos tradicionais (Cavalcanti, 2006), considerado pelos precursores dos métodos ágeis, lento e contraditório à maneira eficiente de trabalho dos engenheiros de *software* (Kniberg, 2007). As metodologias ágeis para desenvolvimento de *software* foram fortemente influenciadas pelo modelo Toyota da indústria japonesa e pelos princípios da manufatura enxuta implementados.

De acordo com *The Standish Group* (2015), através da base de dados de informações de dez mil projetos de *software* entre o ano de 2011 até 2015, em todos os tamanhos de projetos, as abordagens ágeis resultam em projetos mais bem-sucedidos e menor porcentagem de falhas, conforme a Figura 1.1.

Projetos	Metodologias	Sucesso	Falha
Todos os Projetos	Ágil	39%	9%
	Cascata	11%	29%
Projetos Grandes	Ágil	18%	23%
	Cascata	3%	42%
Projetos Médios	Ágil	27%	11%
	Cascata	7%	25%
Projetos Pequeno	Ágil	58%	4%
	Cascata	44%	11%

Figura 1.1 Comparação de resultados entre projetos ágeis e tradicionais, *The Standish Group* (2015).

A utilização de metodologias ágeis está crescendo, devido aos seus inúmeros benefícios agregados na gestão de projetos de *software* (Sethi et al., 2011). Cada método ágil existente hoje carrega consigo os valores e princípios arraigados no manifesto ágil. Dentre as metodologias ágeis disponíveis, duas são mais pesquisadas: O *Scrum*, voltado ao gerenciamento e o *Extreme Programming* (XP), voltado ao desenvolvimento (Abrahamsson, 2008). No Brasil, o *Scrum* é a metodologia ágil mais utilizada, sendo seguida por sua combinação com o XP (Melo et al., 2013).

Mediante as necessidades do mercado e as possíveis metodologias ágeis disponíveis, o modelo proposto investiga os principais *frameworks* para auxiliar a gestão de desenvolvimento de *software*.

1.1 – MOTIVAÇÃO

O mercado de *software* no Brasil está em constante crescimento e investimento. Os dados da Associação Brasileira das Empresas de *Software* (ABES, 2016) apresenta que em 2015, o crescimento dos investimentos em TI no Brasil mostrou um aumento de 9,2% em relação a 2014. Se comparado às demais economias mundiais, o país ainda se destacou, considerando que a média mundial de crescimento dos investimentos em TI foi de 5,6%. Com esse resultado, o Brasil permanece na lista dos países que apresentaram maior crescimento setorial, mantendo a 7ª posição no *ranking* mundial de investimentos em TI.

Para que o mercado de *software* continue crescendo, é preciso que as empresas consigam alinhar pontos fundamentas como: redução de custos, prazos, qualidade. Em busca destes objetivos, ao longo dos anos, as metodologias de gestão de desenvolvimento foram aprimoradas e como resultado temos as metodologias ágeis.

A criação deste trabalho é justificada plenamente para mostrar os benefícios e possíveis prejuízos na utilização das metodologias ágeis. Ele visa refinar as principais metodologias utilizadas e propor a utilização das mesmas dependendo do cenário aplicado.

1.2 - OBJETIVOS DO TRABALHO

Atualmente existem diversas metodologias e ferramentas que auxiliam a gestão do desenvolvimento de *software*. Visando apresentar os benefícios da utilização de metodologias ágeis e suas principais metodologias presentes no mercado. Os objetivos deste trabalho podem ser resumidos em:

a. Apresentação das metodologias ágeis e seus conceitos, levando em consideração aspectos que são vantajosos e desvantajosos para utilização do mesmo;
b. Apresentar os modelos principais e sua melhor utilização para os cenários;
c. Realizar o comparativo e melhor utilização seguindo a referências bibliográficas.

1.3 - METODOLOGIA DE PESQUISA

Para elaboração desde trabalho foi utilizado o método de pesquisa bibliográfica, visto que busca analisar e descrever as metodologias ágeis e verificar a suas relações. Este estudo se baseou em fontes de revisão bibliográfica de trabalhos já publicados.

Quanto à natureza, a pesquisa é classificada como básica, pois objetiva gerar conhecimentos novos para avanço da ciência, sem aplicação prática imediata. Do ponto de vista da forma da abordagem do problema, a pesquisa é qualitativa, pois não foi apresentado os resultados de forma estatística. Quanto aos objetivos da pesquisa, é uma pesquisa exploratória, pois investiga e analisa as práticas das metodologias ágeis XP e *Scrum*. Para tanto, o procedimento técnico utilizado foi a pesquisa bibliográfica. Para a elaboração deste trabalho foram realizadas as seguintes etapas:

Etapa 1. Foi realizado uma revisão bibliográfica das metodologias ágeis e suas principais utilizações, sendo possível o desenvolvimento de conceitos e planejamento da pesquisa.

Etapa 2. Análise dos processos e cenários envolvidos nas metodologias *SCRUM* e XP.

Etapa 3. Criação do modelo comparativo na utilização das metodologias ágeis.

1.4 - CONTRIBUIÇÕES DO TRABALHO

Buscam-se com este trabalho as seguintes contribuições:

- Apresentação dos princípios e valores do manifesto ágil, considerando aspectos fundamentais para análise das metodologias ágeis;
- Apresentação dos trabalhos recentes empregando o conceito e utilização das metodologias ágeis;
- Apresentação de uma proposta de utilização das metodologias ágeis para auxiliar a gestão do desenvolvimento de *software*.

1.5 - ORGANIZAÇÃO DO TRABALHO

Para um melhor entendimento deste trabalho, a sua organização é descrita a seguir.

O presente capítulo contém uma introdução ao trabalho realizado, definindo seu objetivo, metodologia e suas contribuições.

O capítulo 2 oferece uma revisão do manifesto ágil, demonstrando seus valores e princípios.

O capítulo 3 apresenta uma revisão conceitual do *Scrum*, mostrando sua estrutura e execução.

O capítulo 4 apresenta uma revisão conceitual do *Extreme Programming*, mostrando seus valores e práticas a serem utilizadas. O resultado comparativo é apresentado neste capítulo.

O capítulo 5 traz a conclusão do trabalho e as possibilidades de trabalhos futuros.

2 – MANIFESTO AGIL

Este capítulo tem como foco a revisão dos conceitos sobre o manifesto ágil. Na seção 2.1 é abordado a origem e surgimento do manifesto ágil. Na seção 2.2 é apresentado os valores que regem as metodologias ágeis. Na seção 2.3 é apresentado os princípios que regem as metodologias ágeis. Na seção 2.4 é feita uma abordagem resumida das metodologias disponíveis, demonstrando sua utilização.

2.1 – ORIGEM E CONCEITOS

Na primavera de 2000, ocorreu uma reunião com alguns defensores da ideia do *Extreme Programming* organizado por Kent Beck (Engenheiro de *software* criador da metodologia ágil XP e um dos 17 signatários do manifesto ágil) no Rogui River Lodge, no Oregon. Os participantes iniciaram suas manifestações de apoio para uma nova variedade inicialmente chamada de métodos leve (*Lightweight Methods*) (Fowler, 2000). Durante o ano de 2000, foram escritos vários artigos que referenciavam a categoria de processos "leves". Alguns desses artigos se referiam a metodologia leve, como XP, desenvolvimento de *software* adaptativo, Crystal e *Scrum*. Nas conversas, ninguém realmente gostou do apelido "*light*" ou "leves", mas pareceu ficar por enquanto (Highsmith, 2001).

Quanto ao nome inicial de metodologias leves, Alistair Cockburn (Cientista da computação e um dos 17 signatários do manifesto ágil) apresentou descontentamento total com a palavra "*Light*". Ele deixou a seguinte crítica na história do manifesto: "Não me importo que a metodologia seja chamada de peso leve, mas não tenho certeza de que eu queira ser referido como um "peso leve" atendendo a uma reunião de metodologistas leves. De alguma forma, parece ser um grupo de pessoas magras e leves que se esforçam para se lembrar de que dia é " (Highsmith, 2001).

Em setembro de 2000, Bob Martin começou a divulgar uma convocação para um próximo evento. Seria uma conferência pequena de dois dias entre o período de janeiro a fevereiro de 2001, com o objetivo de reunir os líderes de método leve para aproximar e debater sobre o método (Highsmith, 2001).

De 11 a 13 de fevereiro de 2001, nas montanhas de Wasatch, Utah, dezessete pessoas se reuniram para dialogar sobre a metodologia leve. Esse grupo de pessoas era composto por engenheiros e cientistas especialista nas práticas e teorias sobre realizar um

projeto de *software*, esse grupo foi nomeado de "Aliança Ágil". O que emergiu foi o manifesto ágil para desenvolvimento de *software*. O manifesto foi assinado por todos os participantes: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas. O manifesto está contido com seus valores e princípios (Highsmith, 2001).

Apesar desse grupo haver pessoas com experiências e teorias diferente, todos concordaram da necessidade de mudança nas metodologias da época. Estes profissionais de diferentes áreas de formação, com pontos de vista diferentes sobre os modelos e métodos de desenvolvimento de *software* em comum, publicaram um manifesto para encorajar melhores meios de desenvolvedor *software* (Ambler, 2004).

O movimento ágil não pode ser considerado como anti-metodologia, abandonando e descartando todas as técnicas e métodos. Segundo Jim Highsmith na história do manifesto ágil:

> "O movimento ágil não é anti-metodologia, na verdade, muitos de nós querem restaurar a credibilidade da palavra metodologia. Adotamos a modelagem, mas não para arquivar algum diagrama em um repositório corporativo empoeirado. Abraçamos documentação, mas não centenas de páginas de volumes nunca mantidos e raramente usados. Planejamos, mas reconhecemos os limites do planejamento em um ambiente turbulento".

O manifesto reformula as prioridades em alguns itens considerados primários nas metodologias tradicionais, como por exemplo, os processos e ferramentas, a documentação abrangente, a negociação de contratos e o cumprimento de um plano. Isso não quer dizer que o Manifesto condena a utilização destes itens, mas afirma que eles possuem uma importância secundária para o processo de desenvolvimento (Tavares, 2015).

As metodologias ágeis são métodos que vieram dos preceitos do Sistema Toyota de Produção, difundidos na época, que por sua vez vieram do que se conhece hoje como filosofia *Lean*.

As metodologias ágeis, na sua utilização, podem ser comparadas com o modelo de manufatura enxuta. O modelo de manufatura enxuta, foi criado por especialistas para melhoria na produção de carros, entretanto esse modelo pode ser aplicado e adaptado para qualquer outra área de negócio (Almeida, 2014). Da mesma forma, o manifesto ágil, foi

criado a partir de uma necessidade na área de desenvolvimento de *software*, porém pode ser amplamente utilizado e adaptado para outras áreas de negócio. Comparações do Sistema Toyota de Produção com metodologias ágeis já foram feitas e publicadas. Em 2006, Boris Gloger escreveu um artigo comparando o sistema da Toyota com *Scrum* (Gloger, 2006), assim como Moryen com o XP em 2005 (Moryen, 2005) e Van Cauwenberghe em 2005 que o comparou com o gerenciamento ágil (Van Cauwenberghe, 2005).

A importância das metodologias ágeis ganhou atenção do PMI (*Project Management Institute*). Segundo o estudo o uso das metodologias ágeis triplicou de dezembro de 2008 a maio de 2011 (PMI, 2014). Mediante este cenário, o instituto lançou em agosto de 2011 a certificação PMI-ACP (*Project Management Institute – Agile Certified Practioner*), que reconhece o conhecimento de princípios ágeis, práticas, ferramentas e técnicas através de metodologias ágeis (PMI, 2014) (Tavares, 2015).

