GOVERNO DO ESTADO DE SÃO PAULO
CENTRO PAULA SOUZA
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

SCRUM

EDVALDO NOBUO HIGUCHI

Americana, SP
2014

GOVERNO DO ESTADO DE SÃO PAULO
CENTRO PAULA SOUZA
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

SCRUM

EDVALDO NOBUO HIGUCHI

edhiguchi@gmail.com

Projeto desenvolvido em cumprimento curricular da disciplina Projeto de Graduação do Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação da FATEC – Americana, sob orientação do Prof. Dr. Wladimir da Costa.

área: Análise de Sistemas e Tecnologia da Informação

Americana, SP
2014

Higushi, Edvaldo Nobuo
H541s Scrum. / Edvaldo Nobuo Higuchi. – Americana: 2014.
61f.

Monografia (Graduação de Tecnologia em Análise de Sistemas e Tecnologia da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. Wladimir da Costa

1. Desenvolvimento de software 2. Scrum – software de projetos I. Costa, Wladimir da II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.05
681.3.077

BANCA EXAMINADORA

Professor: Waldimir da Costa (Orientador)

Professor: Alberto Martins Junior

Professor: Renato Kraide Soffner

DEDICATÓRIA

À Deus que me iluminou nessa longa jornada e à minha família que sempre estiveram ao meu lado, dando força nos momentos difíceis.

AGRADECIMENTOS

À todos os professores do curso, que foram muito importantes na minha vida acadêmica, em especial ao professor Wladimir que me ajudou no desenvolvimento deste trabalho.

Aos meus colegas de classe, pelas alegrias, tristezas e dores compartilhadas nesse período que passamos juntos, em especial à Gabrielli de Oliveira, que conseguiu tornar tudo isso muito mais fácil.

À todos os meus amigos que estiveram presentes me apoiando, dando força, incentivando e tendo paciência comigo neste período.

À todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

“A vontade de mudar é uma virtude, mesmo que confunda parte da empresa durante um tempo.”
(JACK WELCH)

RESUMO

Nos dias atuais o gerenciamento de projetos é uma área que vem crescendo muito dentro da área de TI, devido a grande preocupação com custos, prazos e qualidade, que acabam causando grandes falhas e danos econômicos as empresas. Neste sentido, visando uma melhora na qualidade, redução de custos e entregas no prazo, este projeto tem como objetivo mostrar a eficiência da metodologia ágil Scrum, que vem crescendo e ganhando espaço no mercado de desenvolvimento de software, desde sua implantação até os resultados gerados e uma comparação com o modelo Clássico, que é uma das metodologias mais utilizadas nos dias de hoje para o desenvolvimento de software, que apesar de se destacar, ele apresenta diversas falhas, e por isso vem perdendo muito espaço no mercado. Na área de tecnologia a mudança é constante e por isso é fundamental que as empresas procurem por métodos, ferramentas e processos melhores e que trazem resultados.

Palavras chave: Scrum, metodologia ágil, TI, gerenciamento de projetos.

ABSTRACT

Nowadays project management is an area that has grown very much in the IT area, due to great concern about cost, time and quality, they end up causing major economic damage and failures to the companies. In this sense, targeting an improvement in quality, cost reduction and on-time deliveries, this project aims to show the efficiency of the agile methodology Scrum, which has been growing and gaining momentum in software development market, from its inception until the results generated and a comparison with the Classic model, which is one of the methodologies used today for software development, which despite stand out, it has several flaws, and so far has been losing market share. In the area of technology change is constant and so it is essential that companies look for methods, tools and processes and bring better results.

Key words: Scrum, Agile, IT, project management.

