UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DÉBORA PENHA FERREIRA

APLICAÇÃO DA METODOLOGIA ÁGIL SCRUM: UMA PROPOSTA PARA O SETOR DE DESENVOLVIMENTO DO NÚCLEO DE TECNOLOGIA DA INFORMAÇÃO DA UFMA

São Luís
2015

DÉBORA PENHA FERREIRA

APLICAÇÃO DA METODOLOGIA ÁGIL SCRUM: UMA PROPOSTA PARA O SETOR DE DESENVOLVIMENTO DO NÚCLEO DE TECNOLOGIA DA INFORMAÇÃO DA UFMA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profª. Ma. Maria Auxiliadora Freire

São Luís
2015

Ferreira, Débora Penha

Aplicação da metodologia Scrum: Uma proposta para o setor de desenvolvimento do Núcleo de Tecnologia da UFMA/
Débora Penha Ferreira. – São Luís, 2015.
61f.

Monografia (Graduação) – Curso de Ciência da Computação, Universidade Federal do Maranhão, 2015.

Orientador: Profª Maria Auxiliadora Freire

1. Desenvolvimento de software. 2. Scrum. 3. Metodologias agéis. I. Título.

Débora Penha Ferreira

APLICAÇÃO DA METODOLOGIA ÁGIL SCRUM
UMA PROPOSTA PARA O SETOR DE DESENVOLVIMENTO DO NÚCLEO DE TECNOLOGIA DA INFORMAÇÃO DA UFMA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em: 24/07/2015

BANCA EXAMINADORA

Profª. Ma. Maria Auxiliadora Freire (Orientadora)
Mestra em Ciência de Engenharia
Universidade Federal do Maranhão

Prof. Me. Carlos Eduardo Portela Serra de Castro
Mestre em Informática
Universidade Federal do Maranhão

Prof. Dr. Samyr Béliche Vale
PhD em Ciência da Computação
Universidade Federal do Maranhão

AGRADECIMENTOS

Agradeço primeiramente a Deus, Criador dos céus e da Terra, por seu amor incondicional, que permitiu que tudo isso acontecesse ao longo da minha vida. Por me dar forças, sabedoria para vencer todos os desafios. Sem Ele nada sou e nada posso fazer.

Agradeço ao meu pai Zedequias e minha mãe Rosinete que nunca mediram esforços para dar o melhor para suas filhas. Pela criação que me deram, por tanto amor, carinho, compreensão, por terem me dado à oportunidade e muito incentivo aos meus estudos e pelo apoio que necessitei nos momentos mais difíceis desta caminhada.

As minhas irmãs Ruty e Sarah pela verdadeira amizade, por todo amor carinho e atenção, pelos preciosos conselhos nos momentos que mais necessitei e por estarem sempre comigo em todos os momentos de minha vida.

Agradeço ao meu melhor amigo e namorado Mauro Romero, que apareceu no momento em que mais precisei. Foi a pessoa que compartilhei todas as minhas tristezas e alegrias. Obrigada pela paciência, compreensão, apoio, carinho, amor, pelas horas despendidas a mim, revisando os meus textos e me dando todo apoio, acreditando na minha capacidade.

Agradeço a minha orientadora, Profª. Maria Auxiliadora, não sei o que seria de mim sem a sua paciência, incentivo, atenção e disponibilidade em me ajudar nessa fase final do curso.

Agradeço aos professores do curso de Bacharelado em Ciência da Computação da UFMA, que contribuíram muito para a minha formação.

Agradeço também a todos os meus colegas e amigos de curso, em especial a minha grande amiga e companheira, que sempre me ajudou em todos os momentos, Luciana Mendes.

Agradeço a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigada.

“Suba o primeiro degrau com fé. Não é necessário que você veja toda a escada. Apenas dê o primeiro passo.”

(Martin Luther King)

RESUMO

Uma nova abordagem para desenvolvimento de software tem despertado grande interesse entre as organizações de todo o mundo. Trata-se das metodologias ágeis que vêm se tornando bastante popular por apresentar como característica uma abordagem simplificada com foco em encontrar uma forma de trabalho para os membros da equipe, de desenvolvimento de software, produzirem o software de forma flexível e em um ambiente em constantes mudanças. Dentro das metodologias ágeis, surgiu uma nova abordagem, denominada Scrum, que procurou otimizar ainda mais a entrega do produto de software aos clientes. Pela experiência dos últimos anos, metodologias ágeis, como o Scrum, muitas vezes precisam ser adaptadas às reais necessidades da empresa. O principal objetivo desta pesquisa é contribuir para os estudos de viabilidade relacionados à adaptação dessa metodologia a um ambiente em que se tem muita demanda de projetos e tarefas, porém com uma forma de desenvolvimento de software que não traz resultados efetivos. Para isso, é apresentado uma proposta da adaptação do Scrum, no Núcleo de Tecnologia da Informação (NTI) da Universidade Federal do Maranhão (UFMA), mostrando os benefícios decorrentes dessa adaptação.

Palavras-chave: Desenvolvimento de software. Scrum. Metodologias agéis. Adaptação

ABSTRACT

A new approach to software development has aroused great interest among organizations around the world. These are the agile methodologies that have become quite popular to present as characteristic a simplified approach focused on finding a way to work for members of the software development team to produce the flexible software and an environment in constant change. Within the agile methods, a new approach has emerged, called Scrum, which sought to optimize the delivery of the software product to customers. The experience of recent years, agile methodologies such as Scrum, often need to be adapted to real business needs. As this type of adaptation has been little studied, the main objective of this research is to contribute to feasibility studies related to the adaptation of this method to an environment where you have a lot of demand for projects and tasks, but with a form of software development that does not bring effective results. For this, a proposal for the adaptation of Scrum is presented in the Information Technology Center (NTI), Federal University of Maranhão (UFMA), showing the benefits of such adaptation.

Keywords: Software Development. Scrum. Agile methods. Adaptation

LISTA DE FIGURAS

Figura 2.1- Funcionamento do Scrum 34
Figura 4.1 - Estrutura organizacional do NTI.....48
Figura 4.2- Página de Tarefas do Redmine 49
Figura 4.3 - Sprint Backlog no Redmine.....54

LISTA DE TABELAS

Tabela 2.1 - Comparação de caracteristicas dos metodologias ágeis XP, Scrum, FDD, ASD e Crystal 35
Tabela 4.1- Principais mudanças na adaptação do Scrum 55

LISTA DE SIGLAS

ASD – Desenvolvimento de software adaptativo

FDD – Desenvolvimento dirigido por funcionalidades

NTI - Núcleo de Tecnologia da Informação

UFMA - Universidade Federal do Maranhão

XP – *Extreme Programming*

Sumário

1 INTRODUÇÃO 13

1.1 Motivação 14

1.2 Objetivos 14

1.3 Organização do Trabalho 14

2 METODOLOGIA ÁGIL DE DESENVOLVIMENTO DE SOFTWARE 16

2.1 Valores das metodologias ágeis 17

2.2 Princípios da metodologia ágil 18

2.3 Metodologias de desenvolvimento ágil 21

2.3.1 *Extreme Programming* 22

2.3.1.1 Práticas do XP 23

2.3.1.2 Vantagens e Desvantagens do XP 25

2.3.2 Crystal 26

2.3.2.1 Vantagens e desvantagens do Crystal 28

2.3.3 Desenvolvimento dirigido por funcionalidades (FDD) 29

2.3.3.1 Vantagens e desvantagens do FDD 30

2.3.4 Desenvolvimento de software adaptativo (ASD) 31

2.3.4.1 Vantagens e desvantagem do ASD 32

2.3.5 Scrum 33

2.3.5.1 Vantagens e desvantagens do Scrum 35

2.4 Comparativos entre metodologias ágeis 35

3 SCRUM 39

3.1 Papéis e Responsabilidades 39

3.1.1 Scrum *Master* 39

3.1.2 *Product Owner* 39

3.1.3 Time Scrum 40

3.2 Eventos do *Sprint* 40

3.2.*1 Sprint Planning* 41

3.2.*4 Daily Scrum* 42

3.2.5 Sprint Review 43

3.2.6 *Sprint Retrospective* 44

3.3 Artefatos 44

3.3.1 Product Backlog 44

3.3.2 Sprint Backlog 45
3.4 Considerações finais 45
4 ESTUDO DE CASO - UMA PROPOSTA DO SCRUM PARA O NÚCLEO DE TECNOLOGIA DA INFORMAÇÃO DA UFMA (NTI - UFMA) 47
4.1 Ambiente do estudo de caso 47
4.2 Definição do estudo de caso 50
4.3 Proposta do Scrum para o NTI 51
4.3.1 Definição dos papéis 51
4.3.2 Desenvolvimento do processo 53
4.3.3 Principais mudanças 55
5 CONCLUSÃO 58
REFERÊNCIAS 60

1 INTRODUÇÃO

Atualmente, o desenvolvimento de software é uma atividade cada vez mais vulnerável de erros de projeto e de execução e, entre seus principais riscos, pode-se citar os gastos que sempre superam o orçamento, o consumo de tempo que supera o cronograma, os requisitos mal elaborados e a baixa qualidade nos sistemas desenvolvidos (FOWLER, 2005).

Em virtude desse cenário, alguns fatores críticos ou exigências para o sucesso se destacam: a agilidade, a capacidade de adaptação, o poder de inovar de forma rápida e eficiente e o potencial de aprimoramento contínuo sob grandes restrições de recursos. Em resposta a essas exigências, vem se buscando cada vez mais melhorias nessa área.

As metodologias ágeis de desenvolvimento vêm ganhando atenção na indústria de software, devido a sua proposta de liberar o produto de forma mais rápida do que os modelos prescritivos de processo. Diferentemente dos modelos tradicionais, que tentam planejar tudo no inicio do desenvolvimento para controlar possíveis modificações de requisitos, os metodologias ágeis defendem o planejamento contínuo, para que as adaptações a essas modificações possam ocorrer (PRESSMAN, 2010).

As metodologias ágeis de desenvolvimento de software se propõem a construir software com maior produtividade e, sobretudo, com qualidade garantida. Para isso elas encaram os projetos sobre um novo paradigma e defendem a adoção de uma série de princípios e práticas. Na metodologia ágil, a evolução é vista a todo o momento, seja em uma reunião, seja ao final de cada semana de trabalho (FOWLER, 2005). As metodologias de desenvolvimento ágil visam, principalmente, aos aspectos humanos dentro de um projeto e, justamente por isso, que eles demonstram sucesso.

Em alguns casos, nem todos os conceitos utilizados em uma metodologia podem ser implementados no ambiente da empresa, e outros conceitos considerados importantes para a empresa podem não ser contemplados pela metodologia escolhida. Em situações como essa, o que normalmente ocorre é uma adaptação da metodologia às necessidades da empresa. A adaptatividade é uma das características mais marcantes de metodologias ágeis como o Scrum.

O Scrum se destaca por ser uma abordagem enxuta de desenvolvimento de produtos. Ele é um processo ágil de desenvolvimento de produto ou administração de qualquer trabalho iterativo e incremental e pode ser aplicado ao desenvolvimento de produtos de maneira geral (SABBAGH, 2013). O uso dessa metodologia tem sido amplamente divulgado por todo o mundo e vem causando grandes modificações nas formas de pensar e agir de gerentes, clientes e todas as pessoas que estão diretamente ligadas com um projeto de desenvolvimento de software.

1.1 **Motivação**

Apresentar soluções para minimizar os problemas relacionados ao gerenciamento de desenvolvimento de software e viabilizar a melhoria do mesmo no Núcleo de Tecnologia da Informação (NTI) da Universidade Federal do Maranhão, para obtenção de bons resultados.

1.2 **Objetivos**

Este trabalho tem como objetivo principal apresentar uma proposta de adaptação da metodologia ágil Scrum para viabilizar melhorias relacionadas ao gerenciamento de software no Núcleo de Tecnologia da Informação (NTI) da Universidade Federal do Maranhão.

Os objetivos específicos são listados:

a) Comparar os métodos ágeis, apresentando as vantagens do Scrum.
b) Analisar os papéis, eventos e os artefatos necessários para atingir bons resultados com a aplicabilidade do Scrum.
c) Apresentar uma proposta que viabiliza a adaptação do Scrum no setor de desenvolvimento do NTI.

1.3 **Organização do Trabalho**

A apresentação detalhada dos aspectos deste trabalho está distribuída da seguinte forma:

Neste primeiro capítulo foi realizada a introdução deste trabalho.

O segundo capítulo apresentará as principais características da metodologia ágil de desenvolvimento de software, evidenciando seus valores e princípios. Também será mostrada uma visão geral de alguns modelos de desenvolvimento ágil

e suas principais características, vantagens e desvantagens, finalizando o capítulo com um comparativo desses modelos e evidenciando as vantagens do Scrum.

O terceiro capítulo apresenta com detalhes o funcionamento do Scrum, apresentando os papéis, eventos e os artefatos necessários para atingir bons resultados com a aplicação do Scrum.

O quarto capítulo apresenta o estudo de caso, em que consiste em uma análise feita no setor de desenvolvimento do Núcleo de Tecnologia da Informação (NTI) da Universidade Federal do Maranhão (UFMA), mostrando o funcionamento da forma como é feito o gerenciamento de desenvolvimento no ambiente, apontando os principais problemas. Depois dessa análise, será sugerido uma proposta de adaptação da abordagem Scrum, com o intuito de viabilizar a melhoria do gerenciamento de desenvolvimento de software do NTI, obtendo bons resultados.