2.2 – VALORES

Cada método ágil existente hoje carrega consigo os valores e princípios arraigados no manifesto ágil.

2.2.1 Indivíduos e interações mais que processos e ferramentas

O manifesto ágil, criado por especialistas na criação de *software*, entenderam que é preciso valorizar o aspecto humano. Quem gera produtos e serviços são os indivíduos, que possuem características únicas individualmente e em equipe, como talento e habilidade (Highsmith, 2004). Alistair Cockburn afirma que as pessoas na equipe são mais importantes do que seus papéis em diagramas de processos. Assim, embora descrições de processos possam ser necessárias para se começar o trabalho, as pessoas envolvidas nos processos não podem ser trocadas como peças (Cockburn,2007).

Os projetos de *software* possuem uma equipe, talvez até muito grande, com papeis e responsabilidades diferentes como: equipe de desenvolvedores, equipe de teste, equipe de qualidade, dentre outras. O manifesta não ignora a necessidade de processos e ferramentas, mas é importante que todas as equipes possam trabalhar juntas e interativas para obter maior sucesso. Esse princípio pretende aumentar o espírito de equipe nas relações internas da equipe (Abrahamsson et al., 2002).

2.2.2 *Software* em funcionamento mais que documentação abrangente

Para Alistair Cockburn, *software* em funcionamento é o único indicador do que a equipe de fato construiu (Cockburn, 2007). Em cada iteração com o cliente, deve entregar um produto executável, que vai se integrando gradualmente ao produto final. Assim o cliente acompanha a construção evolutiva e contínua do produto, tendo a transparência do trabalho realizado, implicando na geração mútua de confiança (Dantas, 2003) (Highsmith, 2004). Tal situação propicia uma redução do risco de insucesso do projeto.

A entrega de pequenas versões é uma das grandes diferenças que o desenvolvimento ágil traz (Mundim et al., 2002). Segundo Jim Highsmith (2001), o movimento ágil não tem por objetivo abolir a documentação, porem as metodologias tradicionais enfatizam a criação de uma vasta documentação que por sua vez não é utilizada na construção do *software*. Segundo Cockburn, a documentação pode ser muito útil para o desenvolvimento do projeto, mas deve-se produzir somente a documentação necessária e suficiente para a realização do trabalho (Cockburn, 2007). O tempo que era vinculado a criar as documentações, pode ser distribuído em desenvolvimento e ganho no prazo de entrega.

2.2.3 Colaboração com o cliente mais que negociação de contratos

O contrato é importante para definir as responsabilidades e direitos, mas a comunicação não pode ser restringida e nem substituída pelo contrato. O desenvolvimento ágil está focado em fornecer valor ao negócio imediatamente no início do projeto, reduzindo assim os riscos de não cumprimento do contrato. Segundo Jim Highsmith, no desenvolvimento de novos produtos como *software* em que há alta volatilidade, ambiguidade e incertezas, a relação cliente desenvolvedor deve ser colaborativa, ao invés de marcada por disputas de contrato antagônicas (Highsmith,2004).

Somente o cliente consegue compreender completamente a solução que espera receber, sendo que muitas vezes eles possuem dificuldades de passar os requisitos com maior definição. Quando há colaboração e iteração constante do cliente, é possível entender melhor suas necessidades e ajuda-lo a desenhar sua solução final.

2.2.4 Responder a mudanças mais que seguir um plano

As mudanças são necessárias para alcançar a satisfação do cliente e construção de um produto eficaz, visto que durante todo o ciclo de desenvolvimento os ambientes, processos, leis e até as tecnologias podem sofrer alterações. Essas mudanças devem ser controladas e organizadas.

Visto a necessidade de executar as mudanças, os projetos de *software* ágeis devem ser flexíveis para comportar as mudanças em qualquer ciclo do projeto. A adaptação do processo e a inspeção do produto são pilares fundamentais para as boas práticas ágeis. Segundo Cockburn, iterações curtas de desenvolvimento permitem que mudanças possam ser mais rapidamente inseridas no projeto, de forma que atendam às novas necessidades dos clientes (Cockburn, 2007).

2.3 – PRINCÍPIOS

- Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *software* de valor.
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
- Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
- Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
- O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
- *Software* funcional é a medida primária de progresso.
- Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
- Contínua atenção à excelência técnica e bom design, aumenta a agilidade.

- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
- As melhores arquiteturas, requisitos e designs emergem de times auto organizáveis.
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

2.4 – METODOLOGIAS ÁGEIS

Dentre as metodologias ágeis disponíveis, duas são mais pesquisadas: O *Scrum*, voltado ao gerenciamento e o XP, voltado ao desenvolvimento (Abrahamsson, 2008). No Brasil, o *Scrum* é a metodologia ágil mais utilizada, sendo seguida por sua combinação com o XP (MELO et al., 2013).

Segundo o relatório da *Versionone* (2016), o *Scrum* é a metodologias mais utilizada no mercado atualmente.

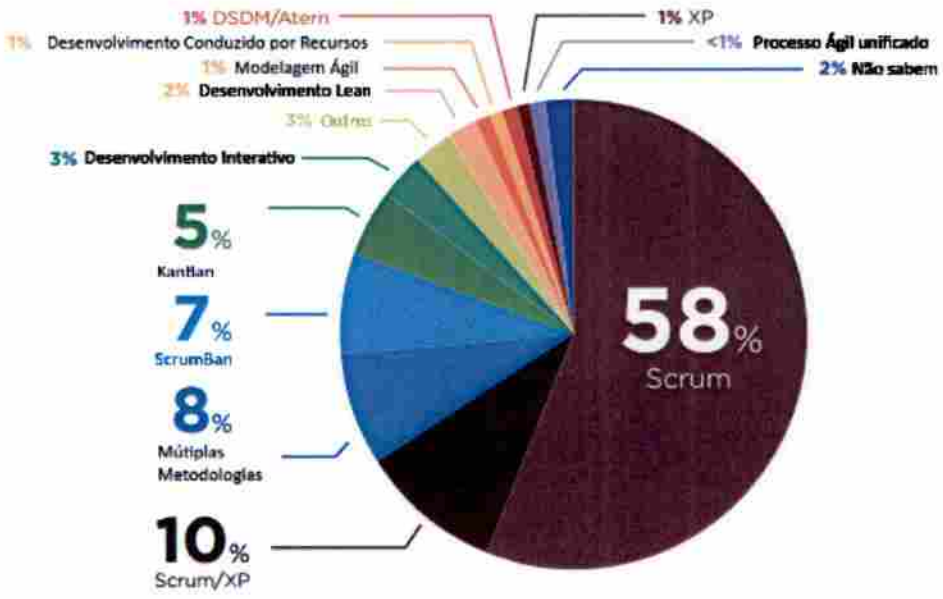

Figura 2.1 – Uso de Metodologias ágeis, *Versionone* (2016).

3 – *SCRUM*

Este capitulo irá apresentar a revisão bibliográfica da metodologia *Scrum*, sendo dividido em seções. A seção 3.1 apresenta a origem da metodologia ágil *Scrum*. A seção 3.2 apresenta algumas definições do *Scrum* como *framework*. A seção 3.3 apresenta os papeis e responsabilidades dos integrantes do time do *Scrum*. A seção 3.4 apresenta os eventos que ocorrem durante a execução do *Scrum*. A seção 3.5 apresenta os artefatos que são gerados pelo *Scrum*.

3.1 ORIGEM

O *Scrum* inicialmente foi utilizado pelos autores Takeuchi e Nonaka em sua publicação, o "*The New New Product Development Game*" (O novo jogo no Desenvolvimento de novos Produtos) em 1986, na comparação da formação *Scrum* no jogo de *rugby* com gerenciamento de projetos na fabricação de automóveis e produtos de consumo. Os autores fizeram analogia do *rugby* para descrever de maneira perfeita a importância de equipes em conjunto e unidas para resolver problemas complexos em geral. Os autores notaram que pequenos projetos que tinham equipes pequenas e multifuncionais obtinham os melhores resultados (Takeuchi e Nonaka, 1986).

A denominação da palavra *Scrum*, possui origem no jogo de *rugby*. Quando há uma penalização, para reinício do jogo parte dos jogadores devem fazer uma formação em torno da bola no mesmo local onde ocorreu a penalização. Essa formação é denominada *Scrum*. Essa jogada tem o objetivo de retirar os obstáculos à frente do jogador que correrá com a bola, para que ele possa avançar o máximo possível no campo e marcar pontos (*England Rugby*, 2017).

Figura 3.1 - Formação *Scrum* no *rugby*, *Englad Rugby* (2017).

Durante a jogado do *rugby* denominada *Scrum*, a equipe precisa está unida com um único propósito em formação específica onde a participação de todos é essencial, a falta de comprometimento de um membro pode fazer a formação cair. O *Scrum* em sua essência, seja no *rugby* ou como metodologia de desenvolvimento de produtos e serviços, sempre vai procurar representar o trabalho em equipe, a sincronia, força e inteligência utilizadas juntas para algum objetivo. E assim devem ser os times de desenvolvimento que adotam o método *Scrum* (Schwaber, 1995).

Seguindo as ideias de Hirotaka Takeuchi e Ikujiro Nonaka a metodologia ágil *Scrum* foi documentada e formalizada para desenvolvimento de *software* por Jeff Sutherland, Ken Schwaber e Mike beedle em 1995 (Schwaber, 1995). *Scrum* enfatiza o gerenciamento de projetos, pois as atividades estão específicas no monitoramento e *feedback*, visando à identificação e correção de quaisquer deficiências ou impedimentos no processo de desenvolvimento (Schwaber, 2004).

3.2 *FRAMEWORK*

Um *framework* pode ser compreendido como estrutura básica que pretende servir de suporte e guia para a construção, enquanto uma metodologia todos os processos são conhecidos e previamente definidos, ou seja, prescritivos, diferentemente do *Scrum* onde o processo de aprendizado é empírico. Segundo o guia do *Scrum* (2013), o *Scrum* pode ser definido como um *framework estrutural* que vem sendo utilizado para gerenciar o desenvolvimento de produtos complexos desde 1990 (Schwaber & Sutherland, 2013).

Scrum não é um processo ou uma técnica para construir produtos; em vez disso, é um *framework dentro* do qual você pode empregar vários processos ou técnicas. O *Scrum* foi criado para obter maior desempenho nas práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las. Por ser um *framework* e fornecer uma estrutura ele pode ser combinado com ou complementado por diferentes métodos e práticas consagrados pelo mercado, que podem ser experimentados e adaptados pelo time para seu contexto específico.

Segundo os criados Schwaber e Sutherland no guia do *Scrum*, ele possui três pilares: transparência, inspeção e adaptação.

A transparência é fundamental para que os resultados e objetivos estejam visíveis a todo envolvidos no projeto de entrega. Este princípio mostra a importância da comunicação entre os participantes do projeto, sendo todos observados e cientes das metas a serem alcançadas (Schwaber & Sutherland, 2013).

Os vários aspectos do processo devem ser inspecionados com frequência suficiente para que as variações inaceitáveis no processo possam ser detectadas. A frequência de inspeção tem que levar em consideração que todos os processos são alterados pelo ato da inspeção (Schwaber & Sutherland, 2013).

Quando algum processo, através da inspeção, foi detectado como não aceitável, e afete o produto, os envolvidos devem ajustar o processo ou o material a ser processado, e essa adaptação deve ser feita o mais rápido possível (Schwaber & Sutherland, 2013). O *Scrum* como *framework,* não determina as ações e estratégias a serem tomadas para ajustar os processos (Schwaber, 2004).