Lista de figuras

Lista de tabelas

LISTA DE ABREVIATURAS

ADM	Advanced Development Methods
DSM	Design Structure Matrix
OMG	Object Management Group
OOAD	Object-Oriented Analysis and Design
RAD	Rapid Application Development
ROI	Return on investiment
TI	Tecnologia da informação
VF	Valor futuro
VLP	Valor presente líquido
VP	Valor presente
XP	Extreme Programming

Sumário

1 INTRODUÇÃO

Apesar do grande avanço tecnológico, muitos projetos de *software* não conseguem atender aos prazos estipulados e com todas as funcionalidades e especificações requeridas, outros projetos acabam não sendo finalizados e outros ainda são entregues com prazos maiores, valores maiores e com menos funcionalidades, muitos desses problemas são causados devido a metodologia utilizada no desenvolvimento de *software*.

Processos voltados à documentação acabam limitando os desenvolvedores, fora que muitas empresas não tem recursos suficientes para essas metodologias consideradas pesadas, principalmente as pequenas, que muitas vezes acabam não utilizando metodologia alguma no desenvolvimento do *software*, gerando um verdadeiro caos dentro da empresa.

A metodologia ágil veio para simplificar o processo de desenvolvimento de *software*, principalmente para pequenos projetos, ela tem como foco as pessoas e não os processos, preocupando sempre com a implementação ao invés da documentação. Outra qualidade da metodologia ágil, é a maneira que ela se adapta as situações, aos problemas que estão ocorrendo durante o processo de desenvolvimento e não nos problemas que poderão ocorrer, como acontecem nas metodologias tradicionais.

Dentre as metodologias ágeis, uma das mais utilizadas é o *Scrum*, voltada para projetos pequenos, e tem como foco encontrar formas flexíveis de desenvolver software em um ambiente em constante mudança.

A pergunta a ser respondida neste trabalho é: Como a implantação do *Scrum* pode trazer resultados para a empresa?

Existe sempre um grande problema quando um projeto não obtém os resultados esperados, como perda de tempo, gastos desnecessários e a perda de credibilidade da empresa.

As hipóteses deste trabalhão são: A empresa obteve sucesso com a utilização do *Scrum* para o desenvolvimento de *software*. A empresa não implantou o *Scrum* corretamente, causando assim, prejuízo à empresa. A empresa implantou o *Scrum* corretamente, mas mesmo assim não obteve sucesso nos negócios.

Este trabalho tem como objetivo geral comparar a metodologia clássica com a

metodologia ágil *Scrum* em um projeto de software e demonstrar a eficiência da metodologia ágil *Scrum* a clássica. Além de explicar o que é a metodologia *Scrum* e suas funcionalidades, mostrar também suas vantagens e desvantagens, mostrar como são feitas as etapas de um desenvolvimento de *software* utilizando a metodologia ágil *Scrum* e discutir os resultados obtidos no estudo comparativo, visando mostrar como a metodologia *Scrum* é eficiente.

A Justificativa para este trabalho é que devido ao *Scrum* ser um metodologia ágil que se destaca, devido a participação do usuário, baixo custo, produção de projetos de sistemas mutáveis e não estáticos em relação a outras metodologias.

O método utilizado neste trabalho é um estudo comparativo, devido a necessidade de comparar as etapas para a implantação do método *Scrum* no desenvolvimento de *software* com outra metodologia. De acordo com Lijphart (1970), o estudo comparativo tem o objetivo de descobrir a relação empírica entre as variáveis estudadas.

Complementando a pesquisa será utilizado também a pesquisa bibliográfica, pois será utilizado livros, revistas e sites relacionados ao tema para a conclusão da pesquisa. A pesquisa bibliográfica é elaborada a partir de um material já publicado, constituído principalmente de livros, artigos e atualmente com materiais disponibilizados na Internet. (MINAYO, 2007; LAKATOS, 1986).

A delimitação do trabalho esta na abrangência que o tema *Scrum* trás, devido ao grande conteúdo, limitando o trabalho apenas aos passos da implantação da metodologia *Scrum* em um pequeno projeto.