O quinto capítulo apresenta a conclusão do trabalho, ressaltando as dificuldades e vantagens da proposta alcançada no estudo de caso.

2 METODOLOGIA ÁGIL DE DESENVOLVIMENTO DE SOFTWARE

Por muitos anos a engenharia de software utilizou as metodologias tradicionais para desenvolvimento de software no meio acadêmico e empresarial, onde eram fundamentados em um conjunto de atividades predefinidas, descritas como processos prescritivos (AMBLER, 2004), onde muitas vezes, o trabalho começava com o levantamento completo de um conjunto de requisitos, seguido por um projeto de alto nível, de uma implementação, de uma validação e, por fim, de uma manutenção (SOMMERVILLE, 2007).

Com certo tempo, as pessoas envolvidas no desenvolvimento de software foram percebendo que essas metodologias tradicionais traziam muitas falhas e não estavam apresentando efetividade no desenvolvimento de software, sendo assim, começaram trabalhar de forma diferenciada das metodologias tradicionais. Foram percebendo, ao longo do tempo, que sua forma de trabalho estava sendo bem mais produtivo, do que seguindo as metodologias tradicionais. Aos poucos esse novo modo de trabalho foi sendo aplicado em muitos projetos, transformando-se em novas metodologias de desenvolvimento. Essas novas metodologias foram chamadas de leves, por serem bem mais práticas e simples, sem a burocracia e a exaustão das formalidades das metodologias tradicionais. Com o tempo, muitos dessas metodologias foram sendo aplicados nos meios acadêmicos e empresarial, ganhando espaços para a discursão e grandes debates, principalmente a confiabilidade, a efetividade e o custo benefício que essas metodologias traziam consigo (BASSI, 2008).

As metodologias ágeis surgiram a partir da segunda metade de 1990. Depois de muitos anos utilizando as metodologias que fugiam dos padrões impostos pela indústria de software, 17 líderes perceberam que trabalhavam de forma parecida. Em 2001, se reuniram, durante alguns dias, em uma estação de ski em Utah para discutir o novo modo de trabalho e chegar a um consenso de uma metodologia mais eficiente em relação às tradicionais. Durante a discussão, concluíram que seria muito difícil chegar a uma só metodologia e que o desenvolvimento de software é um processo muito complexo que não poderia ser definido por uma única maneira. Contudo chegaram a um consenso de princípios que seriam determinantes para a obtenção de bons resultados no desenvolvimento de software. Desse encontro,

resultaram-se os 12 princípios básicos, a criação da Aliança Ágil e o estabelecimento do Manifesto Ágil (BASSI, 2008).

2.1 **Valores das metodologias ágeis**

O Manifesto Ágil não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparado com os indivíduos e interações, com o software funcionando, com a colaboração com o cliente e as respostas rápidas a mudanças e alterações (SOARES, 2004). Os conceitos do Manifesto Ágil determinam a preferência e não opções no desenvolvimento de software, focando a atenção em determinados conceitos sem dispensar outro.

A partir da criação do manifesto para desenvolvimento de software, feita pela Aliança Ágil, foram definidos conceitos que implicam em quatro premissas (BECK et al., 2001):

1. **Indivíduos e interação entre eles mais que processos e ferramentas**: Processos e ferramentas podem melhorar a eficiência do desenvolvimento de software, mas para tomar decisões críticas no projeto, só a capacidade e o conhecimento dos indivíduos para suprir essas decisões (SABBAGH, 2013). As ferramentas usadas para auxiliar o desenvolvimento de software devem ser o mais simples possível. Projetos que utilizaram metodologias pesadas em termos de processos e ferramentas, segundo Jim Highsmith (2004 apud SABBAGH, 2013) obteve sucesso fundamentalmente devido às pessoas envolvidas naqueles projetos. Os processos servem para dar suporte ao desenvolvimento de software, e não chefiar como a equipe deve agir, ou seja, os processos que se adaptam as equipes, um processo não é um substituto para uma habilidade, de forma que segui-lo por si só não cria bons softwares.

   2. Software em funcionamento mais que documentação abrangente: Para Cockburn (COCKBURN, 2007), *software* em funcionamento é o único indício de que a equipe de fato produziu. Jim Highsmith (HIGHSMITH, 2004 apud SABBAGH, 2013) afirma que clientes estão mais entusiasmados com resultados, ou seja, estão mais preocupados no *software* em funcionamento do que na entrega de valor de negócio. A documentação é muito importante para o desenvolvimento de software, mas desde que seja realizada a

documentação necessária e suficiente para aquele projeto, sem excessividades. *Software* em funcionamento não exclui a necessidade de documentação, pois permite a comunicação e colaboração, melhora a transferência de conhecimento, preserva informações históricas, ajuda nas melhorias em progresso e satisfaz as necessidades legais e regulatórias (SABBAGH, 2013).

3. Colaboração com o cliente mais que negociação de contratos: No desenvolvimento de software não há separação entre os desenvolvedores e clientes, ambas as partes devem estar inteiramente envolvidas para que se produza um software de qualidade. Segundo Cockburn (COCKBURN, 2007), essa colaboração envolve companheirismo, tomada de decisão conjunta e rapidez na comunicação, de forma que muitas vezes pode até tornar contratos desnecessários. Projetos bem sucedidos são aqueles que se tem uma interação direta com o cliente, entendendo as suas necessidades e ajudando a expressar o que realmente esperam do software.

4. Responder a mudanças mais que seguir um plano: A mudança está presente em todo tipo de negócio e acontece por várias razões: por evolução da tecnologia, leis e regras que mudam conforme o tempo e a mudança de ideia das pessoas. A incerteza é inevitável e está ligada, de alguma forma, ao desenvolvimento de software, então os desenvolvedores precisam está preparados para suprir essas mudanças. Todo projeto de software deve ter um planejamento, mas esse plano deve ser maleável e deve haver flexibilidade de mudança, caso contrário, se torna irrelevante.

2.2 **Princípios da metodologia ágil**

Para ajudar no entendimento das pessoas sobre os conceitos do desenvolvimento ágil de software, foi refinado a partir do Manifesto, por parte de seus autores, os doze princípios básicos (BECK et al., 2001):

1. **Nossa maior prioridade é satisfazer o cliente por meio da entrega cedo e frequente de *software* com valor:** O foco do desenvolvimento de software está em satisfazer as necessidades e os desejos do cliente. A produtividade mais cedo no desenvolvimento dar-se a partir da entrega de partes do produto que atendam as expectativas dos clientes. Esse princípio, segundo

SABBAGH (2013), defende um plano detalhado, sugerindo que a prioridade está em se adaptar e buscar, em cada momento, o que de fato trará valor aos clientes, para entregar-lhes o mais cedo e frequentemente possível.

2. **Mudanças de requisitos são bem vindas, mesmo em fases tardias do desenvolvimento. Os processos Ágeis utilizam a mudança em favor da vantagem competitiva para o cliente:** É preciso aceitar a mudança como algo normal no desenvolvimento de software para atender as necessidades do cliente. Ao se trabalhar em ciclos curtos de *feedback*, permite-se aos clientes evoluírem o produto à medida que melhor entendem suas necessidades e conforme adaptação as mudanças de mercado, tornando-se mais competitivos (SABBAGH,2013).
3. **Entregar softwares em funcionamento com frequência de algumas semanas e alguns meses, de preferência na menor escala de tempo:** Entregas frequentes de parte do produto, para os clientes, gera, a cada entrega, retorno ao investimento dos clientes e permite obter-se *feedback* sobre o que foi produzido, tornando o estado atual de seu projeto transparente e permitindo que eles forneçam um melhor direcionamento ao time de desenvolvimento (AMBLEW, 2009).
4. **As pessoas do negócio e os desenvolvedores devem trabalhar em conjunto diariamente ao longo do projeto:** Todos os envolvidos do projeto de desenvolvimento de software tem um objetivo de garantir valor ao cliente, e para isso, cooperam, continuamente, durante todo o projeto, interagindo com frequência. Segundo Sabbagh (SABBAGH, 2013), esse princípio se opõe ao cenário de antagonismo comum em projetos de desenvolvimento de *software*, nos quais pessoas de negócios — que frequentemente incluem os próprios clientes do projeto — e desenvolvedores raramente se comunicam e, muitas vezes, estão em lados opostos.
5. **Construa projetos em torno de indivíduos motivados. Dê-lhes o ambiente e o suporte que precisam e confie neles para realizarem o trabalho:** Esse princípio vem mostrar que para um bom desenvolvimento de software não se faz só com as melhores ferramentas e processos, mas que tudo depende da motivação das pessoas envolvidas. O ambiente, o suporte e a confiança são os maiores motivadores para um trabalho de qualidade.

6. **O método mais eficiente e efetivo de se transmitir informação para e entre uma equipe de desenvolvimento é a conversa face a face:** A melhor forma de comunicação entre membros do time que desenvolve o produto e entre esse time e o mundo externo é a comunicação face a face, que é direta, síncrona e enriquecida pela entonação de voz, olhar e linguagem corporal, entre outros fatores. Quando a comunicação presencial não é viável (em um projeto distribuído, por exemplo) é uma boa prática fazer-se o melhor uso possível da tecnologia disponível para se aproximar da comunicação face a face. Esse princípio se opõe à utilização de documentos, emails, telefone e teleconferência, entre outros, como formas padrão de comunicação em um projeto (SABBAGH, 2013).
7. ***Software* em funcionamento é a principal medida de progresso:** A principal medida de progresso do software é a entrega em partes para o cliente em que as necessidades sejam supridas. Segundo Sabbagh (SABBAGH, 2013), esse princípio se opõe à prática de se gerar artefatos como protótipos e extensos documentos de planos e especificações e, assim, acreditar que se progrediu no projeto. Isso também se opõe à geração de quaisquer artefatos e partes do produto — inclusive documentação — que não gerem valor para os clientes do projeto.
8. **Os processos Ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente um ritmo constante:** Busca-se promover um ritmo constante e sustentável para o trabalho do time que desenvolve o produto, o que se torna possível quando esse ritmo é apoiado por toda a cadeia, incluindo usuários e patrocinadores. No entanto, ao se exigir do time um compromisso com mais trabalho do que ele é capaz de produzir, são muitas vezes adotadas as horas extras, o trabalho em fins de semana e a pressa exagerado para se cumprir o prazo de entrega, por exemplo. Essas práticas podem levar à insatisfação dos membros do time de desenvolvimento, a uma menor produtividade e a uma menor qualidade no produto gerado (SABBAGH, 2013).
9. **A atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade**: Quando se tem um software de qualidade e produzido com

técnicas excelentes, permite-se que seja facilmente modificado, e assim, chegue o mais perto possível de atender a necessidade do cliente. Assim, a alta qualidade no produto gerado é essencial para manter a Agilidade.

10. **Simplicidade – a arte de se maximizar a quantidade de trabalho não feito – é essencial**: Os desenvolvedores ágeis esforçam-se em maximizar o retorno sobre os investimentos em TI dos clientes, e corta ou automatiza trabalhos pesados (AMBLEW, 2009).
11. **As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto-organizam**: Segundo Sabbagh (SABBAGH, 2013), equipes com maior autonomia são mais eficientes. Essas equipes auto-organizadas trabalham em direção a metas acordadas, mas têm a liberdade de decidir qual a melhor forma de realizar esse trabalho e, assim, são responsáveis e responsabilizadas por seus resultados. Dessa forma, gera-se um melhor produto.
12. **Em intervalos de tempo regulares, a equipe reflete sobre como se tornar mais efetiva e então refina e ajusta seu comportamento de acordo**: Melhoria do processo de software é um esforço contínuo, e técnicas como retrospectivas devem ser adotadas para permitir a melhoraria da sua abordagem ao desenvolvimento de software (AMBLEW, 2009). Para se tornar cada vez mais efetiva, a equipe regularmente inspeciona suas formas de trabalho, identifica pontos de melhoria e se adapta, promovendo o aperfeiçoamento incremental contínua. É a inspeção e adaptação que o time realiza em seus processos de trabalho (SABBAGH, 2013).

2.3 Metodologias de desenvolvimento ágil

Existem várias metodologias classificadas como ágeis que são utilizadas pelas equipes que desenvolvem software. Todas as metodologias tem em comum o fato de serem aplicado em projetos não muito complexos, utilizando ciclos interativos curtos, planejamento guiado por funcionalidades, *feedback* constante, facilidades de aceitar mudanças, interatividade da equipe, intimidade com o cliente e um foco no ambiente geral de trabalho do time (SABBAGH, 2013). Entretanto, assim como existem semelhanças entre as metodologias, existem também diferenças, principalmente no que tange as práticas utilizadas durante o processo de

desenvolvimento e os papéis da equipe. Sendo assim, apresentam-se a seguir as principais características das metodologias ágeis *Extreme Programming* (XP), *Crystal*, Scrum, Desenvolvimento dirigido por funcionalidades (FDD) e Desenvolvimento de software adaptativo (ASD), mostrando uma visão geral e suas principais características, vantagens e desvantagens.