3.3 PAPÉIS NO *SCRUM*

O *Scrum* possui três papeis definidos: *Scrum Master*, o *Product Owner* e time de desenvolvimento. Para obter sucesso na implementação e utilização do *Scrum*, cada membro deve assumir responsabilidade distintas, sendo cada integrante responsável e responsabilizado pelos resultados e objetivos alcançados para entrega do produto.

3.3.1 *SCRUM MASTER*

O *Scrum Master* é responsável por garantir que o *Scrum* seja compreendido e aplicado (Schwaver & Sutherland, 2013). Ele potencializa o trabalho do *Product Owner* e do time

de desenvolvimento garantindo que a teoria, práticas e regras do *Scrum* sejam aplicadas de forma correta (Sabbagh, 2013).

Em ambos os times de atuação do *Scrum Master*, seja com o *Product Owner* ou time de desenvolvimento, ele deve capacitar os membros do time do *Scrum* para que sejam auto organizáveis e juntos possam construir o produto, tenham maior comunicação e possam melhorar seus processos (Sabbagh, 2013). Para execução do *Scrum* os membros devem ser capacitados dos eventos exigidos e necessários, obtendo maior comunicação e participação de todos os envolvidos na construção do produto (Schwaver & Sutherland, 2013).

De forma mais especifica, o *Scrum Master* deve auxiliar o *Product Owner* no efetivo gerenciamento do *Backlog* do produto, para que as atividades propostas sejam passadas de forma clara e concisa para o time de desenvolvimento. Com o time de desenvolvimento o *Scrum Master* deve manter em constante inspeção, verificando o desempenho das atividades e retirando os possíveis impedimentos.

Durante o desenvolvimento do produto, podem surgir impedimentos que comprometam a entrega do produto. O *Scrum Master* deve atuar para remover ou gerenciar a remoção dos impedimentos que atrapalham o time de desenvolvimento, prevenindo que os impedimentos aconteçam a cada ciclo de entrega (Sabbagh, 2013). Segundo Sabbagh os impedimentos podem ser classificados pela sua natureza como: organizacionais, básicos, administrativos ou de nível de serviço.

- Impedimentos organizacionais ocorrem quando em situações que necessitam de atuação ou intervenção de outra pessoa, outra equipe ou outra equipe dentro da empresa;
- Impedimentos básicos ocorrem pela falta de um ou mais elementos essenciais à realização do trabalho em algum determinado momento. Exemplo desse tipo de impedimento é a falta de ferramentas ou equipamentos necessários para realização do trabalho, falta de ambiente (cadeiras, mesas) ou até mesmo impossibilidade de acesso;
- Impedimentos administrativos são provenientes de ocorrências administrativas como absenteísmos (atrasos, licenças, folgas, faltas por motivos diversos, etc.), demissões e restrições de horários de trabalho.

- Impedimentos de nível de serviço são identificados como problemas na sustentação de algum serviço em operação como servidor de aplicações, servidores de integração, ferramentas diversas de apoio e ambientes de produção.

Apesar dos times possuírem a característica de serem auto organizáveis, o *Scrum Master* deve observar o andamento do produto a cada ciclo e manter todos focados no objetivo da entrega e do prazo. Seu papel é de facilitar a auto-organização e não de gerenciar, quando necessário deve alinhar os objetivos para que a equipe siga corretamente (Sabbagh, 2013).

O *Scrum Master* possui a característica de ser neutro quanto as decisões, tanto na criação e priorização do *Backlog* do produto quanto no desenvolvimento do produto. Para que seja possível manter a neutralidade do *Scrum Master*, o autor Sabbagh (2013) propõem que o *Scrum Master* não exerça outro papel como do time de desenvolvimento ou *Product Owner*, que tem suas opiniões e interesses.

3.3.2 *PRODUCT* OWER

O *Product Owner* é a pessoal responsável pelas definições do produto e deve tomar as decisões de negócio quanto a construção do produto (Sabbagh, 2013). O *Product Owner* é responsável pelo gerenciamento *Backlog* do Produto, podendo delegar ao time de desenvolvimento, mas ainda continua responsável pelo trabalho. Ele deve gerenciar as atividades buscando maximizar o desempenho e entrega do produto (Schwaver & Sutherland, 2013). Segundo Schwaver e Sutherland (2013) o gerenciamento do *Backlog* do produto inclui:

- Expressar claramente os itens do *Backlog* do Produto;
- Ordenar os itens do *Backlog* do Produto para alcançar melhor as metas e missões;
- Garantir o valor do trabalho realizado pelo time de desenvolvimento;
- Garantir que o *Backlog* do Produto seja visível, transparente, claro para todos, e mostrar o que o Time *Scrum* vai trabalhar a seguir; e,

- Garantir que o Time de Desenvolvimento entenda os itens do *Backlog* do Produto no nível necessário.

O *Scrum* sendo *Framework* não defini como o *Product Owner* deve maximizar o desempenho na entrega do produto. Porem ele deve conhecer muito bem as necessidades do produto e relacionar de forma correta as prioridades para o incremento do produto, consequentemente, agregando maior valor ao produto e aos envolvidos.

O papel do *Product Owner* é exercido por apenas uma pessoa, evitando a geração de dúvidas e conflitos na decisão sobre o produto em desenvolvimento (Sabbagh, 2013). O *Product Owner* deve atuar como um facilitador representando os envolvidos de negócio externos ao time do *Scrum* como investidores, clientes e outros que ajudam a definir as necessidades do produto.

Segundo Sabbagh (2013), podem ocorrer alguns problemas na escolha de quem irá atuar como o *Product Owner*. Quando o cliente escolhe alguém para designar o papel, essa pessoa pode não ter conhecimento das responsabilidades do papel e habilidades necessárias para execução das atividades, sendo esse o problema mais comum na escolha do *Product Owner*. Em alguns casos, o cliente pode possuir uma pessoa que tenha o conhecimento e as habilidades necessárias, porém com pouca disponibilidade devido ao envolvimento com as suas atividades cotidianas no cliente. O autor Sabbagh propõem que o *Product Owner* seja designado pela empresa contratada na construção do produto, realizando frequente contato com os clientes e os demais envolvidos.

Para obter sucesso na execução das atividades, Schwaver e Sutherland descrevem a importância de toda a organização respeitar as decisões do *Product Owner*. As definições do produto estão documentadas no *Backlog* do produto, sendo visível a todos envolvidos e possível de questionamentos e sugestões. Somente o *Product Owner* está autorizado a definir e a priorizar as atividades a serem realizadas pela equipe de desenvolvimento.

3.3.3 TIME DE DESENVOLVIMENTO

O time de desenvolvimento é o grupo responsável em realizar o trabalho de construção e desenvolvimento do produto. As tarefas a serem realizadas pelo o time são definidas e priorizadas pelo *Product Owner*, com as definições do produto (Sabbagh, 2013). Esse time possui a característica de serem auto organizáveis e multidisciplinar, para terem maior facilidade e agilidade na adaptação à mudança de requisitos. A auto-organização facilita

que o próprio time tenha a autonomia das ações para realizarem seu trabalho, ao invés de serem gerenciados por membros fora da equipe. A característica multidisciplinar da equipe, demonstra as competências necessárias para completar o trabalho sem dependência de membros fora da equipe (Schwaver & Sutherland, 2013). Este modelo proposto pelo *Scrum* visa maior produtividade e flexibilidade da equipe na construção do produto e maior agilidade na obtenção dos resultados.

Segundo o guia do *Scrum* (Schwaver & Sutherland, 2013), o time de desenvolvimento deve possuir as seguintes características:

- Eles são auto organizados. Ninguém (nem mesmo o *Scrum Master*) diz ao time de desenvolvimento como transformar o *Backlog* do Produto em incrementos de funcionalidades potencialmente utilizáveis;
- Times de desenvolvimento são multifuncionais, possuindo todas as habilidades necessárias, enquanto equipe, para criar o incremento do Produto.
- O *Scrum* não reconhece títulos para os integrantes do time de desenvolvimento que não seja o desenvolvedor, independentemente do trabalho que está sendo realizado pela pessoa; não há exceções para esta regra.
- Individualmente os integrantes do time de desenvolvimento podem ter habilidades especializadas e área de especialização, mas a responsabilidade pertence ao Time de Desenvolvimento como um todo; e,
- Times de desenvolvimento não contém sub times dedicados a domínios específicos de conhecimento, tais como teste ou análise de negócios.

Para obter maior qualidade na utilização do *Scrum* em times de desenvolvimento, as orientações é que possuam uma quantidade pequena de membros. Times de desenvolvimento grandes geram muita complexidade para um processo empírico gerenciar (Schwaver & Sutherland, 2013). De forma geral, a recomendação é que o time tenha entre 3 e 9 membros (Sabbagh, 2013).

A quantidade de membros no time de desenvolvimento orientado por Schwaver e Sutherland no guia do *Scrum*, servem como parâmetros para obter maior qualidade na execução do *Scrum*. Durante os ciclos do *Scrum* a equipe precisa passar por iterações para manter a comunicação. Times grandes têm maiores dificuldade de comunicação e coordenação, possivelmente gerando desperdício de tempo na produção do produto. Em contrapartida ao time grande, quando se têm um time muito pequeno é possível que o

Scrum gere uma sobrecarga com suas regras, papéis e eventos que supere seus benefícios (Sabbagh, 2013).

O autor Sabbagh (2013) propõe a sugestão que quando há um projeto muito grande com times grandes, é possível dividir em times menores para que seja mais fácil a coordenação e obtenha melhores resultados com o *Scrum*.

3.4 EVENTOS

Os eventos do *Scrum* são os ciclos de desenvolvimento do produto. O *framework Scrum* possui 5 eventos: *Sprint*, *Sprint Planning Meeting*, *Daily Scrum*, *Sprint Review* e *Sprint Retrospective*. Esses eventos possui a característica denominada de *Timebox*, em que os eventos possuem períodos de tempo bem definidos.

Estes eventos são fundamentais para o *Scrum* alcançar seus pilares transparência, inspeção e adaptação. Cada um dos eventos *Scrum* possui uma finalidade específica, agenda de execução e participantes claramente definidos, sendo que a não utilização de algum deles resultará na redução da transparência e na perda de oportunidade de inspeção e adaptação (Schwaver & Sutherland, 2013). Na figura 3.2 ilustra os ciclos de eventos para o incremento do produto no *Scrum*.

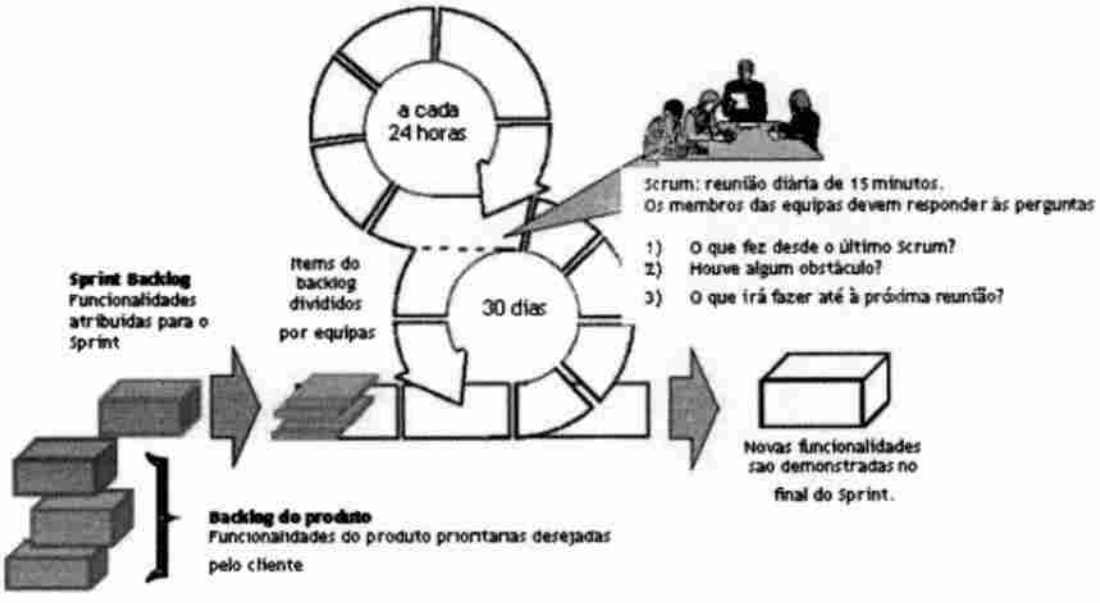

Figura 3.2 Ciclo do *Scrum*, Ferreira et al. (2005).