A relevância do tema *Scrum* se deve ao fato de ser um assunto atual, na qual muitas empresas estão de olho, devido ao sucessos de outras empresas que utilizam desta metodologia para o seu crescimento.

Outro fator importante sobre o tema, é a forma como o processo de implantação dessa metodologia é feito e quais são seus passos, ajudando assim, analistas e gerenciadores de projetos que não tem um conhecimento amplo sobre o assunto, e pretendem implantar essa metodologia em futuros projetos.

No capítulo 2 é apresentado uma visão geral sobre engenharia de software, metodologias tradicionais e metodologias ágeis, focando na metodologia ágil *Scrum*, que vem se destacando e crescendo muito ultimamente.

No capítulo 3 é feito uma comparação entre a metodologia ágil *Scrum* e o modelo Clássico, mostrando suas vantagens e desvantagens, e qual é a melhor

metodologia para o desenvolvimento de software nos dias atuais.

2 METODOLOGIAS PARA DESENVOLVIMENTO DE SOFTWARE

Neste capítulo é dada uma visão geral sobre as metodologias tradicionais e metodologias ágeis, focando a metodologia ágil *Scrum*.

2.1 SISTEMA DE INFORMAÇÃO

Segundo Turban (2007) o termo sistema de informação é utilizado para representar um sistema automatizado ou manual, onde há o envolvimento de pessoas, máquinas e métodos organizados para obter dados, processá-los e transmitir informações para um usuário. Já para Laudon e Laudon (1995) o sistema de informação é um conjunto de componentes que juntos coletam, processam, armazenam e distribuem informações que ajudam nas tomadas de decisões de uma organização.

Um dos objetivos do sistema de informação é coletar os dados e transformá-los em informação ou conhecimento. Os dados são registros, transações, atividades ou eventos que sozinhos não conseguem transmitir nenhum significado, eles podem ser letras, números, figuras, imagens ou sons. Exemplos de dados são salários dos funcionários ou notas dos alunos, que sozinhos não fazem sentido algum. A informação é um conjunto de dados organizados que conseguem fazer algum sentido para o usuário, como por exemplo os salários de uma empresa associado ao nome dos funcionários. O conhecimento são dados ou informações que foram organizados e processados de maneira que transmite entendimento, experiência e aprendizagem para que as pessoas consigam tomar decisões em problemas atuais. Por exemplo, uma empresa que recruta estagiários de uma universidade, conseguiu ao longo dos anos, identificar que os alunos que tem média acima de 3.0 conseguem ser mais bem sucedidos dentro da empresa, e com base nesse conhecimento, a empresa recruta somente alunos que atingirem essa média.

Apesar de existir sistemas de informações manuais, a maioria deles é computadorizado, por isso o termo “sistema de informações “ é sinônimo para “sistemas de informações baseado em computador”. Um sistema de informação é

composto por: *hardware*, *software*, banco de dados, redes, processos e pessoas. Sendo *hardware* a parte física do computador como monitor, teclado, impressora, mouse, entre outros. Já o *software* é a parte lógica do computador, como um programa que roda no computador. Banco de dados é um conjunto de arquivos ou tabelas relacionados contendo dados. Rede é um sistema de conexão com ou sem fio, que permite que computadores diferentes compartilhem recursos. Procedimentos são conjunto de instruções sobre como proceder com os componentes para processar as informações e gerar a saída desejada. E por último as pessoas, que são os indivíduos que utilizam o *software* e o *hardware*.

Há vários tipos de sistema de informações, dentro de uma empresa pode haver um tipo de sistema de informação para cada departamento específico.

2.2 ENGENHARIA DE SOFTWARE

De acordo com Sommerville (2007) a engenharia de *software* é uma disciplina da engenharia que tem relação com todas as etapas na criação de um *software*, desde as especificações do sistema até a manutenção, após o produto ter sido implantado. A engenharia de *software* não esta relacionada apenas as partes técnicas no processo de produção do *software*, mas também na parte de gerenciamento e na parte de desenvolvimento de ferramentas, métodos e teorias que ajudam no desenvolvimento do *software*. Os engenheiros trabalham com uma abordagem sistemática e organizada para construir o *software* com qualidade.