2.3.1 *Extreme Programming*

A metodologia *Extreme Programming*, mais conhecida como XP, é uma evolução de práticas aplicadas em vários projetos nos anos 80 e anos 90, criados por Kent Beck e Ward Cunningham, para uma abordagem de desenvolvimento adaptativo e orientado a pessoas (FOWLER, 2005). O XP é uma metodologia ágil que prioriza a qualidade de projetos e agilidade de equipes, baseado em valores como:

- Comunicação: Princípio do XP que tem como objetivo manter um relacionamento entre o cliente e os desenvolvedores arquitetando um entendimento pessoa-a-pessoa do problema, com o uso máximo de interação “cara-a-cara” entre as pessoas envolvidas no projeto e com o uso mínimo de documentação formal (SAMPAIO, 2004).
- Simplicidade: Possibilita que cada membro da equipe crie códigos simples, ou seja, implementar o software com o menor número possível de classes e metodologias. O propósito é fazer aquilo que é mais simples hoje e criar um ambiente em que o custo de mudanças no futuro seja baixo. O desafio da XP é fazer algo simples hoje e pagar um pouco mais amanhã para fazer alterações necessárias do que implementar algo complicado hoje, como é feita em muitas metodologias tradicionais, que acabam muitas vezes nem sendo usadas (SOARES, 2004).
- *Feedback*: A prática do *feedback* constante significa que o programador terá informações regulares do código e do cliente. A informação do código é dada pelos testes regulares, que indicam os erros tanto individuais quanto do software integrado. Em relação ao cliente, o *feedback* constante significa que ele terá frequentemente uma parte do software totalmente funcional para avaliar. Essa ligação direta com os clientes faz com que, constantemente, o cliente sugira novas características e informações aos desenvolvedores.

Eventuais erros são rapidamente identificados e corrigidos nas próximas versões, fazendo com que no final, o cliente tenha satisfação no software solicitado (SOARES, 2004).

- Coragem: é o princípio extremamente importante para que seja aplicado o XP. Segundo Sampaio (SAMPAIO, 2004): “Por exemplo, não são todas as pessoas que possuem facilidade de comunicação e têm bom relacionamento. A coragem também dá suporte à simplicidade, pois assim que a oportunidade de simplificar o software é percebida, a equipe pode experimentar. Além disso, é preciso coragem para obter *feedback* constante do cliente.” Essas atitudes não devem ser bloqueadas apenas por ter medo de tenta-la, elas são necessárias para trazer melhorias no projeto (SAMPAIO, 2004).

2.3.1.1 Práticas do XP

O foco principal do *Extreme Programming* são doze práticas que foram criadas com base nos princípios pregados pelos valores do XP. Essas práticas já vêm sendo implantadas por tempos atrás. Muitas das práticas de XP são novidades no meio de desenvolvimento de software, como exemplo a programação em pares. Segundo Sampaio (SAMPAIO, 2004), o valor e benefícios de tais práticas devem ser avaliados em conjunto e não individualmente, pois elas foram criadas para serem usadas coletivamente, de forma a reduzir a deficiência umas das outras. As doze práticas de XP são:

- Planejamento: Consiste em avaliar o que é necessário ser feito e o que pode ser adiado no projeto, traçando caminhos a serem seguidos (SOARES, 2004).
- Entregas frequentes: Baseia-se em entregas de versões de menor tamanho possível a cada mês, ou no máximo a cada dois meses, aumentando a possibilidade de *feedback* rápido do cliente, aplicando os requisitos de maior valor para o negócio (MEDEIROS, 2014).
- Metáfora: São as descrições de um software sem a utilização de termos técnicos, apresentado essas descrições de forma mais simples e mais claras com o intuito de facilitar a compreensão e guiar o desenvolvimento do software (SOARES, 2004).

- Projeto simples: O software desenvolvido pela metodologia XP deve ser o mais claro possível e atender os requisitos atuais, sem incomodar com os requisitos futuros.
- Testes: A metodologia XP destaca a validação do projeto durante todo o processo de desenvolvimento.
- Programação em pares: A implementação do código é feita em dupla, ou seja, dois programadores trabalham em um único computador (MEDEIROS, 2014).
- Refatoração: Realça o refinamento do projeto e está presente em todo o desenvolvimento. Segundo Soares (SOARES, 2004): “A refatoração deve ser feita apenas quando necessário, ou seja, quando um desenvolvedor da dupla, ou os dois, percebe que é possível simplificar o módulo atual sem perder nenhuma função”.
- Propriedade coletiva: No XP, todos são responsáveis pelo software inteiro; o código do projeto pertence a todos os membros da equipe (SAMPAIO, 2004).
- Integração contínua: Produzir e interagir o sistema de software várias vezes por dia, mantendo os programadores em concordância, além de possibilitar processos rápidos (MEDEIROS, 2014).
- 40 horas de trabalho semanal: Esta prática procura validar o foco nas pessoas e não em processos e planejamentos. Essa não é uma regra que obriga as equipes em projetos XP a trabalharem somente 40 horas por semana, mas XP assume que não se deve fazer horas extras constantemente. Caso isso ocorra, existe queda de qualidade do código, que deve ser resolvido não com aumento de horas trabalhadas, mas com melhor planejamento (SAMPAIO, 2004).
- Cliente presente: É de fundamental importância a participação do cliente durante todo o desenvolvimento do projeto de software.
- Código padrão: Padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores. Segundo Sampaio (SAMPAIO, 2004), o objetivo é que todos programem da mesma forma, facilitando o entendimento do código e as alterações.

2.3.1.2 Vantagens e Desvantagens do XP

O XP é uma metodologia que estimula a interação contínua entre cliente e desenvolvedor para que as necessidades do cliente sejam visualizadas e compreendidas, obtendo-se assim, o sucesso do projeto. Ela é baseada em valores e princípios exclusivos que devem ser cultivados para que o projeto obtenha o sucesso desejado. Segundo Astels (ASTELS, 2002), as práticas do XP são arquitetadas para funcionarem juntas e fornecerem mais valor do que cada uma poderia prover individualmente. Sendo assim, algumas vantagens que podem ser destacadas:

- Análise prévia de tudo que pode acontecer durante o desenvolvimento do projeto, apresentando qualidade, confiança, data de entregas e custos promissores;
- Com *feedback* constante, faz-se possível adaptar rapidamente eventuais mudanças nos requisitos. O XP é ideal para ser usada em projetos em que o cliente não sabe exatamente o que deseja e pode mudar muito de opinião durante o desenvolvimento do projeto;
- Técnica ágil mais adotada mundialmente;

Apesar de ter inúmeros benefícios, na metodologia XP existem situações que não seria bem aplicado (SAMPAIO, 2004):

- Dificuldade para novas equipes adotarem a prática em um projeto existente. As pessoas envolvidas no desenvolvimento do projeto podem estar inseridas dentro de uma cultura tradicional de desenvolvimento de software, produzindo muita documentação, gastando muito tempo com análise e projeto antecipado, entre outras coisas. Fazer com que a equipe passe a aderir as práticas de XP e esqueça as antigas pode ser muito difícil;
- Difícil imaginar como ficariam alguns conceitos de XP como comunicação, programação em par e outras em uma equipe grande (exemplo: 100 pessoas). O projeto XP deve possuir uma equipe pequena – geralmente até 12 programadores;
- Evitar usar XP quando uma tecnologia é complicada para escrever casos de teste, quando retorna feedback em um tempo longo ou quando não incorpora facilmente as mudanças. A organização do espaço físico onde a equipe de

XP trabalha deve facilitar a comunicação e deixar todos próximos uns dos outros;

- Requer que o cliente participe da equipe do projeto e trabalhe no mesmo local dos demais, estando à disposição, de preferência, o tempo todo para esclarecer dúvidas. Caso não possa ser alguém da organização do cliente deve ser alguém que entenda do negócio do cliente e que seja capacitado para exercer este papel. O cliente não pode estar disponível o tempo todo que for solicitado;

2.3.2 Crystal

Crystal é tido como uma família de metodologias ágeis, criada por Alistair Cockburn e Jim Highsmith a fim de conseguir a "execução de manobras", que é segundo Cockburn (COCKBURN, 2002), especificado por um "jogo cooperativo de invenção e comunicação de recursos ilimitados, com o principal objetivo de entregar softwares prontos e funcionando, com o objetivo secundário de preparar-se para o jogo seguinte".

Segundo Pressman (PRESSMAN, 2010), a família Crystal é na verdade uma união de práticas sustentada por metodologias cada uma com seu elemento central, papéis, padrões de processo, produtos de trabalhos e práticas específicas a cada uma delas. Este conjunto de práticas pode ser adaptada para diferentes tipos de projetos, sendo assim, o objetivo é que uma equipe ágil escolha a melhor prática para ser adotada no seu projeto.

Os membros da família Crystal são identificados por cores que indicam a intensidade do metodologia. Neste caso, quanto mais escura a cor, maior é a complexidade do projeto. *Crystal Clear* é considerada uma metodologia leve, sendo assim para equipes de 2 a 8 pessoas, podendo chegar até 12 em casos especiais; *Yellow*, para equipes por volta de 10 a 20 membros; *Orange* são apropriados para times de 20 a 75 participantes e *Red* para equipes de 75 a 100 membros (COCKBURN, 2002).

O tamanho da equipe, localização geográfica, criticalidade (segurança) e recursos são parâmetros que determinam o método de desenvolvimento. Crystal sugere que a escolha da metodologia deve ser apropriada, considerando fatores como o tamanho do projeto e o quanto pode ser considerado um projeto crítico.

Quanto mais pessoas envolvidas, mais objetos e formalismos são necessários para viabilizar a comunicação (BASSI, 2008).

Segundo Bassi (BASSI, 2008), a família Crystal se apoia em um conjunto de propriedades, estratégias e práticas. As propriedades definem as premissas do modelo que devem ser seguidos durante o desenvolvimento. As principais delas são as entregas frequentes, comunicação intensa e melhora por reflexão. Porém, para cada modelo da família, tem suas propriedades. Para valorizar essas propriedades, que podem ser aplicadas de acordo com as características e o objetivo do projeto, são sugeridas estratégias (BASSI, 2008):

- Exploração de 360 graus: A equipe identifica os pontos de maior dúvida no projeto e explora todas as possibilidades. Podem ser desenvolvidas amostras da solução com as possíveis tecnologias a serem usadas;
- Vitória antecipada: Logo de início deve ser definido um objetivo o mais simples possível. Isso estimula a equipe do projeto, bem como dá uma sensação positiva aos usuários e patrocinadores;
- Esqueleto que anda: Pode existir uma infraestrutura ou ações que podem ser entendidas como o esqueleto do projeto (um exemplo: a comunicação entre tecnologias diferentes). Esse esqueleto deve estar em funcionamento o quanto antes para permitir testes durante a implementação;
- Rearquitetura incremental: Ao longo do projeto a arquitetura do software evolui a partir do esqueleto em funcionamento. O esqueleto possui infraestruturas fundamentais que podem ser substituídas durante o crescimento do software;
- Radiadores de informação: São colocados em lugares de fácil visualização de todos os envolvidos, painéis com informações fundamentais sobre o projeto. As práticas propostas são nove, porém apenas uma é obrigatória, esta é a *reflection workshop*. Segundo Bassi (BASSI, 2008): “O *reflection workshop é* uma reunião periódica focada em três questões principais: ‘o que deve ser mantido?’, ‘onde estamos tendo problemas?’ e ‘o que vamos tentar na próxima iteração?’”. O resultado do *workshop* são três listas com as respostas para essas perguntas apresentadas em um radiador de informações que fica exposto no ambiente de trabalho;

O ciclo de vida da família da metodologia Crystal tem como principais práticas (Cockburn, 2002):

- Metodologia de Formação (*Methodology Shaping*): coleta de informações sobre as experiências anteriores, utilizando-as nas convenções iniciais;
- Oficina de Reflexão (*Reflection Workshop*): a equipe deve fazer uma pausa de uma hora periodicamente, após cada entrega, para refletir sobre o trabalho realizado e realizar pequenas alterações no que for necessário para a próxima entrega;
- Planejamento Relâmpago (*Blitz Planning*): semelhante ao "*planning game*" do XP, porém deve-se fazer uma lista de dependência entre as tarefas;
- *Delphi Estimation*: uma maneira de chegar a uma estimativa inicial para o total do projeto;
- Reuniões Diárias (*Daily Stand-ups Meetings*): maneira rápida de difundir a informação pela equipe; ela não deve ser usada para discutir problemas, e sim para identifica-los;
- *Design* Ágil de Interfaces (*Agile Interaction Design*): uma versão rápida do *design* centrado no usuário;
- Miniatura do Processo (*Process Miniature*): técnica de aprendizagem rápida sobre o processo que será usado;
- Programação lado-a-lado (*Side-by-Side Programming*): uma alternativa menos intensa da Programação em par (aplicado no XP);
- *Burn Charts*: maneira de dar visibilidade ao andamento do projeto, através de relatórios;

2.3.2.1 Vantagens e desvantagens do Crystal

A metodologia Crystal pode ser aplicada em diversos tipos de projetos principalmente por sua capacidade de adaptação, mas essa flexibilidade excessiva pode causar problemas e alguns projetos podem não ser compatíveis com seus modelos de processos. Por ser uma metodologia leve é mais adequada utiliza-la em desenvolvimento de sistemas menores e menos complexos. As suas principais vantagens:

- Entregas frequentes das etapas do projeto, reduzindo o retrabalho;

- Proporciona menos especulações, conversas e mais visibilidade das tarefas que vão sendo executadas;
- Metodologia escalável de acordo com o tamanho do projeto e sua criticidade;
- Única metodologia que especifica o responsável pelo ciclo de vida de projetos críticos;
- Ênfase na parte de testes (geralmente 1 teste para cada time);

As principais desvantagens do Crystal:

- A metodologia não foi desenvolvida para trabalhar com projetos longos;
- Muita documentação pode ser prejudicial para projetos pequenos;
- Pouca atenção para dinâmicas do time;
- Documentação bem mais formal que outras metodologias.