3.4.1 *SPRINT*

A *Sprint* é o ciclo do desenvolvimento principal do *Scrum*, que tem por objetivo a entrega do incremento do produto, com base no *Backlog* do Produto. As *Sprints* são compostas por uma reunião de planejamento da *Sprint*, reuniões diárias, o trabalho de desenvolvimento, uma revisão da *Sprint* e a retrospectiva da *Sprint* conforme a ilustração abaixo na figura 3.3.

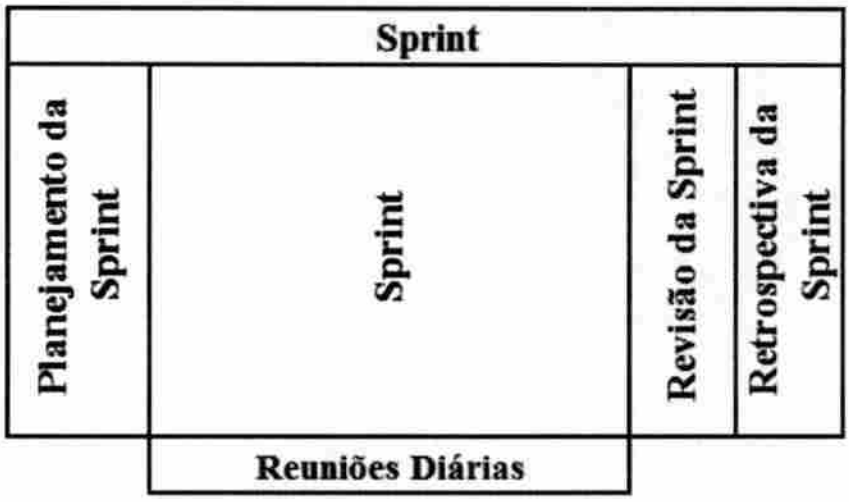

Figura 3.3 Eventos realizados na *Sprint*, Zuliani (2015).

As *Sprints* possuem durações fixas, porem essas durações podem variar de acordo com as necessidades do esforço do time de desenvolvimento para realizar. Quando a *Sprint* é iniciada a sua duração é fixada e não pode ser reduzida ou aumentada. A duração da *Sprint* deve ser menor que o período de um mês (Schwaver & Sutherland, 2013). Durações superiores a um mês podem gerar riscos para a entrega do incremento do produto, sendo que os requisitos e prioridades podem sofrer alterações, em contrapartida prazos curtos podem não ser suficientes para gerar um incremento no produto satisfatória (Zuliane, 2015). Segundo o autor Sabbagh (2013), os eventos terem durações fixas auxilia ao time de desenvolvimento criar ritmo e disciplina com as entregas e regularidade na obtenção de *feedback*.

Segundo o guia do *Scrum* (Schwaver & Sutherland, 2013), a *Sprint* pode ser cancelada pelo *Product Owner* caso o objetivo se torne obsoleto, devido a mudanças estratégicas da organização ou mudanças nas tecnologias. Quando ocorrer o cancelamento da *Sprint*, os itens feitos devem ser avaliados pelo *Product Owner* para aprovação ou

serem estimados novamente para próxima *Sprint*. A possibilidade de cancelamento da *Sprint* é baixa devido as durações de execução serem curtas.

3.4.2 *SPRINT PLANNING*

A *Sprint Planning* é o primeiro evento que ocorre na execução da *Sprint*, que tem por objetivo planejar os itens do *Backlog* do Produto a serem desenvolvidos durante a *Sprint*. A participação é obrigatória do *Product Owner* e do time de desenvolvimento, onde serão negociados os itens a serem desenvolvidos e o incremento do produto, essa lista de itens a serem entregues formam o artefato *Sprint Backlog*. Através do *Sprint Backlog* é possível o time do desenvolvimento monitor a finalização e andamento das atividades para a entrega do incremento do produto. A participação do *Scrum Master* é obrigatória, porém com o papel somente de facilitar a iteração (Sabbagh, 2013).

A duração desse evento é definida paralela a duração da *Sprint*, se a duração da *Sprint* for de um mês esse evento terá a duração máxima de oito horas, para *Sprints* menores a duração desse evento será menor (Schwaver & Sutherland, 2013).

O guia do *Scrum* (Schwaver & Sutherland, 2013), afirma que a respostas das questões abaixo devem ser respondidas no planejamento da *Sprint*:

- O que pode ser entregue como resultado do incremento na *Sprint*?
- Como o trabalho necessário será realizado para entregar o incremento?

O autor Sabbagh (2013), sugeri que a *Sprint Planning* seja realizada em dois momentos, pois a segunda pergunta de como o trabalho será realizado não necessita da participação do *Product Owner*, mas somente do time de desenvolvimento. Nesse segundo momento o time de desenvolvimento irá discutir em termos técnicos a construção e desenvolvimento do incremento, sendo o envolvimento do *Product Owner* desnecessário.

3.4.3 *DAILY SCRUM*

A *Daily Scrum* é uma reunião curta que ocorre diariamente com o time de desenvolvimento. Essa reunião possui a duração máxima de quinze minutos e acontece preferencialmente no mesmo local e horário (Schwaver & Sutherland, 2013) (Sabbagh, 2013). A realização dessas reuniões tem como objetivo disseminar conhecimentos sobre a realização das atividades, identificar impedimentos e priorizar o trabalho a ser realizado no dia que se inicia. Durante a reunião, cada membro do time deve responder três perguntas:

- O que eu fiz desde a última reunião de *Daily Scrum*?
- O que pretende fazer até a próximo reunião de *Daily Scrum*?
- Possui algum impedimento para realizar as atividades?

O time de desenvolvimento pode monitorar as atividades através dessa reunião, podendo se auto organizar para que os objetivos definidos da *Sprint* possam ser alcançados pela equipe, possivelmente reduzindo os riscos de não alcançarem as metas da *Sprint*.

O *Scrum Master* deve auxiliar o time de desenvolvimento a manter a regularidade das reuniões de *Daily Scrum* no horário e local constante. O *Scrum Master* deve observar e ponderar os membros para que a duração não ultrapasse a duração de quinze minutos. Os impedimentos devem ser informados ao *Scrum Master* no momento em que surgirem, a informação dos impedimentos durante a *Daily Scrum* é para criar visibilidade dos impedimentos para o time de desenvolvimento (Sabbagh, 2013).

3.4.4 *SPRINT REVIEW*

A *Sprint Review* é uma reunião que ocorre somente uma vez no final da *Sprint*, em que o Time de Desenvolvimento e o *Product Owner* com auxílio do *Scrum Master*, devem apresentar a entrega do produto desenvolvida na *Sprint*. A presença é opcional de outros envolvidos como clientes e investidores cujo *feedback* é considerado relevante pelo *Product Owner* (Sabbagh, 2013). O evento deve possuir uma duração máxima de quatro horas para uma *Sprint* de quatro semanas (Schwaver & Sutherland, 2013). O *Scrum Master* deve auxiliar na condução e assegurar que os objetivos sejam alçados nessa duração de tempo.

O *Product Owner* deve validar os itens concluídos do *Backlog* do Produto e os itens que ainda faltam implementar, inspecionando o incremento do produto. Finalizado a revisão *Backlog* do Produto é possível definir o *Backlog* da próxima *Sprint*, sendo possível em cada reunião avaliar prazos e orçamentos do projeto (Schwaver & Sutherland, 2013).

Em todas as reuniões de *Sprint Review*, o time do *Scrum* deve avaliar os possíveis problemas e juntos buscar novas maneiras para execução da atividades e correção dos problemas. Com essa reunião é possível que novas decisões surjam, sendo planejadas para as próximas *Sprint*, facilitando a execução da *Sprint Planning*.

3.4.5 *SPRINT RETROSPECTIVE*

A *Sprint Retrospective* é o último evento dentro da *Sprint*, ocorre após a finalização da *Sprint Review*. O objetivo dessa reunião e avaliar o Time *Scrum* e possivelmente criar um plano de melhorias a serem aplicadas nas próximas *Sprint*, porem melhorias podem ser adotadas a qualquer momento (Schwaver & Sutherland, 2013). Essa reunião possui a duração máxima de três horas para uma *Sprint* de quatro semanas. Assim como na *Sprint Review*, O *Scrum Master* deve auxiliar na condução e assegurar que os objetivos sejam alçados nessa duração de tempo.

O Time *Scrum* deve avaliar em relação às pessoas, aos relacionamentos, aos processos e às ferramentas. Através destes pontos de atenção, o time deve identificar quais foram os sucessos e as possíveis melhorias. Identificado os possíveis pontos de melhorias, o time cria um plano para implementar essas melhorias no modo que realizam suas atividades no *Scrum* (Schwaver & Sutherland, 2013). A implementação destas melhorias na próxima *Sprint* é a forma de adaptação à inspeção que o Time *Scrum* faz a si própria.

3.5 ARTEFATOS

Segundo o Guia do *Scrum* (Schwaver & Sutherland, 2013) os artefatos do *Scrum* representam o trabalho ou o valor para o fornecimento de transparência e oportunidades para inspeção e adaptação. O *Scrum* defini a utilização de três artefatos: *Product Backlog*, *Sprint Backlog* e Incremento do Produto.

3.5.1 *PRODUCT BACKLOG*

O *Product Backlog* é uma lista que contém todas as necessidades do produto que será desenvolvido pelo time de desenvolvimento no decorrer do projeto (Sabbagh, 2013). Esta lista é gerenciada pelo *Product Owner*, sendo sua responsabilidade manter atualizada e ordenadas as necessidades.

A lista de itens do *Product Backlog* sofre constante alteração pelo *Product Owner*, pois a medida que o produto e o ambiente evolui é possível identificar com maior clareza os detalhes do produto e possivelmente aumentar os itens da lista (Schwaver & Sutherland, 2013). Segundo Schwaver e Sutherland (2013) os itens são ordenados pelo seu valor, risco,

prioridade e necessidade, sendo os primeiros itens de necessidade imediata para desenvolvimento.

3.5.2 *SPRINT BACKLOG*

O *Sprint Backlog* é a lista de itens selecionados do *Product Backlog*, para serem desenvolvidos no ciclo da *Sprint* (Sabbagh, 2013). Os itens são extraídos do *Product Backlog*, com base nas prioridades definidas pelo *Product Owner* e a capacidade de desenvolvimento pelo time de desenvolvimento. A criação e determinação do *Sprint Backlog* é feita na reunião de *Sprint Planning*.