> “A engenharia de software é um rebento da engenharia de sistemas e de hardware. Ela abrange um conjunto de três elementos fundamentais - métodos, ferramentas e procedimentos - que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente.” (PRESSMAN, 1995)

O processo de *software* é um conjunto de atividades que os desenvolvedores utilizam na criação do *software*, existem quatros atividades comuns que são utilizadas na maioria dos projetos que são elas: especificação do *software*,

desenvolvimento do *software*, validação do *software* e evolução do *software*.

Um modelo de processo de *sofware* é uma descrição simples do processo de *software*, que incluem as atividades que fazem parte do processo, produtos de *software* e os papéis das pessoas que estão envolvidas. Alguns dos modelos de processo de *software* são:

- Um modelo de *workflow*: mostra a sequência de atividades durante o processo, suas entradas e saídas e a relação entre elas, onde as atividades representam as ações humanas.
- Um modelo fluxo de dados ou modelo de atividade: mostra um conjunto de atividades, onde cada uma delas executa alguma transformação de dados, que nesse caso, é executada por pessoas ou por computadores.
- Um modelo de papel/ação: mostra os papéis das pessoas envolvidas no processo de desenvolvimento de *software*, e o que cada uma é responsável por executar.

2.2.1 Fases do desenvolvimento de software

De acordo com Pressman (1995), existem três fases no processo de desenvolvimento de *software* independente do tipo ou complexidade de projeto, que são a definição, desenvolvimento e manutenção.

Na fase da definição é identificado o que o *software* deve conter, as informações que serão processadas, as funções que serão estabelecidas, as restrições que existirão, os critérios de validação que serão exigidos. São feitas três etapas específicas: análise do sistema, onde é definido o papel de cada componente em um sistema; planejamento do projeto de *software*, onde é feito uma análise de todo o *software* e definido o escopo do *software*, recursos, tarefas, custos e riscos; análise de requisitos

Na fase de desenvolvimento é focado em como o *software* deve ser projetado, e são utilizados três passos: projeto de *software*, codificação e realização de testes do *software*.

Na fase de manutenção é concentrado nas mudanças que ocorrem para correção dos erros, existem três mudanças: correção, adaptação e melhoramento

funcional.

2.2.2 Requisitos

Segundo Sommerville (2007) os requisitos são as descrições das funcionalidades e restrições do sistema, que representam as necessidades do cliente para um sistema. É utilizado a engenharia de requisitos para identificar, analisar, documentar e checar os serviços do sistema.

A falta de clareza na definição dos requisitos pode gerar grandes problemas durante o processo de engenharia de requisitos, por isso é feita uma separação dos tipos de requisitos, melhorando assim as especificações para cada diferente tipo de leitor do sistema. Os requisitos são classificados em funcionais, não funcionais e de domínio:

- Requisitos funcionais: são as descrições do que o sistema deve fazer, por isso eles devem ser especificados de forma clara e completa. Quando eles representados na forma de requisitos de usuários, a descrição é mais abstrata, já quando são representados pelas funções do sistema, eles são descritos mais detalhadamente, contendo suas entradas, saídas, etc.

  Quando os requisitos não estão especificados muito bem, acabam gerando grandes problemas na hora do desenvolvimento, pois o programador pode ter uma outra interpretação do requisito. Geralmente é feito mudanças nos requisitos ou é inserido novos requisitos durante o desenvolvimento do projeto, porém isso acaba atrasando todo o cronograma do projeto.
- Requisitos não funcionais: são aqueles que não estão ligados diretamente com as funções do sistemas, mas sim com elementos emergentes do sistema como confiabilidade, armazenamento, tempo de resposta, entre outros. Alguns deles estão relacionados aos processos utilizados no desenvolvimento como o tipo de ferramenta que deve ser utilizado, uma especificação do processo de qualidade ou uma descrição de um processo que deve ser seguido no desenvolvimento do *software*. Os tipos de requisitos não funcionais são: os requisitos do produto, que são as especificações do desempenho do produto, como quantidade de memória requerida, quanto de