2.3.3 Desenvolvimento dirigido por funcionalidades (FDD)

Desenvolvimento Dirigido por Funcionalidades (*Feature Driven Development*) é uma metodologia ágil para gerenciamento e desenvolvimento de software, criada em 1997 num grande projeto em Java para o *United Overseas Bank*, um banco localizado em Singapura. Surgiu a partir da experiência de análise e modelagem orientadas por objetos de Peter Coad, e de gerenciamento de projetos de Jeff de Luca (ROCHA, 2013).

Segundo Pressman (PRESSMAN, 2006), o FDD é um modelo prático para modelagem em orientação a objetos, onde este possui um processo adaptável e ágil, que pode ser aplicado em projetos de software moderados ou maiores. Segundo Rocha (ROCHA, 2013), esta metodologia chama a atenção por algumas características peculiares:

- Resultados úteis a cada duas semanas ou menos.
- Blocos bem pequenos de funcionalidade valorizada pelo cliente, chamados *"Features"*.
- Planejamento detalhado e guia para medição.
- Rastreabilidade e relatórios com incrível precisão.
- Monitoramento detalhado dentro do projeto, com resumos de alto nível para clientes e gerentes, tudo em termos de negócio.

A FDD é uma metodologia muito objetiva, classicamente descrita por duas fases e composta por cinco processos bem definidos e integrados: a fase de concepção e planejamento, composta pelos processos de “desenvolver um modelo abrangente”, “construir uma lista de funcionalidades” e “planejar por funcionalidade” e a fase iterativa de construção, composta de processos de “detalhar por funcionalidade” e “construir por funcionalidade” (Mallmann, 2011):

- Desenvolver um Modelo Abrangente: pode compreender desenvolvimento de requisitos, análise orientada por objetos, modelagem lógica de dados e outras técnicas para entendimento do domínio de negócio em questão. O resultado é um modelo de objetos (e/ou de dados) de alto nível, que guiará a equipe durante os ciclos de construção;
- Construir uma Lista de Funcionalidades: divisão funcional do modelo do domínio, em três camadas: áreas de negócio, atividades de negócio e passos automatizados da atividade (funcionalidades). O resultado é uma hierarquia de funcionalidades que representa o produto a ser construído (também chamado de *product backlog*, ou lista de espera do produto);
- Planejar por Funcionalidade: envolve a estimativa de complexidade e dependência das funcionalidades, também levando em consideração a prioridade e valor para o negócio/cliente. O resultado é um plano de desenvolvimento, com os pacotes de trabalho na sequência apropriada para a construção;
- Detalhar por Funcionalidade: dentro de uma iteração de construção, a equipe detalha os requisitos e outros artefatos para a codificação de cada funcionalidade, incluindo os testes. O projeto para as funcionalidades é inspecionado. O resultado é o modelo de domínio mais detalhado e os esqueletos de código prontos para serem preenchidos;
- Construir por Funcionalidade: cada esqueleto de código é preenchido, testado e inspecionado. O resultado é um incremento do produto integrado ao repositório principal de código, com qualidade e potencial para ser usado pelo cliente/usuário.

2.3.3.1 Vantagens e desvantagens do FDD

Como FDD possui fases muito fortes de análise, se comparada com outras metodologias ágeis, seus autores a recomendam para qualquer tipo de desenvolvimento. Algumas das vantagens do FDD:

- Vários times podem trabalhar em paralelo;
- Por cada *feature* ser pequena, coletar os requisitos se torna mais fácil, pois estes podem ser mais bem descritos, e durante a revisão, se torna mais fácil encontrar ambiguações e erros;
- Todos os aspectos do projeto são rastreados por funcionalidade;
- Modelagem e desenvolvimento por funcionalidade são fáceis de entender e de adotar;

Quanto à escalabilidade do time, há controvérsias sobre o tamanho mínimo de um time FDD. Ainda que FDD pressuponha um ciclo completo de documentação, a especificação original da metodologia não comenta sobre sua aplicabilidade à manutenção de sistemas. Além disso, outras desvantagens no FDD podem ser destacadas:

- Questionamento sobre efetividade/aplicabilidade do FDD;
- Não existe um consenso do tamanho que cada *feature* deve ter;
- Não tem uma manutenção tão acessível;
- Iterações não são tão bem definidas como nas outras metodologias.

2.3.4 Desenvolvimento de software adaptativo (ASD)

Segundo Pressman (PRESSMAN, 2010), o Desenvolvimento de Software Adaptativo (*Adaptive Software Development*), conhecido como ASD, foi proposto por Jim Highismith como uma técnica para construção de sistemas e software complexos. O ASD possui como filosofia a concentração na colaboração humana e na auto-organização da equipe. O foco desse modelo é a colaboração e auto-organização das equipes (MEDEIROS, 2013).

O modelo adaptativo determina um ciclo de vida baseando-se em três fases: especulação, colaboração e aprendizagem.

Na fase de especulação, projeto é iniciado e tem-se o planejamento de ciclos adaptáveis, que usa as informações contidas no início do projeto como: a missão do cliente, restrições do projeto e os requisitos básicos. Os requisitos essências serão utilizados para definir o conjunto de ciclos da versão, ou seja, os incrementos de software operacional. Esse plano de ciclos sofrerá mudanças. Após completar cada ciclo, o plano é revisado e adaptado para que se tenha o trabalho reajustado à realidade que a equipe ASD está trabalhando (MEDEIROS, 2013).

A colaboração é um fator muito evidenciado na utilização e discussão nos metodologias ágeis. As pessoas motivadas utilizam a colaboração de uma forma que multiplique seus talentos e produções criativas. Porém, colaboração não é algo fácil; por isto envolve comunicação e trabalho em equipe, mas também enfatiza o individualismo, pois a criatividade individual desempenha um importante papel no pensamento colaborativo; sendo assim, trata-se de uma questão de confiança. A colaboração envolve confiabilidade, críticas sem ressentimento, auxílios, trabalho árduo e comunicação dos problemas ou preocupações de forma a conduzir ações efetivas (PRESSMAN, 2010).

O ASD tem como elemento principal o aprendizado, muito importante para que possa se conseguir uma equipe auto-organizada. As equipes ASD aprendem através de três maneiras: grupos focados, revisões técnicas e autópsias de projetos (MEDEIROS, 2013). O ASD é caracterizado por seis propriedades (HIGHSMITH, 2002):

- Orientado a missões: Atividades são justificadas através de uma missão que pode mudar ao longo do projeto;
- Baseado em componentes: Construir o sistema em pequenos pedaços.
- Iterativo: O ASD possui foco em refazer do que fazer corretamente já na primeira vez;
- Prazos pré-fixados: Fixação de prazos para evitar ambiguidade em projetos, com prazos tangíveis forçando com que a equipe defina a risca decisões do projeto logo cedo;
- Tolerância a mudanças: As mudanças são frequentes e é sempre melhor estar pronto a adaptá-las do que controlá-las. Constante avaliação de quais componentes podem mudar;
- Orientado a riscos: Itens de alto risco são desenvolvidos primeiro.

2.3.4.1 Vantagens e desvantagem do ASD

A filosofia ASD tem seus méritos independentemente do modelo de processos utilizado. A ênfase global da ASD está na dinâmica das equipes auto-organizadas, na colaboração interpessoal, na aprendizagem individual, que levam as equipes de projeto de software a uma probabilidade muito maior de sucesso (PRESSMAN, 2010). Além disso, algumas vantagens podem ser destacadas:

- Produz resultados com rapidez;
- Pode ser utilizado em projetos que necessitem de avaliação constante dos clientes;
- O planejamento pode ser adaptado em qualquer fase do projeto;

Apesar de algumas vantagens, tem alguns casos que o ASD não seria bem aplicado. As principais desvantagens que podem ser destacadas são:

- Tanto os desenvolvedores como os clientes devem estar comprometidos com o passo rápido do projeto. Se algum grupo não estiver acompanhando, o ASD estará destinado a falhar;
- Quando os riscos de projeto são altos, o ASD não pode ser adaptado;
- Projetos grandes exigem grandes equipes (para que haja efetivamente o aprendizado e compartilhamento de informações).

2.3.5 Scrum

O Scrum (nome derivado de uma atividade que ocorre durante um jogo de rugby) é umas das mais novas metodologias comparadas com outras metodologias. O primeiro Time de Scrum foi criado em 1993, por Jeff Sutherland. Jeff trabalhava em uma empresa de *software Easel Corporation* e aplicou o Scrum em um projeto com um maior grau de dificuldade e foram obtidos resultados excepcionais com essa nova forma de trabalhar. Segundo Sabbagh (SABBAGH, 2013), em 1995, Jeff apresentou o primeiro time de Scrum a Ken Shwaber, que era diretor executivo da empresa *Advanced Development Methods* e que já vinha trabalhando com ideias bem parecidas com o alvo em abordagens empíricas para o desenvolvimento de software. Com isso, Ken e Jeff trabalharam juntos para criar uma definição formal do Scrum, apresentada em um congresso de orientação a objetos em 1995. Em 2001, tanto Jeff Sutherland quanto Ken Shwaber foram adeptos do Manifesto ágil, na qual esse manifesto deu início ao chamado movimento ágil, na qual o Scrum faz parte.

Segundo os seus criadores, Scrum é um processo incremental e interativo para desenvolvimento de produtos ou para gerenciamento. Diferentemente da maioria das outras metodologias que tratam o desenvolvimento a partir das tarefas de programação explicitamente, o Scrum é orientado para o gerenciamento dos projetos de software. A metodologia em questão é indicada para projetos com curtos

prazos, requisitos que estão sempre sendo modificados e que são cruciais para o negócio (MEDEIROS, 2013). Segundo Sabbagh (SABBAGH, 2013):

> Scrum é um *framework* Ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Scrum é embasado no empirismo e utiliza uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto.

No Scrum, os projetos são divididos em ciclos (tipicamente mensais) chamados de *Sprints*. As aplicações a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*. No início de cada Sprint, faz-se um *Sprint Planning Meeting*, ou seja, uma reunião de planejamento na qual o *Product Owner* prioriza os itens do *Product Backlog* e a equipe seleciona as atividades que ela será capaz de implementar durante o Sprint que se inicia. As tarefas alocadas em um *Sprint* são transferidas do *Product Backlog* para o *Sprint Backlog*. A cada dia de uma *Sprint*, a equipe faz uma breve reunião (normalmente pela manhã), chamada *Daily Scrum*. O objetivo é disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia. Ao final de um *Sprint*, a equipe apresenta as aplicações implementadas em uma *Sprint Review* . Finalmente, faz-se uma *Sprint Retrospective* e a equipe parte para o planejamento do próximo Sprint. Assim reinicia-se o ciclo (DESENVOLVIMENTO, 2013).

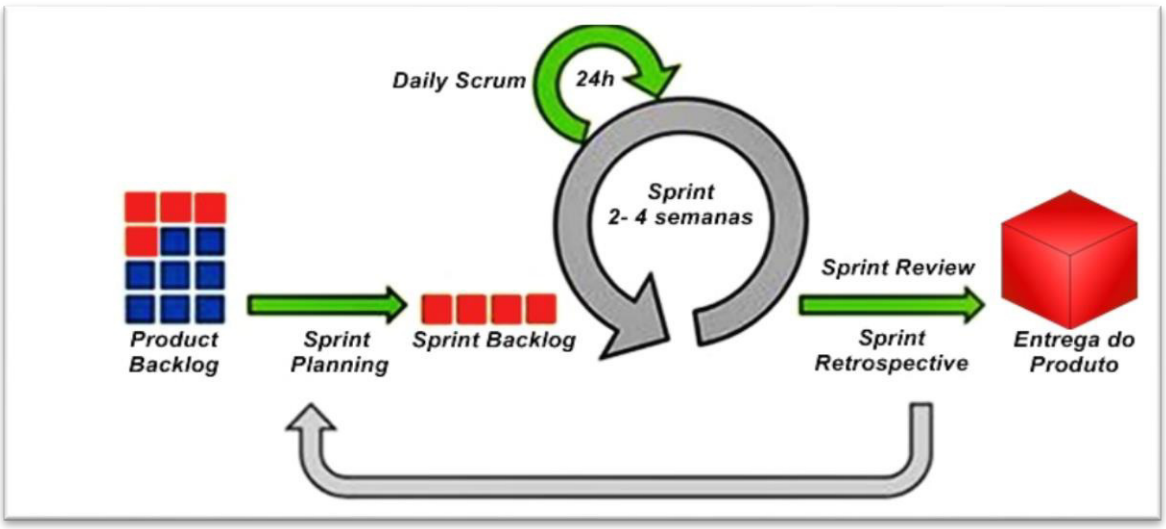

Figura 2.1- Funcionamento do Scrum.