O time de desenvolvimento é responsável pelo gerenciamento do *Sprint Backlog*. O *Sprint Backlog* pode ser alterado durante todo o ciclo da *Sprint*, mesmo sendo para atualizar tarefas finalizadas como também para inserir e modificar novos itens. O *Sprint Backlog* é um retrato visível das atividades realizadas na *Sprint* (Schwaver & Sutherland, 2013). Na figura 3.5.2 abaixo é possível ver a ilustração da visibilidade do *Sprint Backlog*.

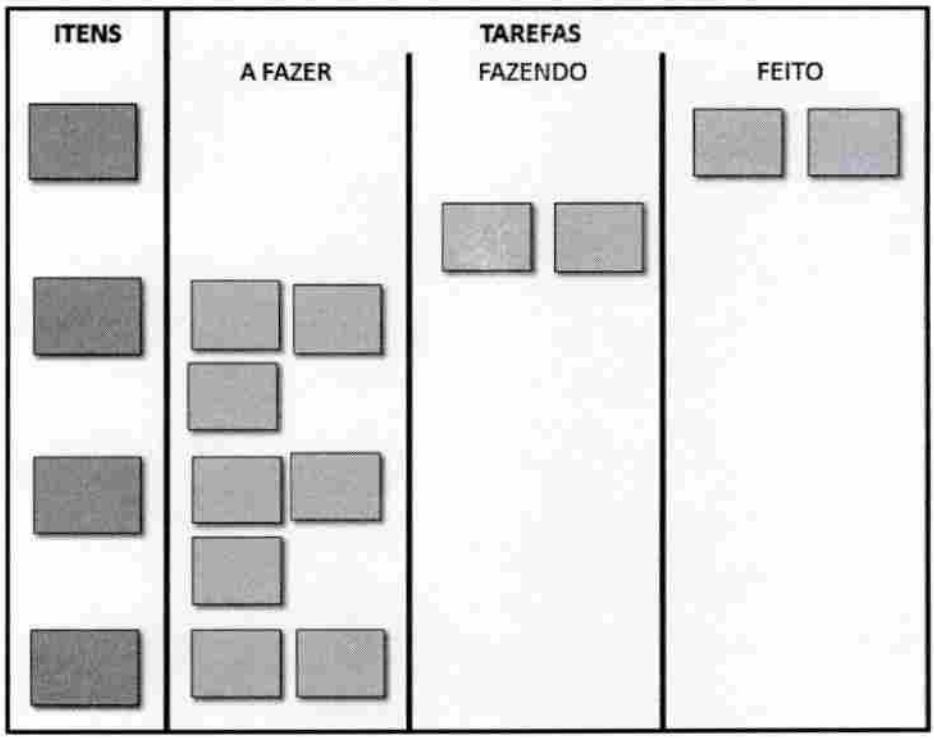

Figura 3.4 - Ilustração do Sprint Backlog, Sabbah (2013).

3.5.3 INCREMENTO

Ao final de todas as *Sprints*, o time de desenvolvimento deve realizar a entrega parcial do produto denominada como incremento. O incremento é definido no guia do *Scrum* como o resultado da finalização dos itens do *Product Backlog* durante a *Sprint* somado aos incrementos das *Sprints* anteriores (Schwaver & Sutherland, 2013).

Realizado a entrega do incremento ao *Product Owner*, ele deve avaliar se essa entrega ira compor uma nova release para os clientes finais, ou será acumulada com outros incrementos para gerar um release de entrega final (Sabbagh, 2013). Em cada incremento novas funcionalidade ou correções do produto final é entregue.

4 – *EXTREME PROGRAMMING*

Este capitulo irá apresentar a revisão bibliográfica da metodologia XP, sendo dividido em seções. A seção 4.1 apresenta a origem da metodologia XP. A seção 4.2 apresenta as definições do XP. A seção 4.3 apresenta os valores necessários para utilização do XP. A seção 4.4 apresenta as práticas necessárias para execução e utilização do XP. A seção 4.5 apresenta o resultado comparativo entre as metodologias.

4.1 ORIGEM

A criação da metodologia XP, embora utilizado pela primeira vez em 1996, é considerado como o resultado na junção de princípios e boas práticas de programação feitos por Kent Beck e Ward Cunninhgam, em um período de uma década enquanto trabalhavam como consultores de problemas em *Smalltalk* (Castro, 2007) (Santana, 2008).

Em março de 1996, Kent Beck foi convidado pela empresa Chrysler para realizar o projeto de reescrever a aplicação da folha de pagamento da empresa. O projeto foi denominado de C3 - *Chrysler Comprehensive Compensation System* (Sistema de Compensação Abrangente da Chrysler). O objetivo do projeto era unificar os quatro sistemas de *software* legado diferentes que estavam sendo usados há vintes anos, sendo oitenta e seis mil o número de funcionários.

O projeto C3 foi concluído com sucesso em 1997 por Kent Beck, sendo esse o primeiro projeto a utilizar a metodologia XP (Teles, 2006). Essa metodologia vem sendo utilizado, desde 1997, por várias empresas ao redor do mundo, especialmente nos Estados Unidos e Europa (Castro, 2007) (Teles, 2006).

4.2 DEFINIÇÃO

A XP é uma metodologia ágil voltada para a construção de *software*, porem ela possui características multidisciplinares e pode ser adaptada e utilizada em outras áreas (Bezerra & Conceição, 2012).

Segundo Santa Junior (2008), a metodologia XP possui ênfase em atividades que tragam valor ao cliente, que enfatiza o desenvolvimento de *software* onde os requisitos vagos ou em constantes mudanças. Teles (2006) afirma que o método XP é um processo de

desenvolvimento de *softwares*, que busca assegurar que o cliente receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento.

Segundo Kuhn e Pamplona (2009), em uma equipe de XP devem possuir desenvolvedores com alguns papeis e responsabilidades diferentes são eles:

- Gerente de projeto - Pessoa responsável pelos assuntos administrativos do projeto, buscando maior relacionamento com cliente para que o mesmo participe das atividades do projeto.
- Líder Técnico – Pessoa com maior conhecimento do processo de desenvolvimento e auxilie as práticas do XP, inspecionando e sinalizando a equipe.
- Analista de teste – Pessoa responsável em garantir a qualidade do sistema através dos testes escritos. Ele deve ajudar o cliente a escrever os casos de testes e no final de cada iteração verificar se o *software* atende todos os casos de testes.
- Redator técnico – Pessoa responsável em criar as documentações do sistemas, permitindo maior dedicação dos desenvolvedores na codificação. Essa pessoa deve estar alinhada com todos da equipe, para que a documentação reflita o código escrito e as regras de negócio atendidas pelo sistema.
- Desenvolvedor - Pessoa responsável em analisar, projetar e codificar o sistema. Naturalmente existe níveis distintos de desenvolvedores dentro de uma equipe, mas com as práticas do XP, como programação em par, a tendência é a equipe se tornar uniforme em suas habilidades.

4.3 VALORES

A metodologia XP é composta por quatro valores: *feedback*, comunicação, simplicidade e coragem.

4.3.1 *FEEDBACK*

O *feedback* pode ser definido como o processo de troca de informações que ocorre entre cliente e a equipe de desenvolvimento durante a produção de um *software*. Esse

mecanismo do XP, permite uma maior produtividade, sendo que as entregas da equipe de desenvolvimento são avaliadas e direcionadas de acordo com a resposta do cliente. Para Teles (2006) um *feedback* constante é a base de todos os processos ágeis de desenvolvimento.

O desenvolvimento no XP é organizado para obter o *feedback* dos clientes em ciclos curtos. Apresentando as funcionalidades o cliente avalia e posiciona caso não esteja seguindo as necessidades. As necessidades solicitadas pelos clientes, em muitos casos, não são compreendidas na sua totalidade pela equipe de desenvolvimento. Segundo Teles (2006), quanto mais cedo for identificado as mudanças através do *feedback* mais barato fica para corrigir e melhor a satisfação do cliente. A satisfação do cliente é algo fundamental para o sucesso do projeto.

Os clientes não têm como prever corretamente todas as funcionalidades de que necessitarão, por isso, é fundamental que haja uma forte interação com os desenvolvedores ao longo do projeto. As entregas e interações possibilita o cliente ter maior visão sobre aquilo que seria adequado para se produzir no *software* (Teles, 2005).

4.3.2 COMUNICAÇÃO

A comunicação não está presente somente no desenvolvimento de *software*, mas está presente no âmbito geral para obtenção de sucesso nos projetos. Segundo Teles (2005), para a realização de um projeto de *software*, normalmente envolvem a presença mínima de duas pessoas, um usuário e um desenvolvedor, o que causa a necessidade de comunicação entre elas. As organizações, em geral, têm encontrado dificuldades em implantar um processo de comunicação eficiente

Nos métodos tradicionais de desenvolvimento de *software*, as equipes de desenvolvimento gastam muitos esforços para criar documentações extensas. Com a presença constante do cliente durante a construção do *software*, é possível obter maior comunicação e clareza das necessidades do cliente. Mesmo quando há uma documentação, os desenvolvedores podem ter perguntas que ajudariam a entender rapidamente a necessidade ou a falta de clareza do cliente (Teles, 2006).

Segundo Soares (2004), a melhor forma de comunicação em XP é a conversa face-a-face, pois através dela é possível que o interlocutor leve em conta o conteúdo emocional da fala, dos gestos e da expressão facial para interpretar a informação transmitida. Teles

(2005) em sua dissertação, exemplifica a importância da comunicação face-a-face da seguinte forma:

> "A riqueza do meio de comunicação exerce influência ainda maior quando se observa a transmissão de conhecimento tácito. Imaginemos uma situação na qual uma pessoa ao telefone tenta ensinar a seu interlocutor como dar um laço no cadarço de seu tênis. Usando o telefone, as chances de sucesso são reduzidas. Entretanto, se os interlocutores estivessem na presença um do outro, seria fácil demonstrar como dar um laço através de um exemplo. Além disso, à medida que o aprendiz fizesse algumas tentativas, o mentor poderia fornecer *feedback* de modo a corrigir eventuais erros. "

A comunicação com XP em projetos busca envolver ativamente seus usuários fazendo com que se tornem parte integrante da equipe de desenvolvimento (Teles, 2005).

Para obter maior qualidade na comunicação entre os integrantes da equipe, a quantidade de pessoas deve ser avaliada. Segundo Beck (2000), o XP precisa possuir uma quantidade reduzida de participantes, frequentemente menor que doze pessoas.

4.3.3 SIMPLICIDADE

A simplicidade em XP tem por objetivo desenvolver apenas o suficiente para atender as necessidades atuais do cliente, desprezando qualquer funcionalidade não essencial. Segundo Teles (2005), os projetos de construção de *softwares* frequentemente investem grande parte dos recursos em esforços desnecessários.

Em muitos projetos de *software*, as definições são fixadas e as alterações no decorrer dos desenvolvimentos são evitados. Consequência dessa utilização, os *softwares* entregues não possuem as necessidades desejadas do cliente e acabam entregando algo inutilizado (Teles, 2005). Essas situações implicam na perda de tempo, aumento nos custos e disponibilização de pessoas em tarefas desnecessárias.

A construção de funcionalidade simples que gastem menos tempo de desenvolvimento e que funcione, possibilita a redução de custo e cria oportunidade de *feedback* rápido do cliente (Kuhn & Pamplona, 2009). Teles (2006) afirma que ao codificar

uma funcionalidade visando atender problemas futuros, recorre a um erro muito frequente em criar trabalhos especulativos, e a possível "ganha de tempo" acaba virando desperdício.