velocidade; requisitos organizacionais, que são decorrentes de políticas e procedimentos do cliente, como quais métodos e linguagens devem ser utilizados no desenvolvimento, qual data deve ser entregue o produto, quais requisitos éticos e legais que o sistema precisa para estar dentro das normas da lei.

- Requisitos de domínio: estão relacionados ao conceito do domínio e podem delimitar os requisitos funcionais atuais ou estipular como devem ser executados. São requisitos específicos, onde os desenvolvedores encontram dificuldades para associa-los à outros requisitos do sistema, porém são muito importantes, pois demonstram o domínio da aplicação.
- Requisitos de usuário: são as descrições dos requisitos funcionais e não funcionais da forma mais simples possível para que o usuário, que não tem um conhecimento técnico, possa entender com clareza. Os maiores problemas encontrados nos requisitos de usuários são a falta de clareza nas descrições, confusão dos requisitos, porque muitos usuários não conseguem diferenciar entre os requisitos funcionais e não funcionais, e a fusão entre os requisitos.
- Requisitos do sistema: são versões mais elaboradas dos requisitos de usuários e são usadas pelos engenheiros de *software* como um ponto inicial para a desenvolvimento do projeto, eles usam as informações dos usuários e explicam como as funções solicitadas serão executadas pelo sistema, isso tudo de forma bem detalhada e completa.

  O documento de requisitos é onde esta documentado todos requisitos que são considerados oficiais e serão utilizados no desenvolvimento de *software*. Ele é utilizado por vários tipos de usuários, desde de a gerência que paga pelo *software*, até os desenvolvedores do projeto, por isso ele deve estar escrito de maneira que todos possam entender.

2.3 METODOLOGIAS TRADICIONAIS

Segundo Koscianski e Soares (2007) as metodologias tradicionais são aquelas consideradas pesadas ou voltadas à documentação, elas surgiram em um

ambiente de desenvolvimento de *software* muito diferente do atual, onde eram utilizados apenas *mainframes* e terminais burros. Nesta época era muito alto o preço para alterações e correções, devido a limitação ao acesso aos computadores e não existiam muitas opções de ferramentas de apoio no desenvolvimento de *software*, como depuradores e analisadores de código, por isso era feito um planejamento do *software* e documentado antes mesmo de ser implementado. Uma metodologia tradicional muito utilizada ainda nos dias de hoje é o modelo Clássico.

2.3.1 Modelo Clássico

Conhecido também como modelo cascata ou sequencial, o modelo Clássico foi a primeira metodologia publicada no desenvolvimento de *software*. Nele é estabelecido uma sequência de etapas, onde ao final de cada uma delas é feita sua documentação, onde ela deve ser aprovada antes de passar para outra etapa. (Koscianski, Soares, 2007)

O ciclo de vida deste modelo passa pelas etapas de engenharia de sistemas, análise, projeto, codificação, testes e manutenção. (Pressman, 1995)

Segundo Koscianski e Soares (2007) as etapas no modelo Clássico não são executadas de maneira flexível, por exemplo no modelo cascata após a fase de desenvolvimento não se prevê mudanças nas especificações, enquanto em outros modelo, como o modelo espiral, é permitido o retorno às etapas anteriores, porém não pode existir a execução paralela entre as etapas. Isto pode ocorrer em um projeto em que se aplique engenharia concorrente ou componentes diferentes do produto que sejam desenvolvidos de maneira independente.

O custo das alterações no *software* aumentam conforme o andamento do projeto, o custo pode ser cem vezes maior depois que *software* esta pronto, por isso o modelo Clássico deve ser utilizado somente quando os requisitos forem estáveis.