Fonte: Adaptado de MEDEIROS, 2013.

2.3.5.1 Vantagens e desvantagens do Scrum

As características do SCRUM proporcionam vantagens consideráveis, tanto para as equipes de projetos quanto para os clientes. Dentre tantas vantagens se destacam:

- Os programadores se sentem muito mais motivados devido ao seu interesse de entregar o Sprint no prazo;
- Com o SCRUM, os projetos, que antes eram visualizadas de forma integral e integradas apenas pelos gerentes de projetos, passam a ter um *backlog* com todas as entregas acessíveis a todas as pessoas do time;
- Como a qualidade é mais importante do que o prazo de entrega, o produto apresenta uma diminuição significativa de erros (*bug*);
- Os programadores podem manejar as prioridades sem problemas, garantindo, assim, que os *sprints* que ainda não foram finalizados possam ser alterados sem problemas;

O Scrum é falho quando encontra resistência ao meio que ele é aplicado, pois suas regras são simples e seu foco é claro: na entrega de funcionalidades prontas ao final de cada *sprint*. Assim, o Scrum pode ter seus pontos fracos, destacando-se algumas desvantagens:

- Como a qualidade é mais importante do que o resultado, pode ser que os prazos não sejam estipulados de forma coerente, levando a um atraso do resultado final, deixando o cliente insatisfeito;
- A presença de papéis indefinidos nas funções presentes no projeto podem dar alguns problemas relacionados à comunicação interna e deixar os programadores confusos quanto as suas tarefas;
- A falta de documentações sobre o andamento do projeto pode ser um grande problema.

2.4 **Comparativos entre metodologias ágeis**

Os metodologias ágeis têm em comuns princípios e valores, mas existem suas particularidades que diferenciam uma da outra. Na tabela 2.1, faz-se uma comparação resumida a partir de algumas características dos metodologias citados anteriormente.

Tabela 2.1 - Comparação de características das metodologias ágeis XP, Scrum, FDD, ASD e Crystal

Características	XP	Scrum	FDD	ASD	Crystal
Abordagem	Incrementos interativos	Incrementos interativos	Iterativo	Iterativo	Incremental
Tempo de iteração	1-6 semanas	2-4 semanas	2 dias a 2 semanas	4-8 semanas	Depende do método
Tamanho do time	Pequenos < 20	Qualquer tamanho	Médios/ Grandes	Pequenos 5-9	Qualquer tamanho
Comunicação	Informal	Informal	Documentos	Informal	Informal
Tamanho do projeto	Pequenos	Qualquer tamanho	Projetos complexos	Pequenos	Qualquer tamanho
Envolvimento do cliente	Envolvido	Envolvido com o PO	Envolvido através de relatórios	Envolvido através das releases	Envolvido através das releases
Documentação	Básica	Básica	Formal	Básica	Básica
Especialidades	Programação em pares, 40 horas de trabalho, Refatoração...	Sprint, *product backlog*, Scrum master…	Diagramas UML	Ciclos de aprendizado	Metodologias adaptativos

A partir dos dados ou das informações apresentadas na tabela 2.1, pode-se inferir que os metodologias ágeis selecionados para fazerem parte deste estudo apresentam atividades bastante semelhantes em relação aos seus processos de desenvolvimento. Todos os metodologias têm suas vantagens e desvantagem como foi mostrado anteriormente, mas para o estudo desse trabalho foi escolhido a abordagem Scrum, por ter inúmeras vantagens e por ser uma metodologia de grande aceitação atualmente. Segundo Sabbagh (SABBAGH , 2013),a adoção do Scrum não significa que todos os problemas estão resolvidos, mas pode trazer diversos benefícios em comparação a outras formas de conduzir projetos, mas somente se bem utilizadas. O Scrum pode permitir reduzir os riscos de insucesso, entregar valor mais rápido desde cedo, e lidar com as inevitáveis mudanças de escopo, transformando-as em vantagem competitiva. Utilizado em diferentes mercados, desde projetos para produção de *softwares* comerciais, *sites* de Internet, de *softwares* embarcados, de aplicativos até ser usado em empresas de *marketing* e de desenvolvimento de hardware, por exemplo. Faz- se necessário lembrar que não existe uma solução única para todos os problemas. Scrum é um framework simples e pequeno e, assim, funciona bem em cada contexto se for utilizado em conjunto

com outras técnicas e práticas a serem experimentadas e adaptadas. Além disso, os benefícios incluem (SABBAGH, 2013):

- Entregas frequentes de retorno ao investimento dos clientes: Em projetos que utilizam metodologias tradicionais, apenas uma ou poucas entregas são realizadas ao final do projeto ou ao final de uma grande etapa. Com o Scrum, partes prontas do produto são geradas em ciclos curtos de desenvolvimento, que ocorrem um após outro. As partes entregues são as mais necessárias para seus clientes e usuários no momento da entrega e, por essa razão, serão utilizadas imediatamente.
- Redução dos riscos do projeto: Scrum visa à redução dos riscos de negócios do projeto pela colaboração com os clientes e demais partes interessadas durante todo o seu decorrer. Os riscos também são reduzidos com a produção de partes do produto prontas em cada ciclo de desenvolvimento, que serão imediatamente validadas pelo *feedback* dos clientes do projeto e demais partes interessadas.
- Maior qualidade no produto gerado: Dentro do ciclo de desenvolvimento, a validação é necessária para garantir a qualidade, ou seja, que a parte do produto esteja funcionando como se espera. Ao antecipar as validações do que é produzido, há muito mais chances dos problemas ou modificações serem detectadas cedo, garantindo assim mais qualidade no produto gerado.
- Mudanças utilizadas como vantagem competitiva: As necessidades de negócios e consequentes especificações e planos não são definidos detalhadamente no principio do projeto. Ao contrario, esses detalhes são descobertos e reavaliados ao longo do desenvolvimento, a partir do *feedback* frequente dos seus clientes e demais partes interessadas. Dessa forma, o produto é incrementalmente construído e modificado, e assim a mudança é utilizada para garantir que o produto entregue realmente signifique valor para esses clientes.
- Visibilidade do progresso do projeto: Diversas práticas e artefatos do Scrum visam garantir a visibilidade ou transparência do progresso do projeto para seus participantes e envolvidos, que incluem tanto os clientes e demais partes interessadas quanto o próprio time.

- Redução do desperdício: O desperdício é reduzido e evitado por meio da busca da simplicidade. A ideia central é que o time que usa Scrum produza e utilize apenas o que é necessário e suficiente.
- Aumento da produtividade: Diversos fatores potencializam a produtividade de times que utilizam Scrum. Entre esses fatores está, trabalho em equipe e a autonomia do time na realização desse trabalho, a existência de facilitação e de remoção de impedimentos, a melhoria continua dos processos de trabalho, um ritmo sustentável de trabalho e a maior motivação do time.

3 SCRUM

Segundo Schwaber (SCHWABER, 2010), as regras do Scrum agregam os papéis, eventos e artefatos, controlando as relações e interações entre eles. Cada elemento dentro do Scrum serve a um propósito específico e é fundamental para o uso e sucesso do Scrum.

3.1 **Papéis e Responsabilidades**

O Scrum possui três papéis: *Scrum Master*, *Product Owner* e o Time Scrum. A seguir serão detalhado cada papel e suas responsabilidades para do desenvolvimento de software.

3.1.1 Scrum *Master*[1]

Responsável por garantir o entendimento do processo do Scrum e por motivar e treinar a equipe. O papel do *Scrum Master* é de grande importância, portando uma função extremamente desafiadora durante a fase inicial de implantação. Seu papel principal é evitar impedimentos, entender as dificuldades do time e manter uma boa comunicação, seja dentro do time ou da própria empresa. As tarefas adicionais são (VARASCHIN, 2009):

- Manter o time funcional e produtivo;
- Proteger o time de interferência externa.

Talvez seja o papel com maior grau de adaptação no inicio da aplicação do Scrum.

3.1.2 *Product Owner* [2]

Segundo Schwaber (2010), o *Product Owner* é responsável pela priorização de requisitos que devem ser implementados, ou seja, responsável pelo gerenciamento do *Product Backlog* (lista de funcionalidades que serão implementadas no projeto). O *Product Owner* é o cliente de um time Scrum, ele é

[1] Mestre do Scrum
[2] Dono do Produto

responsável pelo retorno do investimento de um produto. Entre suas tarefas estão (VARASCHIN, 2009):

- Determinar todas as funcionalidades que o produto deve conter. As propostas de histórias podem ser feitas por qualquer pessoa, mas a inclusão destas no *Product Backlog* é de controle do *Product Owner* ;
- Priorizar as funcionalidades que devem ser incluídas a cada *Sprint* de acordo com o valor de negócio;
- Analisar as entregas do time, aceitando ou rejeitando.

3.1.3 Time Scrum

Responsável pela concretização e implementação das funcionalidades. O time Scrum deve ser multidisciplinar e auto-organizado, devendo ser pequeno com no máximo 10 a 15 participantes (SCHWABER, 2010).

Segundo Varaschin (VARASCHIN, 2009), o conjunto do conhecimento dos seus participantes deve englobar todas as características necessárias para a implementação do projeto a que se propõe. Embora o time Scrum seja responsável por atingir os objetivos do *Sprint*, ele deve ser também autônomo e ter controle sobre o seu processo de desenvolvimento, sendo de sua responsabilidade ao final de cada *Sprint* mostrar os resultados do trabalho para o cliente.

3.2 **Eventos do *Sprint*[3]**

De acordo com Schwaber (SCHWABER, 2010), o *Sprint* é definido como um ciclo de desenvolvimento curto em que o time foca em atingir o objetivo proposto pelo *Product Owner*. Durante este período o time deve ter total autoridade para seu gerenciamento, não devendo sofrer interferências externas do *Product Owner* ou mesmo de outras pessoas. Não se pode garantir o sucesso de um projeto com o uso do Scrum deixando de incluir qualquer um dos seus eventos que são a *Sprint*, a *Sprint Planning*, a *Daily Scrum,* a *Sprint Review* e a *Sprint Retrospective*.

[3] Iteração

3.2.*1 Sprint Planning* [4]

Dentro de um Sprint, a primeira atividade a ser realizada é uma reunião denominada *Sprint Planning*. Durante o *Sprint Planning* é efetuada a seleção das histórias que serão implementadas durante aquele ciclo. É importante frisar que a tarefa do *Sprint Planning* já supõe que exista um *Product Backlog*, com suas histórias, valores de negócio e complexidade. O *Sprint Planning* geralmente é feito em um dia, mas de acordo com o tamanho do Sprint este tempo pode variar. O *Sprint Planning* tem dois momentos, o primeiro, onde o time juntamente com o *Product Owner* e o Scrum *Master* planejam as funcionalidades que serão contruídas durante o Sprint. A primeira etapa do Sprint Planning é (VARASCHIN, 2009):

- *Product Owner* prioriza os itens em ordem de importância;
- Time seleciona os itens que considera possíveis de desenvolvimento dentro do Sprint.

Segundo Schwaber (SCHWABER, 2010), a primeira etapa de reunião a tarefa do time e o *Product Owner* é analisar as histórias existentes no *Product Backlog* e definir segundo o *Product Owner* aqueles que possuem maior valor para o produto. Na maioria das vezes é avaliada a relação entre o valor de negócio da história e a complexidade do desenvolvimento ditada pelo time. Neste momento o *Product Owner* tem a autoridade para definir as histórias que serão encaixadas no próximo Sprint. É necessário salientar que a responsabilidade pela qualidade do produto é do time, e o mesmo tem a possibilidade de incluir histórias no *Product Backlog* relacionadas com a qualidade, arquitetura e outros. É responsabilidade do time também, mostrar ao *Product Owner* que determinadas histórias criadas por ele impactam diretamente o negócio e têm grande valor, devendo, portanto ser desenvolvidas. Ao final desta etapa as histórias devem ter sido selecionadas e devem se encaixar na velocidade do time para um *Sprint* chegando então o momento da segunda etapa que é o detalhamento das tarefas (SCHWABER, 2010).

Na segunda parte do *Sprint Planning* o time tem como meta dividir as histórias em tarefas que serão cumpridas durante o *Sprint*. O objetivo desta segunda etapa é bem claro, o time deve realizar um detalhamento de todas as necessidades para desenvolver uma história e verificar se a estimativa inicial dada está de acordo com

[4] Reunião de Planejamento da Sprint

todas as atividades existentes. Para isto é necessário que eles façam o seguinte (VARASCHIN, 2009):

- Para cada item o time detalha as atividades e estima o tempo necessário para o desenvolvimento de cada uma delas. As tarefas devem requerer no máximo um dia para que possam ser acompanhadas no *Daily Meeting*. Este modelo pode ser ajustado pela empresa. Existem modelos em que as tarefas são definidas em número de horas e seu acompanhamento é realizado desta forma.
- Ao finalizar as estimativas é revisto se o tempo do *Sprint* é suficiente para a realização de todas as tarefas. Caso o time detecte que após o detalhamento existe mais trabalho do que o *Sprint* prevê, é necessário dialogar com o *Product Owner* para rever as histórias que compõem o *Sprint*. Do mesmo modo se sobrar tempo deve-se prever a possibilidade de adicionar mais histórias do *Product Backlog*. Não são alocados recursos para as tarefas, atividade esta que é diária e realizada nos *Daily Meetings* (VARASCHIN, 2009).