4.3.4 CORAGEM

A metodologia XP foi criada a partir de melhorias em diversas premissas dos processos tradicionais de desenvolvimento, por isso a XP afirma a necessidade da equipe criar confiança em adotar as boas práticas. Os projetos XP deixam claros de que problemas irão acontecer, inclusive aqueles mais temidos. Entretanto, a equipe utiliza de meios que possam ajudar a reduzir ou eliminar as consequências desses problemas (Teles, 2005).

A coragem é uma ferramenta substancial para adotar uma forma de pensamento diferenciado frente as dificuldades. As mudanças que ocorrem no projeto não devem ser encaradas negativamente apenas como problemas a serem resolvidos, mas sim como oportunidades a serem exploradas para a melhoria como um todo da equipe e do projeto (Teles, 2006).

Segundo Beck e Fowler (2001) alguns medos destacam e exercem influência significativa nos processos de desenvolvimento, seja com os clientes ou os desenvolvedores.

Clientes temem:

- Não obter o que pediram;
- Pedir a coisa errada;
- Pagar demais por muito pouco;
- Jamais ver um plano relevante;
- Não saber o que está acontecendo; e
- Fixarem-se em suas primeiras decisões e não serem capazes de reagir a mudanças nos negócios.

Desenvolvedores temem:

- Ser solicitados a fazer mais do que sabem fazer;
- Ser ordenados a fazer coisas que não fazem sentido;
- Ficar para trás tecnicamente;
- Receber responsabilidades, sem autoridade;
- Não receber definições claras sobre o que precisa ser feito;
- Sacrificar a qualidade em função de prazo;

- Ter que resolver problemas complicados sem ajuda; e
- Não ter tempo suficiente para fazer um bom trabalho.

4.4 PRÁTICAS

4.4.1 CLIENTE PRESENTE

No desenvolvimento de *software*, como em outras áreas, necessitam de pessoas que autem com responsabilidades diferente, no caso do desenvolvimento de *software*: cliente e equipe de desenvolvimento (Teles, 2005). Diferente das metodologias tradicionais, em que os clientes não possuem presença constante, o XP visa integrar o cliente ao time de desenvolvimento para juntos obterem maior sucesso na definição e construção do *software* (Kuhn & Pamplona, 2009).

O XP defini as funcionalidades do sistema em breve estórias em conjunto com os testes conceituais e serão estes os indicadores para uma boa implementação (Kuhn & Pamplona, 2009). Segundo Kuhn e Pamplona (2009), durante o desenvolvimento através das estórias nada mais eficaz do que dialogar com o cliente para entender a estória, fazendo-se necessária a presença do cliente no ambiente de desenvolvimento.

Segundo Teles (2005), a participação do cliente tem sido apontada como um dos principais fatores a gerar falhas nos projetos de *software*. Essa prática do XP se preocupa com esta questão e procura solucioná-la, não somente uma mudança ideológica, mas uma mudança física. O XP sugere que o cliente e a equipe de desenvolvimento trabalhem no mesmo ambiente, possibilitando a constante comunicação e *feedback* (Teles, 2005).

4.4.2 JOGO DO PLANEJAMENTO

No XP as funcionalidades do sistema são descritas em estórias pelo cliente e devem ser estimadas pela equipe de desenvolvimento. Feita a estimativa de tempo, é preciso que o cliente priorize as estórias para o desenvolvimento. Segundo Teles (2005), decidir o que implementar é uma das atividades mais importantes a serem conduzidas durante o desenvolvimento de um sistema.

Esse planejamento ocorre várias vezes durante o projeto e deve assegurar que a equipe de desenvolvimento entregue o que é mais necessário ao cliente, respeitando a simplicidade do cliente (Teles, 2005) (Kuhn & Pamplona, 2009).

O XP trabalha com ciclo de iterações e entrega de releases, visando que o *software* seja entregue de forma incremental. Durante a reunião de planejamento, as estórias são selecionadas e alocadas em releases que devem possuem o intervalor máximo de dois meses. As estórias são divididas em tarefas para serem executadas em cada iteração, podendo ela ter duração entre uma a três semanas (Kuhn & Pamplona, 2009).

Em cada ciclo de release é papel no cliente gerenciar o escopo do que está sendo desenvolvido, buscando obter a melhor release com a data estimada dos desenvolvimentos (Teles, 2005).

A figura 4.1 abaixo, foi criada pelos autores Daniel Wildt, Dionatan Moura, Guilherme Lacerda, Rafael Helm (2005), em uma ilustração das regras de movimento do "jogo" de planejamento dividido em duas partes: Planejamento de release e planejamento de Interações.

Figura 4.1 Jogo do Planejamento, Wildt et al. (2005).

4.4.3 STAND UP MEETING

A *Stand Up Meeting* é uma breve reunião realizada diariamente, normalmente de manhã. Essa reunião possui aproximadamente a duração de vinte minutes e deve preferencialmente ser realizada com todos os participantes em pé (Kuhn & Pamplona, 2009).

Durante essa reunião, os participantes devem informar os resultados obtidos no dia anterior, e através dessas informações a equipe deve priorizar as serem realizado no dia que se inicia. O XP assegura que as atividades executadas sejam as atividades com maior

prioridade estabelecida pelo cliente, agregando maior valor e agilidade na entrega das releases (Teles, 2005). Segundo Teles (2006), no trecho abaixo ele afirma que a equipe após a execução da *Stand Up Meeting* tem capacidade de se reorganizar para obter maior desempenho.

> "No fim do *stand up meeting*, cada membro da equipe sabe o que deverá fazer ao longo do dia. É importante notar que a decisão sobre o que fazer ao longo do dia não é tomada por uma única pessoa. Essa decisão é tomada em equipe. Isso faz sentido, porque quando todos se reúnem, é possível ter uma visão de todo o desenvolvimento e não apenas dá uma parte. Desta forma, é factível decidir com mais eficácia quais são as prioridades do dia" (Teles, 2006).

4.4.4 PROGRAMACAO EM PAR

O XP exige que todo e qualquer código implementado no projeto seja efetuado em dupla, diante do mesmo computador, revezando. Um deles é responsável pela digitação denominada de condutor e outro acompanhando o trabalho do parceiro denominado de navegador (Kuhn & Pamplona, 2009).

Essa técnica ainda é vista como desperdício por muitos, pelo fato de utilizar mais recurso, porem seu potencial traz muitos benefícios quantitativos ao projeto. Segundo Maguire (1995), a cada dez linhas de código implementados um erro é gerado, sendo que para correção de um erro gasta em média doze horas.

Enquanto um desenvolvedor implementa o outro desenvolvedor faz uma inspeção imediata de todo o código que é produzido. Identificar possíveis erros mais cedo pode significar uma economia no tempo dedicado à depuração e solução de bugs. Mesmo que eventuais problemas apareçam nos sistemas, a programação em par acelera a depuração e localização dos erros (Teles, 2005).

Segundo Kuhn e Pamplona (2009), um dos maiores benefícios na programação em par é troca de experiência e ideias entre os desenvolvedores. Quando dois desenvolvedores estão atuando juntos, o resultado é soma de duas ideias de soluções diferentes que tornam a codificação mais simples e eficaz.

Durante a execução dos projetos é inevitável que impedimentos aconteçam, como perda ou ausência temporária de um membro, e que isso acabe afetando o andamento do projeto. A programação em par é útil para evitar esses tipos de cenários. O conhecimento é

passado constantemente entre as duplas. Consequentemente a equipe de desenvolvimento eleva o conhecimento de todos, sendo possível igualar o conhecimento de todos (Teles, 2005).

A produtividade de uma equipe que utiliza programação em par comparada a equipes que tenham desenvolvedores sozinhos é praticamente a mesma. Os benefícios na qualidade do código são superiores para programação em par, facilitando a manutenção e outros ganhos a médio e longo prazo (Kuhn & Pamplona, 2009). A programação em par permitir que codificação seja mais rigorosa com os padrões definidos.

4.4.5 CÓDIGO COLETIVO

Essa prática do XP, em comum com os objetivos da programação em par, defini que todos os desenvolvedores tenham acesso e permissão para alterar todas as partes do sistema (Teles, 2005).

É comum ver equipes de desenvolvimento serem divididas por áreas especificas de conhecimento. Essa divisão gera um risco grande para entrega dos projetos, sendo possível esse membro da equipe de afastar e os outros membros não terem os conhecimentos necessários para continuar o desenvolvimento (Teles, 2005).

O XP sugeri que a equipe possua revezamento das suas atividades, quanto a diferenciação dos conhecimentos, sendo possível criar uma equipe mais robusta e capacitada. Mesmo que não tenham os revezamentos, os desenvolvedores têm acesso total ao sistema e podem buscar o auto aprendizado (Teles, 2005).

4.4.6 CÓDIGO PRADONIZADO

Na utilização do XP, os desenvolvedores devem se reunir antes do início do projeto e definir os padrões de codificação. O padrão definido preferencial deve seguir os utilizados pela comunidade da linguagem de desenvolvimento (Kuhn & Pamplona, 2009).

A utilização de padrão de codificação auxilia que a equipe de desenvolvimento a alcançar as outras práticas do XP como programação em par, código coletivo e legível para todos os integrantes da equipe (Teles, 2005).

4.4.7 DESIGN SIMPLES

O XP visa obter maior satisfação do cliente através dos *feedbacks* constantes e iterações de intervalos de tempos reduzidos. Para que seja possível realizar as entregas em curto período de tempo, o design do sistema deve ser o mais simples possível para que atenda a necessidade do cliente (Kuhn & Pamplona, 2009) (Beck, 2000).

Para alcançar essa simplicidade e previsão de possíveis alterações, muitas equipes de desenvolvimento buscam criar sistemas genéricos que sejam mais fáceis de sofrer mudanças. Essas construções de soluções generalistas acabam gerando desperdício de esforços, sendo que a maioria do que foi feita não será utilizada (Teles, 2005).

Segundo Teles (2005), as equipes devem manter a simplicidade e eventuais flexibilidades sejam adiadas até que se torno efetivamente necessárias pela solicitação do cliente. Portanto, devem esperar o *feedback* das iterações e receber as informações concreta sobre onde a arquitetura precisa ser melhorada.

4.4.8 DESENVOLVIMENTO ORIENTADO A TESTES

A criação de código que contenham erros e bugs possuem uma incidência muito grande de ocorrência, como vimos na programação em par. A utilização dessa técnica pelo XP unida com as demais, esforça ao máximo para diminuir a incidência de defeitos e acelerar a identificação dos mesmos (Teles, 2005).

A técnica de desenvolvimento orientado a testes, orienta que primeiro seja criado os mecanismos de testes automatizados para depois a criação dos códigos. Segundo Teles (2005), quanto mais demorado for a identificação dos erros maior será o custo para resolver. Quando um defeito é identificado, faz-se necessário depurar o *software*, sendo essa uma parte difícil e lenta da programação de sistemas

A criação dos testes automatizados busca comprovar as solicitações dos usuários estão sendo atendidas de forma completa, podendo gerar as saídas esperadas para cada entrada possível. Essa técnica auxilia os desenvolvedores a começar a identificar os elementos necessários como classes, métodos que serão necessários para codificação das solicitações (Teles, 2005).

O XP utiliza a criação de dois tipos de testes, os testes de unidade e de aceitação. O teste de unidade verifica os resultados obtidos assegurando que o sistema funcione corretamente. Os testes de aceitação verificam a interação dos componentes construídos

com os demais já existentes, garantindo a funcionalidade completa do sistema. Esses testes devem ser criados pelos desenvolvedores com auxílio do cliente para descrever as necessidades (Beck, 2000) (Kuhn & Pamplona, 2009).