Até o início da década de 1990, o modelo Clássico era quem comandava a forma de desenvolvimento de *software*, apesar das críticas dos pesquisadores na época, que identificavam muitos problemas com a adoção dessa sequência de tarefas.A figura 1 mostra o ciclo de vida de um modelo Clássico.

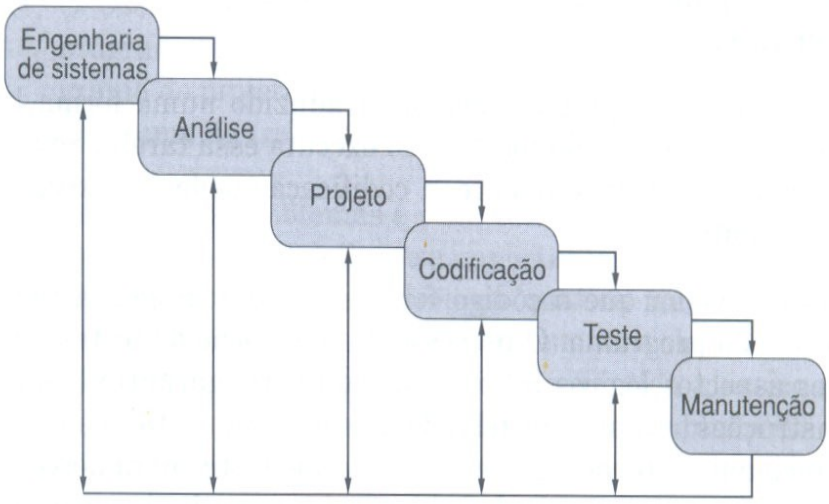

Figura 1: Ciclo de vida Clássico.

Fonte: PRESSMAN, 1995, p. 33

2.3.1.1 Etapas do modelo Clássico

Segundo Pressman (1995) o modelo Clássico passa por algumas etapas para o desenvolvimento de *software* que são:

- Análise e engenharia de sistemas: Para o *software* funcionar é necessário a integração do *software* com outros componentes como *hardware*, banco de dados e pessoas, por isso nesta etapa é feito uma coleta dos requisitos no nível do sistema, uma pequena parte de projeto e também uma análise de alto nível.
- Análise de requisitos de *software*: Nesta etapa é feito uma coleta mais rígida dos requisitos e feito uma análise por parte dos analistas, que por sua vez tem que entender de forma clara o que o cliente quer, as funções, desempenho e interface do *software*. Os requisitos precisam ser revistos no caso de dúvidas e devem ser documentados. Em algumas empresas esta etapa é feito junto a etapa de análise e engenharia de sistemas.

- Projeto: Existem vários passos nesta etapa, que centralizam em quatro atributos que são: a estrutura de dados, a arquitetura de *software*, os detalhes dos procedimentos e a caracterização da interface. Toda a parte das preliminares do *software* e o projeto de forma detalhada é feito nesta etapa.
- Codificação: Nesta etapa o projeto deve ser todo detalhado em forma de código.
- Testes: Após a codificação começa a etapa dos testes, onde tudo que foi codificado deve ser executado, garantindo que todas as funcionalidades estejam funcionando, descobrindo todos os erros e garantindo que tudo que foi exigido esteja em conformidade com os resultados.
- Manutenção: Após a implantação do *software*, pode ocorrer alguns erros que são corrigidos nesta etapa. O cliente pode exigir também além das correções algumas mudanças que também são feitas nesta etapa, porém todas as alterações são feitas no *software* já existente e não criado um novo.