3.2.*4 Daily Scrum*[5]

O Daily Scrum é uma reunião rápida que deve ter o conceito de tempo fechado, realizada todo dia, em pé durante a qual os membros do time explicam entre si o que foi feito desde a última reunião, o que será feito até a próxima e quais são os impedimentos que surgiram. As principais características do Daily Scrum (VARASCHIN, 2009):

- As reuniões devem começar no horário marcado. Frequentemente o time define penalidades por atrasos.
- Todos podem assistir as reuniões, mas apenas o time tem o direito de falar durante a reunião. Caso seja necessário eles podem solicitar informações para o *Scrum Master* ou *Product Owner.*
- Deve durar até 15 minutos. Em times maiores esta reunião pode se estender um pouco mais. Caso o time esteja muito grande avalie a possibilidade da criação de dois times.

[5] Reunião Diária

- Os participantes devem permanecer em pé
- Deve ser realizada sempre no mesmo local na mesma hora do dia.

As perguntas que são respondidas durante a reunião são (SCHWABER, 2010):

- O que você fez desde ontem?
- O que você planeja fazer até amanhã?
- Existiu algum impedimento?

Segundo Varaschin (VARASCHIN, 2009), o objetivo desta reunião, além de acompanhar o desenvolvimento do projeto e distribuir as tarefas é criar a pressão entre os membros do time para a entrega do objetivo. A cobrança é do time para o time, não existem chefes e subordinados, apenas a equipe em busca do objetivo da entrega.

3.2.5 Sprint Review [6]

O *Sprint Review* acontece no final de cada Sprint. Durante esta reunião o time mostra o resultado do seu trabalho para o *Product Owner*. Os itens avaliados durante o *Sprint Review* foram definidos como requisitos durante o *Sprint Planning* através de histórias de aceitação (VARASCHIN, 2009). Como boa prática o *Sprint* deve ser acompanhado pelo *Product Owner* que verifica o andamento do desenvolvimento das histórias. Durante o *review* o *Product Owner* já possui noção de praticamente todo o andamento do *Sprint* e quais são as histórias que foram completadas. Deve ser realizado durante esta etapa (VARASCHIN, 2009):

- A equipe irá demonstrar cada uma das histórias desenvolvidas, sendo cada uma delas avaliada de acordo com o os casos de aceitação existentes;
- O *Product Owner* realizará os testes de cada uma das histórias para verificar se as condições estabelecidas foram satisfeitas;
- No caso de histórias não finalizadas a equipe deve discutir com o *Product Owner* os motivos do não atingimento do objetivo.

O ideal é que a validação seja realizada em ambiente de produção dado que o Scrum prega software funcionando como final da etapa, mas isto pode variar de acordo com a empresa, sendo possível em alguns casos validar em um ambiente de pré-produção.

[6] Revisão da Sprint

3.2.6 *Sprint Retrospective*[7]

A reunião chamada de *Sprint Retrospective* tem como principal foco o time. Ela é realizada sempre ao final de cada Sprint e deve ser utilizada para a correção dos problemas detectados pelo time no processo de desenvolvimento, bem como para a implementação de melhorias a partir de sugestões. As melhorias no processo de desenvolvimento são contínuas e podem demorar até um ano. Por ser um processo iterativo todas as mudanças devem ser gradativas, fazendo com que cada nível novo seja implementado, avaliado, testada e só posteriormente nova complexidade seja adicionada (SCHWABER, 2010).

Segundo Varaschin (VARASCHIN, 2009), outro item importante é manter o foco nos novos metodologias, pois existe a tendência dos times a retornarem ao uso de velhas práticas. Sem a melhoria contínua existe a possibilidade dos times perderem desempenho, já que não existem novos desafios. Participam na retrospectiva o time e o *Scrum Master*, para os demais é necessário convite por parte do time. Este é o momento em que o Scrum *Master* tem o trabalho de ajudar o time a encontrar seus verdadeiros problemas.

3.3 Artefatos

Os artefatos do Scrum são a representação do trabalho que deve ser feito para que o "produto" exista e o projeto se materialize, ou seja, são referências para realizar planejamento, execução e controle. Os artefatos básicos do Scrum são o *Product Backlog* e *Sprint Backlog.*

3.3.1 Product Backlog[8]

O *Product Backlog* é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto (SCHWABER, 2010). É importante ressaltar que esta lista está em constante evolução, seja porque novos requisitos de negócio surgem, seja porque tecnicamente faz se necessária manutenção no software ou um refatoramento.

[7] Retrospectiva da Sprint
[8] Backlog do Produto

É função do *Product Owner* manter a lista com suas prioridades e mostrar para o time quais são as perspectivas para o projeto. Segundo Varaschin (2009), um dos maiores desafios do Scrum é manter um *Product Backlog* organizado, seja com relação às prioridades ou com relação à complexidade necessária para o cumprimento das histórias. Cada história existente no *Product Backlog* deve ter um valor de negócio e uma complexidade associada. O primeiro é de responsabilidade do *Product Owner* e o segundo do time, que deve ser sempre solicitado a reavaliar as estimativas constantes no *Product Backlog*.

3.3.2 Sprint Backlog [9]

O *Sprint Backlog* é gerado a partir das histórias retiradas do *Product Backlog* e que serão implementadas durante o *Sprint* (SCHWABER, 2010).

Segundo Varaschin (VARASCHIN, 2009), durante o *Sprint Planning* as histórias aceitas pela equipe são desmembradas em tarefas menores, de no máximo um dia, para que possam ser acompanhadas pela equipe durante o *Sprint*, além disto, as histórias de aceitação devem ser redigidas e registradas para o aceite do *Sprint*, gerando novas tarefas. Este trabalho gera o *Sprint Backlog*, dele fazem parte todas as tarefas necessárias para alcançar o objetivo do *Sprint*. É importante ressaltar que durante o *Sprint,* novas tarefas podem surgir, ou porque foram esquecidas ou porque determinada tarefa terá que ser dividida.

Durante o *Sprint*, caso todas as tarefas do *Sprint Backlog* já tenham sido realizadas, é possível que o time, com o consentimento do *Product Owner*, adicione mais histórias do *Product Backlog* ao *Sprint Backlog*. Neste caso o time assume o compromisso de finalizar aquelas histórias até o final do *Sprint* (VARASCHIN, 2009).

3.4 Considerações finais

Em alguns casos, nem todos os conceitos utilizados em uma metodologia podem ser implementados no ambiente da empresa, e outros conceitos considerados importantes para a empresa podem não ser contemplados pela metodologia escolhida. A adaptatividade é uma das características mais marcantes nas metodologias ágeis como o Scrum. Os papéis, eventos e artefatos do Scrum,

[9] Backlog da Sprint

podem e devem ser adaptados conforme a estruturação do ambiente, sem perder suas fundações Ágeis. Normalmente a equipe do Scrum deve educar e negociar bastante para que tudo ocorra bem. O Scrum é apenas um meio para o fim e, por essa razão, as pessoas envolvidas devem ser pragmáticas em vez de dogmáticas (PHAM, 2011).No próximo capitulo será mostrado uma proposta de como poderá ser adaptada, sem perder os conceitos principais, a abordagem Scrum.

4 ESTUDO DE CASO - UMA PROPOSTA DO SCRUM PARA O NÚCLEO DE TECNOLOGIA DA INFORMAÇÃO DA UFMA (NTI - UFMA)

O propósito deste estudo de caso é adquirir conhecimento a fim de contribuir para a evolução dos conhecimentos relacionados aos processos de adaptação de metodologias ágeis, especificamente o Scrum, a um ambiente em que se tem muita demanda de projetos e tarefas, mas não se tem uma forma de desenvolvimento de software bem definida. Em particular, espera-se contribuir com novas informações sobre a viabilidade da adaptação do Scrum a esse espaço para obtenção de ótimos resultados.

4.1 Ambiente do estudo de caso

O ambiente escolhido para esse estudo de caso foi o Núcleo de Tecnologia da Informação da Universidade Federal do Maranhão (NTI – UFMA). A principal responsabilidade do NTI é “assessorar e dar suporte aos órgãos envolvidos nas atividades de ensino, pesquisa, extensão e administração, no que se refere à gestão da tecnologia da informação e comunicação” (NTI, s/d). E tem como visão, ser referência em soluções de tecnologia da Informação e comunicação. O NTI tem como valores (NTI, s/d):

- Efetividade: Atua direcionada para os resultados que assegurem o cumprimento da missão.
- Transparência: Cultivar o desejo de informar através de um processo de comunicação eficiente.
- Integração: Promover um ambiente propício à construção coletiva de soluções e do conhecimento.
- Inovação: Buscar soluções inovadoras para melhor prover as informações e os recursos de TIC da UFMA (Tecnologias de Informação e Comunicação).
- Responsabilidade Corporativa: Assegurar a máxima sustentabilidade dos negócios, incorporando considerações de ordem ética, social e ambiental em todos os processos e relacionamentos.

O NTI tem como estrutura organizacional: a Diretoria, Gerência de Desenvolvimento, Gerência de Redes, Gerência Administrativa, Suporte, Comitê de Planejamento e Gestão TIC. Graficamente se estrutura conforme mostra a figura 4.1.

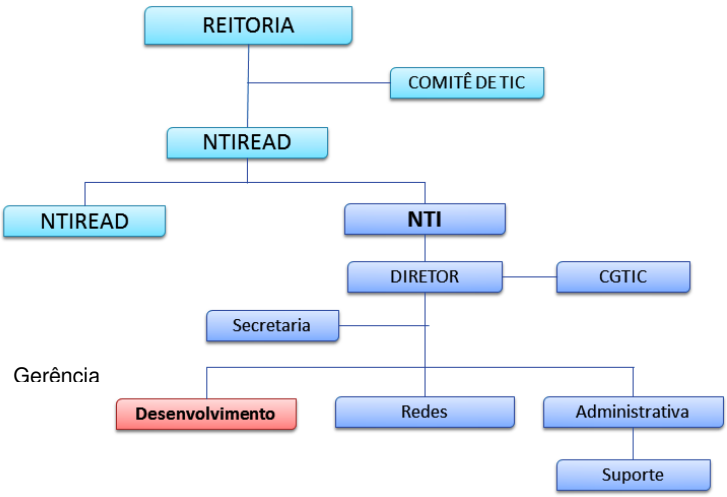

Figura 4.1 – Estrutura Organizacional do NTI

A proposta para aplicação do Scrum deste trabalho foi feita para setor de Desenvolvimento, onde o mesmo é responsável pela criação, treinamento, suporte e/ou manutenção de portais da reitoria e dos sistemas coorporativos como:

- SIGAA - Sistema Integrado de Gestão de Atividades Acadêmicas;
- SIGRH - Sistema Integrado de Gestão de Recursos Humanos;
- SIPAC - Sistema Integrado de Patrimônio, Administração e Contratos.

Já foram utilizados vários metodologias de desenvolvimento de software no setor de desenvolvimento, mas nenhum trouxe resultados totalmente satisfatórios. Algumas ferramentas e metodologias já foram adaptados como RUP[10], Kaban[11], algumas atividades do XP, Scrum, entre outros.

Atualmente, para organização e controle no desenvolvimento de software, o setor foi dividido em três equipes: requisitos, desenvolvimento e suporte, onde cada equipe tem um líder, que na teoria tem o papel de supervisionar as tarefas juntamente com o gerente do setor de desenvolvimento. Diariamente, todos os membros preenche um relatório feito no google docs, relatando o que foi produzido, ou até mesmo o que não foi produzido, compartilhado com os líderes e o gerente.

[10] *Rational Unified Process (*RUP) é um framework de processo de engenharia de software que fornece um conjunto de práticas para desenvolvimento de software e gerência de projetos.

[11] Método ágil de desenvolvimento de software baseado nas práticas Lean, e que tem como objetivo otimizar o processo de desenvolvimento de software.

Em um tempo não determinado, o gerente do desenvolvimento, junto com os líderes de cada equipe, analisa o que esta sendo feito, removendo eventuais obstáculos.

Para o gerenciamento dos projetos que estão em andamento e os concluídos, utilizam-se o *Redmine*. *Redmine* é um software livre, gerenciador de projetos baseados na web e ferramenta de gerenciamento. Ele contém calendário e gráficos de *Gantt*[12] para ajudar na representação visual dos projetos e prazos de entrega, podendo também trabalhar com múltiplos projetos.

Cada membro da equipe dispõe uma conta cadastrada onde tem acesso aos projetos que possui permissão para acessar. Na tarefa solicitada ao desenvolvedor, este poderá atualizar (detalhando o que foi feito e como foi feito pra desenvolver aquilo que foi proposto) e assim que concluir mudar o status de “em andamento” para “fechada”, como mostra na figura 4.2.