4.4.9 REFATORACAO

Para obtenção de sucesso na construção de sistemas, a boa qualidade influencia na continuidade e manutenção dos mesmos. O XP valoriza todas as práticas de desenvolvimento que criem códigos com a maior qualidade possível (Teles, 2005). A refatoração é processo de identificação de códigos com baixa qualidade, mesmo que estejam funcionais, e melhorar sem alterar o seu funcionamento.

Segundo Kuhn e Pamplona (2009), os desenvolvedores que identificarem código duplicado, pouco legível, mal codificado, sem padronização, lendo, com código legado ou uso incorreto de outras implementações devem obrigatoriamente realizar a refatoração desse código. Teles (2005), afirma que a refatoração deve alcançar as seguintes características:

- Simplicidade
- Clareza
- Adequação ao uso
- Ausência de repetição
- Ausência de funcionalidades extras

Os desenvolvedores possuem os testes automatizado como forma de auxílio para executar a refatoração. Ao finalizar as melhorias, devem executar os testes e verificar se resultados obtidos mantem o funcionamento dos mesmos (Kuhn & Pamplona, 2009).

4.4.10 INTEGRACAO CONTINUA

Realizada a divisão das tarefas, cada desenvolvedor devera trabalhar em uma parte do sistema que integre ao todo no final. No desenvolvimento utilizado o XP, as duplas precisam ser capazes de evoluir rapidamente sem interferir no trabalho das demais duplas (Teles, 2005).

A técnica de integração continua no XP, faz com que as duplas trabalhem de forma isolada, mas que produzam versões pequenas e faça a integração rapidamente a solução total, se possível diversas vezes ao dia (Kuhn & Pamplona, 2009) (Beck, 2000).

Quando as duplas realizam a integração de partes pequena da solução, os possíveis problemas identificados tende a ser mais rápidos para resolver. Todas as duplas ao realizar a integração devem garantir que os testes sejam executados com sucesso e o sistema está consistente e preparado para receber as demais integrações (Teles, 2005) (Beck, 2000).

4.4.11 RELEASE CURTOS

Seguindo os valores do manifesto ágil em ter *software* em funcionamento, o XP valoriza obter entregas rápidas maximizando o retorno dos investimentos do cliente (Teles, 2005).

A realização dessas entregas rápidas deve iniciar com o trabalho de priorização das estórias pelo cliente, sendo o que é mais necessário e agregue maior valor será construído primeiro (Teles, 2005). A release é o conjunto de funcionalidade bem definidas que gastem até dois meses para serem entregues (Kuhn & Pamplona, 2009).

Essa técnica possui muito aceitação pelos clientes, que desejam o mais rápido possível ter o sistema em funcionamento (Teles, 2005). A utilização dessa técnica auxilia a gestão de risco do projeto (Teles, 2006). Segundo Teles (2005), o cliente tem a oportunidade de realizar o *feedback* concreto sobre o real potencial de retorno do projeto possivelmente decidir a continuidade do projeto.

4.4.12 RITMO SUSTENTAVEL

Muitos projetos *software* acabam passando por problema no planejamento tempo, consequência disso começam a utilizar de horas-extras de trabalho levando os desenvolvedores ao seu limite.

Kuhn e Pamplona afirma que nos primeiros momentos a adoção de horas-extras possuem efeitos positivos, porém com o tempo o rendimento da equipe cai drasticamente elevando as margens de erros e concentração. Visando obter o máximo de aproveito da equipe, o XP recomenda que as jornadas de trabalho tenham duração de oito horas diárias, não sendo preciso a utiliza horas-extras.

Para obter sucesso nas metas, os prazos devem ser mantidos e as atividades devem ser ajustadas e distribuídas para se adequar aos prazos (Teles, 2005). Podendo dessa forma manter o desempenho da equipe com boa concentração sem desgastes físicos.

4.5 RESULTADOS OBTIDOS

As metodologias apresentadas para o estudo foram o *Scrum* e XP. As duas metodologias compartilham os valores do manifesto ágil, porem podem ser aplicadas em cenários e necessidades diferentes. Devida a essas diferenças de propostas, foi criado os comparativos visando diferenciar suas atividades e execução.

As duas metodologias implementam os valores e princípios do manifesto ágil, porem cada uma delas possuem especificidades de valores. A tabela 4.1 abaixo mostra os valores que são pilares em cada metodologia.

Tabela 4.1 Comparativo de Valores

Scrum	**XP**
Transparência, Inspeção e Adaptação.	*Feedback*, Comunicação, Simplicidade e Coragem.

Durante a execução e utilização das metodologias, alguns eventos são necessários e obrigatórios para cada uma. A tabela 4.2 mostra o comparativo dos eventos entre o *Scrum* e XP.

Tabela 4.2 Comparativo de Eventos

Scrum	**XP**
O *Scrum* defini a utilização da *Daily Scrum*, *Sprint Planning*, *Sprint* Review e *Sprint* Retrospective. A *Daily Scrum* é a reunião que acontece diariamente para comunicação entre a equipe. Normalmente essa reunião é realizada utilizando também a prática do XP, para que seja realizada em pé e consequentemente seja mais rápida.	O XP através das suas práticas definem o jogo do planejamento e a *Stand Up Meeting*. O jogo do planejamento é uma reunião ocorre periodicamente a cada ciclo de iteração para planejamento das atividades. A *Stand Up Meeting* é a reunião diária que preferencialmente deve ser feita com todos os participantes em pé.

A *Sprint Planning*, *Sprint Review* e *Sprint Retrospective* são reuniões que ocorrem somente uma vez durante a *Sprint*. Com essas reuniões, conforme o capítulo 3, a equipe realiza a definição e priorização das atividades, revisão os acontecimentos e possíveis melhoras no processo e planejam essas melhorias para serem utilizadas.	

O *Scrum* e o XP, sugerem que as equipes de atuação sejam pequenas para obter melhor capacidade de comunicação e interação entres os membros. A tabela 4.3 abaixo mostra o comparativo entre os papeis e responsabilidades que os membros da equipe devem ter.

Tabela 4.3 Comparativo de Papeis e Responsabilidades

Papeis	***Scrum***	**XP**
Cliente	O *Scrum* defini o cliente ou a pessoa responsável pela área de negócio seja o *Product Owner*. Além das definições do produto o *Product Owner* deve priorizar para que as necessidades mais significativas sejam construídas primeiro.	O XP defini o cliente através das suas práticas, colocando algumas características na sua atuação. O cliente deve estar sempre presente e sugeri que cliente trabalhe no mesmo ambiente que a equipe de desenvolvimento, obtendo maior integração com o time e não somente disponível esporadicamente.

Desenvolvimento	O *Scrum* defini que o time de desenvolvimento tenha quantidade reduzida, sendo entre 3 a 9 membros. Esse time possui a característica de serem auto organizáveis e multidisciplinar, para terem maior facilidade e agilidade na adaptação à mudança de requisitos. O time de desenvolvimento deve ser capaz como um todo, em construir e transforma o *backlog* do produto em incremento. O *Scrum* não pode ser confundido como um processo ou uma técnica para construir produto.	O time de desenvolvimento, igualmente ao cliente, é definido através das práticas do XP. Notavelmente o XP possui mais práticas que o *Scrum* focadas no desenvolvimento e como devem ser realizados. O XP defini que o desenvolvimento seja feito em par, os desenvolvedores tenham liberdade e capacidade para desenvolvedor em toda a ferramenta, seja estabelecido um padrão de codificação e cumprido pela equipe, a equipe deve manter o padrão de qualidade de codificação. O XP valoriza os elementos técnicos para executar os desenvolvimentos, buscando maior agilidade e qualidade nas entregas.
Auxilio a Gestão	Para auxílio na gestão e apoio da utilização da metodologia, o *Scrum* defini o papel do *Scrum Master*. Ele deve garantir que a teoria, práticas e regras do *Scrum* sejam aplicadas de forma correta.	O XP não defini o papel ou responsabilidade de um membro para auxiliar ou aplicar a metodologia. Alguns autores sugerem que dentro da equipe de desenvolvimento membros tenham atuações diferentes, sendo uma delas a de gerente de projetos.

Ao realizar o comparativo do *Scrum* e XP, não é possível mostrar um resultado qualitativo que uma será melhor que a outra. A utilização pode beneficiar ou não dependendo das necessidades e cenário aplicado. A tabela 4.4 mostra o comparativo de objetivos entre as duas metodologias.

Tabela 4.4 Comparativo de Objetivos

Scrum	**XP**
O *Scrum* é fundamentado nas teorias empíricas de controle de processos e possui seu foco em maximizar as habilidades da equipe em responder de forma ágil as mudanças constantes. O *Scrum* é um framework e deve ser utilizado para gerenciar de forma ágil o desenvolvimento de produtos complexos.	O XP busca através trazer mudanças na forma como são construídos e gerenciados os projetos de *software*. Através do conjunto de valores e práticas objetivam a construção de *software* com maior qualidade e assertividade das definições com o cliente.
O *Scrum* pode ser melhor aplicada para obter resultados gerenciais de forma ágil	O XP visa obter melhores resultados na codificação do *software*

A tabela 4.5 abaixo mostra o comparativo do *Scrum* e XP, quanto a sua capacidade de adaptação e utilização em outras áreas de negócio.

Tabela 4.5 Comparativo de Adaptação e Utilização

Scrum	**XP**
O *Scrum* como um framework não defini regras e atividades especificas na construção, mas estrutura o gerenciamento da construção de um produto, sendo possível e facilmente adaptado e utilizado no gerenciamento de projetos que não sejam de *software*.	O XP por possuir muita das suas práticas especificas para o desenvolvimento de *software*, não possui a capacidade de adaptação e utilização em outras áreas de negócio.

As duas metodologias geram artefatos e utilizam de documentos diferentes para seguir o desenvolvimento dos produtos. A tabela 4.6 abaixo mostra o comparativo dos artefatos gerados pelo *Scrum* e XP.

Tabela 4.6 Comparativo de Artefatos

Scrum	**XP**
O *Scrum* defini a utilização de três artefatos: *Product Backlog*, *Sprint Backlog* e Incremento do Produto.	O XP sugeri que sejam usados como artefatos o cartão de estória, plano de testes e incremento.

O *Product Backlog* é a lista das necessidades do produto. Essa lista deve ser gerenciada e priorizada pelo *Product Owner*. A *Sprint Backlog* é a lista das necessidades do produto a serem desenvolvidas durante o ciclo da *Sprint*. O incremento do produto é a entrega parcial do produto das necessidade no *Sprint Backlog*.	As estórias são utilizadas para armazenar as definições informadas pelo cliente de uma determinada funcionalidades. Após a criação e documentação da estória é passado para a equipe de desenvolvimento fazer a estimativa de prazo. Para cada estória a ser desenvolvida deve criar o artefato de teste que irá validar o funcionamento conforme as definições. O incremento, igualmente ao *Scrum*, é a entrega parcial do *software* construído a partir das estórias selecionadas como prioritárias.

Para obter qualidade na construção e desenvolvimento de um produto de *software*, como em outras áreas, precisam ser realizados testes e aprovação do produto. Na tabela 4.7 abaixo mostra o comparativo de testes.

Tabela 4.7 Comparativo de Testes

Scrum	**XP**
No *Scrum* os testes ocorrem paralelo as implementações realizadas no produto.	A codificação no XP é inspirada na criação previamente dos testes, evitando possíveis erros. As funcionalidades são aprovadas somente após a execução com sucesso dos testes.

No decorrer da execução das duas metodologias, cada uma das metodologias define ciclos diferentes para realização dos eventos, atividades e entrega do produto. Na tabela 4.8 abaixo é apresentado o comparativo dos ciclos de entrega entre as metodologias.