2.3.1.2 A implementação Bottom-up

Segundo Yourdon (1990) um dos principais problemas do modelo Clássico é a implementação bottom-up, onde é feito todos os testes no final do projeto, não se sabe a origem dessa abordagem, mas acredita-se que pode ter sido emprestada por indústrias na fabrição de automóveis, porém são produtos totalmente diferentes e por isso são encontrados vários problemas:

- O projeto esta terminado no final do projeto, ou seja, caso o projeto atrase e o cliente exigir um posicionamento, a empresa não terá nada concreto para mostrar, dependendo da etapa que estiver, terá somente códigos.
- Os principais erros encontrados no projeto são na fase de testes, e dependendo dos erros é necessário uma modificação de alguns módulos, o que pode ser cansativo para os programadores, já que muitos estão trabalhando meses no projeto.
- A depuração dos erros é outro problema, pois depois de descoberto o erro nas fases de teste, é muito difícil entrar em qual módulo esta o erro, pois dependendo do código ele pode estar em uma das milhares linhas.

- Devido a realização dos testes no final do projeto, muitas empresas tem que disponibilizar servidores e computadores por várias horas no final do projeto, o que pode ser um problema, se a empresa não tiver disponível servidores para testes durante horas.

2.3.1.3 Progressão Sequencial

Segundo Yourdon (1990) outro fator negativo é a insistência para as fases sejam sequencialmente seguidas, isso seria ótimo se tudo ocorresse bem em todas as etapas, porém é muito difícil que tudo ocorra perfeitamente bem na primeira etapa e que não tenha nada para arrumar futuramente para dizer que a etapa esta totalmente concluída.

Outro problema associado ao modelo Clássico é que dependendo do software, o projeto pode durar meses ou até anos, e o ambiente do cliente pode sofrer alterações, um requisito que não é mais necessário ou até uma nova lei do governo, e as mudanças nos requisitos não é flexível neste modelo, uma vez definido os requisitos e documentado, ele não pode ser alterado até o final do projeto.

Uma característica do modelo Clássico é o fato de se basear em técnicas ultrapassadas, como por exemplo não utilizar programação estruturada ou outras técnicas mais atuais para desenvolvimento. Apesar de nada impedir os gerentes de projetos de utilizar dessas novas técnicas, a maioria vê o modelo Clássico como uma diretriz da orientação da alta direção, por isso como o modelo Clássico não diz nada sobre essas técnicas, eles acreditam que por isso não sejam necessárias.

2.4 METODOLOGIA ÁGEIS EM PROJETOS DE TI

Segundo Koscianski e Soares (2007) as metodologias ágeis aceitam as mudanças que podem ocorrer durante um projeto, por isso elas são perfeitas para projetos que necessitem da mudança de requisitos com frequência.

O termo metodologia ágil surgiu em 2001, quando 17 especialistas que atuavam em um processo de desenvolvimento de *software*, que representavam os métodos *Extreme Programming* (XP), *Scrum*, DSM, *Crystal*, entre outros, definiram alguns princípios que representavam esse método, foi criado então a Aliança Ágil e o estabelecimento do Manifesto Ágil. As principais características da metodologia ágil são desenvolvimento iterativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva.

Os principais conceitos do Manifesto Ágil são:

- indivíduos e interações em vez de processos e ferramentas;
- software executável em vez de documentação;
- colaboração do cliente ao invés de negociação de contratos;
- respostas rápidas a mudanças em vez de seguir planos.

> "A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento. A filosofia defende a satisfação do cliente e a entrega de incremental prévio; equipes de projetos pequenas e altamente motivadas; métodos informais; artefatos de engenharia de software mínimos e, acima de tudo, simplicidade no desenvolvimento geral. Os princípios de desenvolvimento priorizam a entrega mais que a análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes". (PRESSMAN, 2011)

2.4.1 Extreme Programming (XP)

Não tem como falar de *Scrum* e não falar também sobre o *Extreme Programming* (XP), que também é uma metodologia ágil, porém seu foco é o desenvolvimento rápido de projetos, buscando garantir a satisfação do cliente. Segundo Brod (2013) seus valores são: comunicação, simplicidade, *feedback* e coragem.