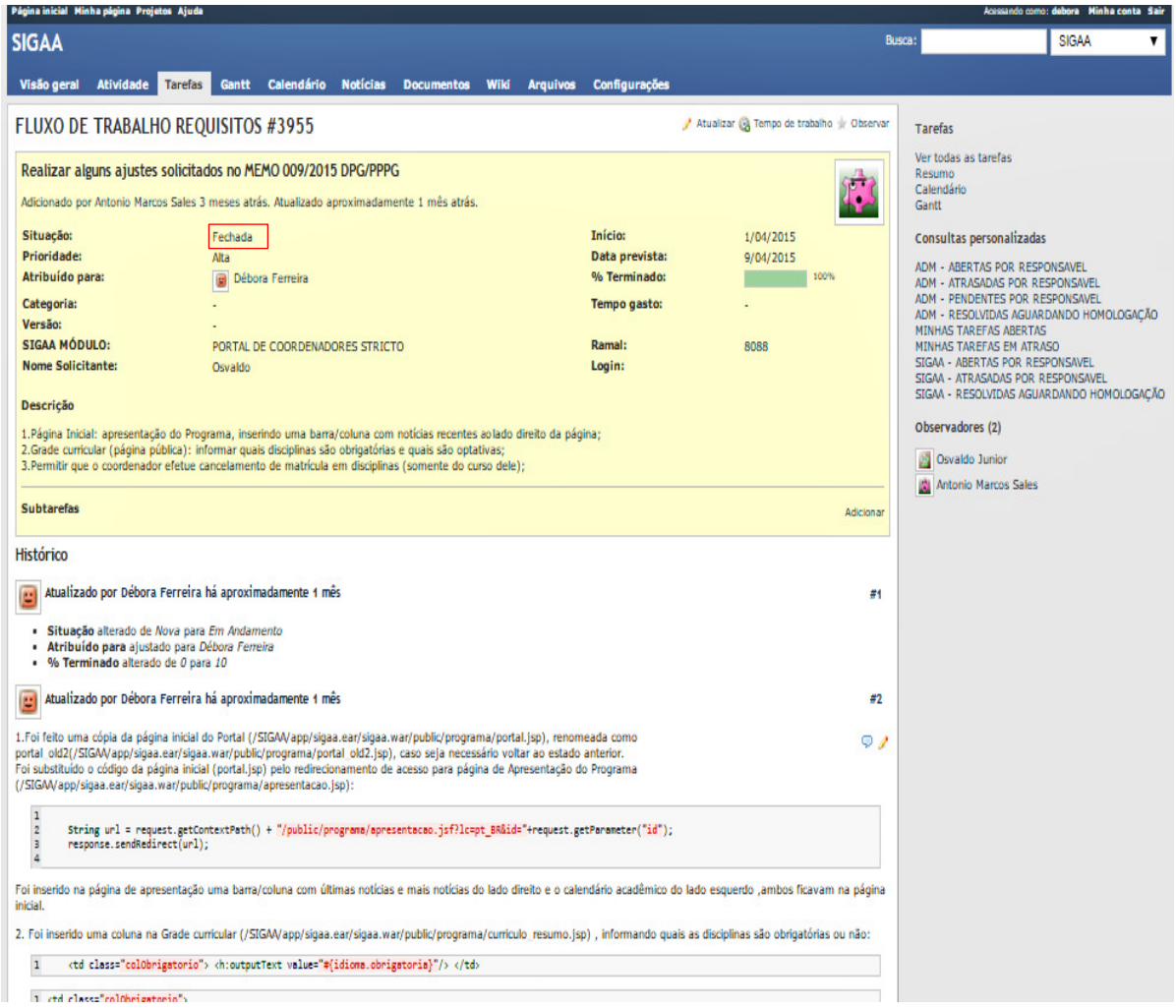

Figura 4.2 - Página de Tarefas do *Redmine*

[12] Gráfico usado para ilustrar o avanço das diferentes etapas de um projeto.

4.2 Definição do estudo de caso

Para coletar os dados para a análise do estudo de caso, utilizou-se a técnica de entrevistas e o contato direto com ambiente. Levando em consideração toda análise feita durante alguns meses, pode- se concluir:

- Nenhum método aplicado no NTI teve tanta durabilidade. Alguns metodologias trouxeram bons resultados, mas por vários fatores, como diminuição, mudanças no quadro de funcionários e falta de comprometimento, fizeram com que certos metodologias deixasse de ser executado.
- A forma como é estruturado o gerenciamento de software, de modo geral, não está trazendo resultados satisfatórios.
- A falta de comprometimento de alguns membros da equipe de desenvolvimento é o principal fator para que não tenha um bom desempenho nos resultados finais.
- Sobrecarga ao gerente de desenvolvimento, em ficar supervisionando, sendo cobrado e cobrando a equipe de desenvolvimento em todas as tarefas e até mesmo intervindo em algumas tarefas para agilizar a entrega.
- Por falta de uma organização, as demandas de projetos e tarefas se acumulam.
- Não há envolvimento do cliente em tempo integral.
- Não são realizadas reuniões rotineiras ou, periodicamente, determinadas.
- Falta de organização e comunicação entre as pessoas envolvidas.
- Falta de definição de prioridades às tarefas e projetos a serem desenvolvidas.

Como já foi dito, anos atrás utilizaram a metodologia Scrum, mas depois de algum tempo não foi apresentando resultados satisfatórios. Os motivos foram: por não aplicar corretamente algumas atividades do Scrum, por queda no número de funcionários, falta de treinamento e comprometimento dos membros. Mesmo com esses motivos, percebeu-se que a metodologia Scrum foi bem aceito e quando foi aplicada de forma correta, geraram bons resultados.

Analisando todos os problemas identificados e visando a necessidade da melhoria no gerenciamento dos projetos, desenvolveu-se uma proposta para adaptação da metodologia ágil Scrum no NTI. Para elaborar o cenário futuro, descrevem-se a seguir uma proposta à metodologia aplicada e o respectivo reflexo de suas funções, observando-se as principais mudanças, além de especificar as

etapas requeridas pelo desenvolvimento. Definem-se os papéis de cada responsável pela elaboração das customizações, desde o problema constatado até o produto ou funcionalidade desenvolvida.

4.3 Proposta do Scrum para o NTI

Aplicar Scrum traz várias mudanças, principalmente culturais na empresa. Sem o comprometimento de todos os envolvidos, com certeza, os resultados não serão satisfatórios. O principal problema do NTI nas aplicações dos metodologias ágeis, como já mencionado, é a falta de comprometimento dos envolvidos.

Para isso, propõe-se, inicialmente, um treinamento para todos os colaboradores, tanto aqueles que já tinham utilizado o Scrum anteriormente, como para aqueles membros novos que não conhecem a fundo a metodologia, para que todos possam compreender as atividades a serem desempenhadas na nova adaptação da metodologia de gerência de projeto e assim nivelar o conhecimento adquirido. No treinamento terão que ser repassados os conhecimentos a cerca do ciclo do Scrum e mostrando, em detalhes, cada evento a ser executado com suas adaptações da metodologia ágil ao ambiente, as vantagens e facilidades proporcionadas pelo Scrum e o comprometimento em realizar com pontualidade as atividades, para obter bons resultados, trazendo motivações para que possa firmar o comprometimento dos envolvidos.

4.3.1 Definição dos papéis

Os responsáveis pela manutenção de todo o ciclo de desenvolvimento do Scrum junto à Instituição têm de estar alinhados àquilo que será desenvolvido, porquanto, não podem fugir do objetivo principal que será apresentado durante a definição da nova adaptação. Conforme conceitos apresentados da metodologia ágil Scrum, este trabalha com três perfis de pessoas que são responsáveis pela realização de todo o processo. Na aplicação do estudo de caso será usado os três papéis, conforme segue-se descrito:

- *Scrum Master*: No caso deste estudo, foram definidos como *Scrum Master* o gerente de Desenvolvimento e um membro de cada equipe para auxiliar, já que um dos problemas do NTI é a sobrecarga para o gerente de desenvolvimento, onde fica responsável pela supervisão de todos os projetos

que estão sendo desenvolvidos, e às vezes, pelo atraso de certas tarefas, faz-se necessário efetuar um acompanhamento mais direto para não ter grandes atrasos. O *Scrum Master* deve ter a preocupação em manter a disciplina do processo, as estruturas, mais os objetivos das reuniões e o time ciente das regras básicas como o comprometimento com as entregas do *Sprint*. Sem isto, a tendência dos times é relaxar com relação ao processo e voltar a adotar velhas práticas.

- *Product Owner:* Pode-se notar que, por motivos de disponibilidade, não há envolvimento do cliente (papel que seria equivalente ao *Product Owner*) em tempo integral no NTI. Esse papel sofrerá adaptações na aplicação da metodologia Scrum no NTI, devido a essa falta de comunicação direta com o cliente. Durante a *Sprint,* a comunicação entre a equipe e o *Product Owner* poderá ser realizada através do fórum, e-mail ou ferramentas de comunicação, seja para o esclarecimento de dúvidas ou alinhamento do andamento da *Sprint* e poderá ser escolhido um membro da equipe, de preferência da área de requisitos, para fazer a interface com o cliente (*Product Owner*). Em alguns casos, faz- se necessário que o time tenha uma postura pró-ativo para tomar algumas decisões sem envolver o *Product Owner*.
- Time Scrum: Para definir a equipe do ciclo Scrum no setor de desenvolvimento no NTI, os membros poderão ser divididos em times conforme a demanda de projetos e tarefas. Atualmente, tem-se 15 pessoas mais o gerente. Sendo assim, conforme a demanda de tarefas, pode-se dividir em 3 times com 5 pessoas. Mas essa divisão terá que ser bem arquitetada para que nenhum time possa ficar sobrecarregado e assim afetar os resultados. Muito importante que todos dentro do time estejam dispostos a colaborar; o trabalho individual não deve impactar na entrega do time. Determinados membros do time podem ainda estar acostumados ao acompanhamento direto do gerente, assim tendem a ter uma entrega menor no novo modelo. Se o time não cobra o membro e opta por uma posição de ignorar, retirando deste indivíduo atividades importantes, o time perde em produtividade e terá em pouco tempo um membro sem participação e entrega. Neste caso, o Scrum Master deve estar atento para este tipo de

comportamento que deve ser rapidamente identificando e corrigir tudo aquilo que está impedindo o membro de realizar determinada tarefa. A colaboração dentro do time é fundamental e quando uma das peças começa a não entregar, o time deve ficar preocupado e ser o primeiro a solicitar alguma providencia para que não afete o desempenho do time.

Todos os envolvidos nesse processo têm importante participação para atingir o resultado esperado.

4.3.2 Desenvolvimento do processo

No ciclo da metodologia ágil Scrum, no contexto da Instituição,depois de feito o treinamento e definido os papéis, o projeto será dividido em ciclos periódicos chamados *Sprints*. Cada *Sprint*, geralmente, varia-se entre uma semana a um mês, representa um volume de esforço pré-definido no qual um grupo de atividades deverá ser executado e seu produto final será uma nova funcionalidade ou customização.

No começo de cada *Sprint* deve ser realizada uma *Sprint Planning* (reunião de planejamento), feita com todos os papéis (Scrum *Master* da equipe, o time e o *Product Owner*) na qual é definido o *Product Backlog, (a*s prioridades do *Product Backlog* (*Sprint Backlog*) e atribuição das tarefas aos componentes da equipe.

O *Product Backlog* deve está facilmente acessível e visível para todo o time em qualquer momento. Para isso, pode-se usar a ferramenta que já utilizam atualmente, o *Redmine*. A lista será atualizada após cada *Sprint*, durante a *Sprint Review* (reunião de revisão), com suas prioridades, situação (andamento ou pendente), para quem a tarefa foi atribuída e o prazo de entrega. A lista de tarefas que tem prioridades classificadas como “Alta” seria o *Sprint Backlog*. As outras tarefas (*Product Backlog*) ficariam aguardando para serem desenvolvidas nos próximos *Sprint*. Na figura 4.3, mostra um exemplo de como ficaria no *Redmine* depois da *Sprint Planning*, em uma equipe formada por 5 pessoas, responsáveis pela implementação de algumas tarefas no SIGAA.