Tabela 4.8 Comparativo de Entrega

Scrum	XP
O *Scrum* realiza a entrega do incremento do produto ao final de cada *Sprint*, podendo variar de duas a quatro semanas de duração.	Para o XP, a duração da entrega do incremento do produto é relativa a capacidade de produção da equipe, porem devem ter duração máxima de dois meses para obedecer a pratica de releases curtos.

5 - CONCLUSÕES

As metodologias ágeis foram criadas com a visão de absorver melhor as mudanças e falta das definições, podendo realizar entregas rápidas com melhor qualidade para o cliente. Os dezessete signatários do manifesto ágil, possuíam experiências e conhecimentos técnicos diferente, mas todos compreendiam as necessidades de melhorias na reformulação das prioridades na construção de *software* para obter melhores resultados.

Com a pesquisa foi possível observar que as metodologias ágeis têm sido bem aceitas pela indústria de *software*, uma vez que o foco principal é obter valor para os clientes. A aplicação de métodos ágeis nos projetos de *software* obteve melhores resultados e conseguiram administrar melhor as constantes mudanças alcançando maior assertividade de prazos, dos custos e ganho de qualidade na entrega.

Com os dados comparativos é possível que as empresas analisem as atividades e execuções de cada uma das metodologias, para auxiliar na escolha e opção de utilização de cada uma delas. Com os dados comparativos é possível observar melhor os objetivos e foco entre as duas metodologias. As duas metodologias não possuem restrições de utilização integradas, sendo possível utilizar e adaptar para juntas alçarem benefícios diversos.

5.1 - TRABALHOS FUTUROS

As duas metodologias possuem objetivos semelhantes, porém foco e maneiras diferentes de executar. Em trabalhos futuros, podem analisar e criar a integração das duas metodologias, fazendo com que uma complemente a outra. O *Scrum* e o XP não possuem restrições de utilização e podem ser adaptadas para realizar essa integração.

As duas metodologias devem ser aplicadas para times pequenos. Alguns autores sugerem para que equipe grandes do *Scrum* seja divido em núcleos menores para não prejudicarem a comunicação entre os integrantes. Em trabalhos futuros pode analisar ferramentas e adaptação nas metodologias para poderem aceitar times grandes, mantendo a qualidade de comunicação e feedback.

Apesar de ser focada no desenvolvimento, a metodologia XP em trabalhos futuros pode ser estudada novos meios para transformar as suas práticas e valores para que seja possível adaptar e utilizar em outras áreas de negócio. A utilização do XP possui muitos

benefícios na qualidade do *software* entregue e pode ser convertido em qualidade na entrega de outros produtos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABES - Associação Brasileira das Empresas de *Software*. (2017). Mercado Brasileiro de *Software*: panorama e tendências. Disponível em: http://www.abes*software*.com.br/ >. Acesso em: maio de 2017.

ABRAHAMSSON, P.; SALO, O. (2002) *Agile Software Development Methods – Review and Analysis*. VTT Publications 478 107, Espoo.

ALMEIDA, Marcell. (2014). O que é Manufatura Enxuta? Disponível em: https://medium.com/*product*rocks/o-que-%C3%A9-manufatura-enxuta-b2a168d088ae. Acesso em: maio de 2017.

AMBLER, Scott W. (2004). *Lessons in Agility from Internet-Based Development*. IEEE *Software*. p. 66-73.

BECK, Kent. (2000). *Extreme Programming explained: embrace change*. 1ª ed. *Reading*, MA: Addison-Wesley.

BECK, Kent; FOWLER, Martin. (2001). *Planning Extreme Programming*. 1ª ed. Boston: Addison-Wesley.

CARVALHO, B. D., & Mello, C. H. P. (2012). Aplicação do método ágil *Scrum* no desenvolvimento de produtos de *software* em uma pequena empresa de base tecnológica. Gestão & Produção, 19(3).

CASTRO, Vinicius A. (2007). Desenvolvimento Ágil com Programação Extrema. Monografia – Curso de Ciência da Computação, Universidade Federal de Sergipe, São Cristóvão, Sergipe.

CAVALCANTI, Ricardo de Oliveira. (2006). O Toyota *way* e o desenvolvimento ágil de *software*.

COCKBURN, A. (2007). *Agile software development: the cooperative game*. 2. ed. Reading, MA, Estados Unidos: Addison-Wesley.

DANTAS, V. F. (2003). Uma Metodologia para o Desenvolvimento de Aplicações Web num Cenário Global. Dissertação de mestrado. Universidade Federal de Campina Grande. Centro de Ciências e Tecnologia. Campina Grande.

DANTAS, V. F. (2003). Uma metodologia para o desenvolvimento de aplicações Web num cenário global. Dissertação Mestrado em Centro de Ciências e Tecnologia, Universidade Federal de Campina Grande, Campina Grande.

ENGLAND RUGBY. (2017). Disponível no link: http://www.englandrugby.com/my-

rugby/players/rugby-basics/set-pieces/. Acessado em: junho de 2017.

FERNANDES, Jorge Fabio. (2008). CMMI e ITIL como ferramentas para aumentar a qualidade e eficiência no desenvolvimento de *software*. Monografia para obtenção do grau de Tecnólogo em Processamento de Dados. Faculdade de Tecnologia de São Paulo. São Paulo.

FERREIRA, Décio; COSTA, Felipe; ALONSO, Felipe; ALVES, Pedro; NUNES, Tiago. (2005). *Scrum* Um modelo Ágil para Gestão de Projetos de *Software*. Engenharia de *Software*, FEUP - Portugal.

FOWLER, M. (2000). *Put your process on a diet. Software Development*, CMP Media.

GLOGER, Boris. (2006). *Scrum Alliance. Scrum Delivers*. Disponível no Link: https://www.*Scrum*alliance.org/community/articles/2006/may/*Scrum*-delivers. Acesso em: junho de 2017.

HIGHSMITH, J. (2004) *Agile project management*: *Creating Innovative Products*. Reading, MA, Estados Unidos: Addison-Wesley.

HIGHSMITH, Jim. (2001) História do manifesto ágil. Disponível no link: http://agilemanifesto.org/history.html. Acesso em: maio de 2017.

JAKOBSEN, C.R.; JOHNSON, K.A. (2008). *Mature Agile with a Twist of* CMMI. *Agile* '08 *Conference*.

KNIBERG, H. (2007). "*Scrum* e XP direto das Trincheiras", Editora C4 Media, Publisher of.

KUHN, Giovane Roslindo; PAMPLONA, Vitor Fernando. (2009). Apresentando XP. Encante seus clientes com *Extreme Programming*. Disponível no link: http://javafree.uol.com.br/artigo/871447/Apresentando-XP-Encante-seus-clientes-com-Extreme-Programming.html. Acesso em: Julho de 2017.

MAGUIRE, Steve, (1995). *Writing Solid Code*, Microsoft Press.

MANN, C. & MAURER, F. (2005). *A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction. Agile Development Conference*, p. 70-79. IEEE *Computer Society*.

MELO, C. de O., Santos, V., Katayama, E., Corbucci, H., Prikladnicki, R., Goldman, A., & Kon, F. (2013). *The evolution of agile software development in Brazil. Journal of the Brazilian Computer Society*,

MORYEN, Roy. (2005). *Agile Management and the Toyota Way for Software Project Management*. In: IEEE, Perth, Austrália. *Conference Proceedings* Perth.

MUNDIM, A. P. F. et al. (2002). Aplicando o cenário de desenvolvimento de produtos em

um caso prático de capacitação profissional. Gestão & Produção, v. 9.

MUNDIM, A. P. F.; ROZENFELD, H.; AMARAL, D.C.; SILVA, S.L.; GUERRERO, V.; HORTA, L.C. (2002). Aplicando o cenário de desenvolvimento de produtos em um caso prático de capacitação profissional. Gestão & Produção.

PMI (2014). *Project Management Institute.* PMI-ACP - Profissional Certificado em Métodos Ágeis. Disponível no link: https://brasil.pmi.org/brazil/CertificationsAndCredentials/PMI-ACP.aspx. Acesso em: junho de 2017.

POPPENDIECK M.; POPPENDIECK. (2003). T. *Lean Software Development.* Addison Wesley.

PRESSMAN, R. S. (1995). Engenharia de *Software.* Makron Books. São Paulo. Brasil.

RISING, L.; Janoff, N. (2000) *The Scrum software development process for small teams.* IEEE, v. 17, nº 4.

SALO, O.; ABRAHAMSSON, P. (2008). *Agile methods in European embedded software development organizations: a survey on the actual use and usefulness of Extreme Programming and Scrum.* IET *Software.*

SANTANA JUNIOR, Célio Andrade. (2008). Avaliação da utilização de Metodologias Ágeis no contexto dos modelos de qualidade de *software.* 112 f. Dissertação (Mestrado de Engenharia da Computação) – Departamento de Sistemas e Computação da Escola Politéc- nica de Pernambuco (UPE), Pernambuco.

SCHWABER, K. (1995). *SCRUM Development Process.* Disponível no link: http://www.jeffsutherland.org/oopsla/schwapub.pdf. Acessado em: Junho de 2017.

SCHWABER, K. (2004). *Agile Project Management with Scrum.* Microsoft Press.

SCHWABER, K.; BEEDLE, M. (2002) *Agile Software Development with SCRUM.* Prentice Hall.

SETHI, TS; HARI, CVMK; KAUSHAL, BSS; SHARMA, A. (2011). *Cluster Analysis and Pso for Software Cost Estimation. Information Technology and Mobile Communication*, v. 147, p. 281-286.

SOARES, Michel dos Santos. (2004). Metodologias Ágeis *Extreme Programming* e *Scrum* para o Desenvolvimento de *Software.* Universidade Presidente Antônio Carlos. Disponível no link: ftp://atenas.cpd.ufv.br/dpi/mestrado/XP/artigo06.pdf. Acessado em: julho de 2017.

SOARES, Michel dos Santos. (2004). Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de *Software.* INFOCOMP *Journal of*

Computer Science, v. 3, n. 2, ISSN 1982-3363. Disponível no link: http://www.dcc.ufla.br/infocomp/index.php/INFOCOMP/article/view/68. Acesso em: junho de 2017.

TAKEUCHI, Hirotaka, and IKUJIRO Nonaka. (1986). *"The New New Product Development Game."* Harvard Business *Review* 64, no. 1.

TAVARES, Breno Gontijo. (2015), Análise da Gestão de Riscos no desenvolvimento de projetos de *software* via *Scrum*. Universidade Federal de Itajubá, Programa de pós-graduação em engenharia de produção.

TELES, Vinícius Manhães. (2005). Um estudo de caso da adoção das práticas e valores do *Extreme programming*. 179 f. Dissertação (Mestrado em Informática) – Instituto de Matemática e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro.

TELES, Vinícius Manhães, (2006). *Extreme Programming*: aprenda como encantar seus usuários desenvolvendo *software* com agilidade e alta qualidade 2ª ed. São Paulo, Novatec.

VAN CAUWENBERGHE, Pascal. *The Toyota Way of Managing Projects*. In: XPDAYS GERMANY. Karlsruhe. 2005. Disponível no link: http://xpdays.de/2005/sessions/The_Toyota_Way.html. Acesso em: maio de 2017.

VERSIONONE. (2016). *VersionOne* 10th *Annual State of Agile Report*. Disponível no link: http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf. Acessado em: junho de 2017.

ZULIANI, Emerson. (2015). EVENTOS DO *SCRUM* – A SPRINT. Disponível no link: http://emersonzuliani.com.br/eventos-do-*Scrum*-a-sprint/. Acesso em: Julho de 2017.