#	Tipo	Tarefa pai	Situação	Prioridade	Título	Atribuído para	Início	Data prevista
1895	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Simular Migração PROEB - Curso, Disciplina, etc	Ticiane Aguiar	24/06/2015	08/07/2015
4104	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Carga de alunos novos do COLUN	Bruno Alberth	24/06/2015	08/07/2015
4113	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Desenvolver funcionalidade para conclusão de alunos do COLUN	Bruno Alberth	23/06/2015	08/07/2015
1939	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Elaboração de relatórios estatísticos	Marcio Regis Fraga	24/06/2015	08/07/2015
3172	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Criar relatório que liste as disciplinas de determinado bimestre que não possuem notas para ensino médio e fundamental	Walter Araújo	24/06/2015	08/07/2015
3119	FLUXO DE TRABALHO REQUISITOS		Em Andamento	Alta	Criar sequências para as respectivas tabelas solicitacao_turma e nota_unidade e sincronizar os seus valores.	Marcos Mendes Lauande	24/06/2015	08/07/2015
3382	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	Verificar a funcionalidade das opções de 'Cancelamento de aluno', 'Estornar afastamento' e 'Cancelar aluno por reprovações' no portal do nível Técnico			
3695	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	Acesso por auto cadastro - Portal Saúde Baseada em Evidências			
3785	FLUXO DE TRABALHO REQUISITOS		Nova	Normal	Adicionar parâmetro ao módulo Lato Sensu que configure um limite mínimo de 360h para propostas de curso Lato Sensu			
3935	FLUXO DE TRABALHO REQUISITOS		Nova	Normal	Alterações no questionário da Turma Virtual do SIGAA			
3936	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	Mostrar pontuação em vez de porcentagem para questionários com questões de pontos diferentes na Turma Virtual do SIGAA			
3947	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	Alterações no Fórum da Turma Virtual do SIGAA			
3405	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	O Problema do SIGAA (módulo Biblioteca - Aba Aquisições - Criar Assinaturas) (problema da geração de códigos)			
3223	FLUXO DE TRABALHO REQUISITOS		Pendente	Normal	Fazer alterações no módulo de diplomas conforme solicitação da DIRED			

Figura 4.3 - *Sprint Backlog* no *Redmine*

Para não sobrecarregar tanto a equipe, a *Daily Scrum* (que a princípio recomendaria se fazer diariamente), será feita em dias alternados, de preferência a tarde, após o horário almoço (percebe-se ser um horário que todos estão presentes), de no máximo 15 minutos. O objetivo da *Daily Scrum*, no ambiente, é disseminar conhecimento sobre as atividades realizadas no dia anterior, identificar riscos e problemas e priorizar as tarefas no dia corrente.

Conforme o exemplo da figura 4.3, seria necessário 2 a 3 *sprints*, dependendo do desempenho inicial da equipe. A cada término de *Sprint,* a equipe apresentaria as funcionalidades implementadas na *Sprint Review* (Reunião de Revisão), e os membros do time planejam o próximo *Sprint*, reiniciando todo o ciclo novamente.

Após esta reunião, faz-se uma *Sprint Retrospective* (reunião de retrospectiva) para demonstrar as lições aprendidas, sendo possível identificar os principais desafios enfrentados pela equipe e poderá ser conduzida por um integrante do time que ficará responsável por coletar os pontos fortes, fracos e sugestões de melhoria de cada participante.

Para apoio ao *Sprint*, podem-se fazer o uso do *Redmine* para o acompanhamento diário das atividades e controle sobre as tarefas e projetos

executados no *Sprint*. Faz-se necessário lembrar que a atualização no *Redmine* das tarefas é de responsabilidade do time, mantê-lo com as informações corretas, para que possa ser acompanhado diariamente por todos e assim melhorar os resultados finais.

Pode-se verificar, com base na proposta, que a integração entre os membros do processo é essencial, pois, ambos precisam estar a par de todas as situações relativas às modificações necessárias, bem como, em razão dos problemas que possam impedir a elaboração dessas modificações. Por tais motivos, constata-se que para a Instituição viabilizar a aplicação de uma metodologia ágil para gerenciar o processo de desenvolvimento de software, os envolvidos têm que se aplicar, seguir os padrões e conceitos, ter um comprometimento com a aplicação dessa metodologia. Além disso, deve-se ter a consciência de que a qualificação do processo pode aproximar o cliente e oportunizar novas maneiras de relacionamento e, o mais importante, aumentar a competitividade e obter ótimos resultados.

4.3.3 Principais mudanças

Com a aplicação da metodologia Scrum para qualificar e aperfeiçoar o processo de desenvolvimento de software, seguem-se elencadas na tabela 4.1, as principais mudanças sugeridas com base nos problemas constatados.

Tabela 4.1 – Principais mudanças com a adaptação do Scrum.

Problemas	Solução
Nenhum método aplicado ao NTI teve tanta durabilidade e a forma como é estruturado o gerenciamento de software, atualmente, não está trazendo resultados satisfatórios.	Aplicar a proposta descrita de forma pontual e com comprometimento irá trazer resultados satisfatórios.
Falta de comprometimento de alguns membros da equipe.	Propõe-se no início da aplicação do método fazer treinamentos e estimular a equipe a ter comprometimento.

Sobrecarga ao gerente de desenvolvimento.	Com a criação do Scrum Master em cada equipe, o gerente ficará menos sobrecarregado.
Expiração de prazos de entrega.	Dividir em várias equipes conforme a demanda e aplicando a *Daily Scrum*, identificando-se os riscos e problemas constantes, faz-se necessário que se tenha agilidade na entrega das tarefas.
Não há envolvimento do cliente em tempo integral.	- A comunicação poderá ser realizada através de fórum, e-mail ou ferramentas de comunicação. - Um membro do time fica responsável em manter a interface com o cliente. - O time em alguns momentos deve tomar algumas decisões sem envolver o *Product Owner*.
Falta de organização e comunicação entre as pessoas envolvidas.	Integrar as pessoas envolvidas, aplicando-se as definições dos *sprints* e a aplicação de reuniões diárias (*Daily Scrum*).
Falta de definição de prioridades às tarefas e projetos a serem desenvolvidas.	Definir prioridades do *Product Backlog* resultando na *Sprint Backlog.*

Ao utilizar os conceitos e metodologias da metodologia ágil Scrum é possível qualificar o processo de manutenção do sistema e solucionar alguns dos problemas constatados. Ressalte-se que, no caso deste estudo, um dos pontos a ser desenvolvido visa à motivação e o treinamento dos profissionais envolvidos, visto que a motivação da equipe causa impactos em todo o processo. Faz-se importante ressaltar que a necessidade de se utilizar o Scrum em partes é garantir a organização, construção e implementação do desenvolvimento de software, e fazer com que todos os itens mencionados anteriormente possam qualificar o processo,

gerar menor custo e melhor fluxo de trabalho, a fim de facilitar todo o processo. No entanto, com o uso da metodologia proposta é exigida maior participação de todos os membros para garantir a construção da solução desejada.

Com a integração e a comunicação propostas, o Scrum auxiliará na disseminação do conhecimento construído junto ao time. Além disso, as equipes que fazem parte do ciclo da metodologia Scrum tendem a ficar motivadas e, com o passar do tempo, criam laços com o produto e, por conseguinte, imprimem qualidade e eficiência na finalização. A proposta de implantação da metodologia ágil Scrum pode gerar choque cultural ou conflitos, porquanto, cumpre aos Scrum Masters mediar esses conflitos e flexibilizar, na medida do possível, os novos processos de trabalho.

5 CONCLUSÃO

Neste trabalho foi apresentada uma proposta de adaptação da abordagem Scrum, com o intuito de contribuir para a evolução dos conhecimentos relacionados aos processos de adaptação de metodologias ágeis, especificamente o Scrum e viabilizar a melhoria do gerenciamento de desenvolvimento de software no Núcleo de Tecnologia da Informação (NTI) da Universidade Federal do Maranhão.

Baseado no estudo das metodologias ágeis realizado pode-se concluir que a decisão de adotar metodologias ágeis no processo de desenvolvimento de software depende de vários fatores como: a complexidade do produto de software, a maturidade e o comprometimento das equipes, a disponibilidade do cliente, a cultura da empresa, entre outros. Dentre as metodologias descritas, o mais indicado para o trabalho foi a abordagem Scrum, por ser um *framework* simples e pequeno e, assim, funcionar bem em cada contexto, mesmo sendo utilizado em conjunto com outras técnicas e práticas, ou seja, a adaptatividade é uma das características mais marcantes do Scrum. Com isso, apresentaram-se as regras do Scrum, que agregam os papéis, eventos e artefatos, mostrando que cada elemento dentro do Scrum serve a um propósito específico e é fundamental para o uso e sucesso da metodologia em questão. Por fim, apresentou-se o ambiente do estudo de caso, o NTI, e a forma em que atualmente gerenciam o desenvolvimento de software, sem o uso de nenhum método tão bem estruturado, mostrando todos os problemas gerados com a falta de organização no gerenciamento de desenvolvimento.

Analisando algumas vantagens do Scrum, todos os problemas identificados no ambiente de desenvolvimento e visando à necessidade da melhoria no gerenciamento dos projetos, descreveu-se uma proposta para adaptação da metodologia ágil Scrum no NTI.

O grande desafio do estudo de caso foi integrar o Scrum as necessidades do NTI, do cliente e do time Scrum, estabelecendo as adaptações necessárias. Pode-se notar que apenas alguns conceitos do Scrum foram utilizados puramente nessa proposta e alguns adaptados. Mesmo com a implementação parcial desta metodologia, acredita-se que será possível observar alguns impactos positivos, só do fato de não ter atualmente nenhuma forma bem estruturada e definida de gerenciamento de desenvolvimento de software.

Essa proposta talvez encontraria resistência na falta de compromisso dos membros da equipe. A mudança da metodologia no desenvolvimento de software em uma empresa não muda em pouco tempo, precisa-se mudar o jeito de agir e pensar de todos os integrantes da equipe de desenvolvimento e outras pessoas envolvidas com essa equipe, ou seja, uma mudança cultural deve acontecer de forma gradual.

Os impactos realmente serão vistos com a aplicação da proposta, ficando como sugestão para trabalho futuro, a avaliação dos resultados da aplicação da proposta deste trabalho, a adaptação da abordagem Scrum no NTI.

REFERÊNCIAS

AMBLER, S. **Modelagem ágil: práticas eficazes para a Programação Extrema e o Processo Unificado**. Porto Alegre: Bookman, 2004.

AMBLEW, Scott W., **Examining the Agile Manifesto**. 2009. Disponível em <http://www.ambysoft.com/essays/agileManifesto.html>. Acesso em 01 de abril de 2015.

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. **Extreme Programming: Guia prático.** Rio de Janeiro, 2002.

BASSI, D. L. F. **Experiências com desenvolvimento ágil**. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo. São Paulo, 2008.

BECK, K. et al. **Manifesto para Desenvolvimento Ágil de Software.** 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 11 de abril de 2015.

COCKBURN, Alistair. **Agile Software Development: The Cooperative Game**. Addison Wesley, 2002.

COCKBURN, Alistair. **Escrevendo Casos de Uso Eficazes**. Bookman, 2007.

DESENVOLVIMENTO ÁGIL. COM. BR**. Scrum.** Disponível: http://www.desenvolvimentoagil.com.br/Scrum/**.** Acesso 01 de maio de 2015.

FOWLER, M. ***The New Methodology*.** 2005. Disponível em: <http://martinfowler.com/articles/newMethodology.html>. Acesso em 05 de abril de 2015.

HIGHSMITH, J. **Gerenciamento Ágil de Projetos Desenvolvimento de Software Adaptativo**, 2002.Disponível em: <http://www.adaptivesd.com>. Acesso em: 02 de junho de 2015.

HIGHSMITH, Jim. **Agile Project Management: Creating Inovative Products**. Boston: Addison-Wesley, 2004.

MALLMANN, Paulo Roberto. **Um modelo abstrato de gerencia de software para metodologias ágei**s. Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos. São Leopoldo, RS, 2011.

MEDEIROS, Higor. **Modelos de Processos Ágeis: conceitos e princípios**, 2014. Disponível: <http://www.devmedia.com.br/modelos-de-processos-ageis-conceitos-e principios/30059>. Acesso em 23 de maio de 2015.

NTI, Nucleo de tecnologia da informação. Disponível: <http://caxias.ufma.br:8080/PortalNti/paginas/referencial-estrategico.jsf> Acesso em 01 de junho de 2015.

PHAM, Andrew; PHAM, Phuong-Van. **Scrum em ação: gerenciamento e desenvolvimento ágil de projetos de software**. São Paulo:Novatec Editora, 2011.

PRESSMAN, Roger S. **Engenharia de Software**, Sexta Edição. Editora MCGrawHill: Porto Alegre, 2010.

ROCHA, Fabio Gomes. **Introdução ao FDD - Feature Driven Development.** 2013. Disponível em:<http://www.devmedia.com.br/introducao-ao-fdd-feature-driven-development/27971>. Acesso em 20 de Maio de 2015.

SABBAGH, Rafael. **Scrum: Gestão Ágil para Projetos de Sucesso**, São Paulo, Casa do Código, 2013.

SAMPAIO, Américo Tadeu Falcone. **XWEBPROCESS: um processo ágil para o desenvolvimento de aplicações web**. Dissertação (Mestrado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, 2004.

SCHWABER, Ken. **Guia do Scrum**. Scrum Alliance, 2010. Disponível em: <http://www.Scrumguides.org/docs/Scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf.pdf > acesso em 10 de junho 2015.

SOARES, Michel dos Santos. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**. Unipac - Universidade Presidente Antônio Carlos, Faculdade de Tecnologia e Ciências de Conselheiro Lafaiete, 2004.

SOARES, Michel dos Santos. **Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software**, 2004. Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/view/146>. Acesso em 10 de maio de 2015.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Pearson, 2007.

VARASCHIN J. D. Monografia: **Implantando o SCRUM em um Ambiente de Desenvolvimento de Produtos para Internet**, Pontifícia Universidade Católica do Rio de Janeiro. PUC, 2009.Disponível em: < ftp://ftp.inf.puc-rio.br/pub/docs/techreports/09_07_varaschim.pdf>. Acesso em: 15 de junho 2015.