O objetivo da comunicação é garantir que o relacionamento entre o cliente e os desenvolvedores da melhor maneira possível, dando sempre preferencia para conversas pessoais. As conversas entre o gerente de projeto e os desenvolvedores também é importante.

A finalidade da simplicidade no projeto é garantir que o código do programa seja o mais enxuto possível, evitando que funções desnecessárias sejam adicionadas, visando manter o foco nos requisitos atuais, ao invés de pensar em funções que poderiam ser úteis no futuro. A implementação de novos requisitos custarão bem menos do que criar um código extenso e complexo.

A prática do *feedback* constante é importante porque sempre os desenvolvedores terão informações do cliente sobre o código. A intenção do *feedback* é testar sempre código para detectar e eliminar os erros, garantindo que no final o *software* esteja totalmente funcional, e entregar partes do *software* que ja estão funcionando para que o cliente teste e passe as informações para os desenvolvedores, desta maneira ao final do projeto, o *software* atenderá todos os requisitos solicitados pelo cliente.

É preciso muita coragem para colocar os três valores anteriores em prática, a coragem representa a força que uma equipe de XP precisa ter para garantir que a comunicação esta sendo feita, garantir um código simples e exigir *feedback* constante do cliente.

Além das quatro valores descritos, a XP é baseada também em 12 práticas que são elas:

- Planejamento: a XP trabalha com requisitos atuais e não em requisitos futuros, por isso o planejamento é baseado nos requisitos que vão ser trabalhados agora e o restante é adiado. O planejamento é um trabalho em comum entre a área de negócios e a área de desenvolvimento, enquanto a área de negócios corre atrás do escopo, composição das versões e datas de entrega, a área de desenvolvimento se preocupa com o processo de desenvolvimento, cronograma detalhada e as estimativas de prazo.
- Entregas frequentes: a primeira entrega é a do *software* simples, e a medida que surgem novos requisitos, eles são implementados em novas versões, que devem ser cada vez mais compactas. O ideal seria uma nova versão por mês ou uma há cada dois meses, sempre recebendo os *feedbacks* do cliente.
- Metáforas: são as descrições do *software* de maneira simples, sem o uso de termos técnicos, afim de facilitar o entendimento do desenvolvedores do *software*.
- Projeto simples: o *software* deve ser simples, de maneira a atender os

requisitos atuais, sem se preocupar com requisitos futuros, de maneira que o *software* não tenha funcionalidades desnecessárias. Conforme novas necessidades vão surgindo, são implementadas novas versões.

- Testes: o *software* é desenvolvido com casos de testes, para que o projeto seja testado durante todo o processo de desenvolvimento.
- Programação em pares: em um único computador, dois desenvolvedores trabalham na implementação do código, enquanto um trabalha no código o outro fica observando para identificar possíveis falhas, e eles vão alternando durante o processo, dessa maneira fica mais fácil para corrigir os erros.
- Refatoração: durante todo o período do projeto, é feito uma busca por maneiras de aperfeiçoar o *software*, de maneira que simplifique o *software* sem perder nenhuma de suas funcionalidades.
- Propriedade coletiva: no XP todos são responsáveis pelo projeto do *software*, então qualquer pessoa, mesmo que não tenha desenvolvido o *software* pode alterá-lo, desde que passe por todos os testes necessários, desta maneira quando um membro deixa o projeto, o restante consegue continuar com o projeto, mesmo que com certa dificuldade.
- Integração contínua: toda vez que o código for testado e validado, ele é implementado no sistema, que também deve ser testado, esse sistema deve ficar em uma única máquina que deve ter acesso livre para todos os membros da equipe. Esse processo deve ser feito gradativamente, com a finalidade de isolar erros e falhas.
- Trabalho semanal de 40 horas: o projeto deve ser realizado no período normal de trabalho, não deve existir horas extras no projeto, caso tenha a necessidade de realizar horas extras pela segunda semana consecutiva, é porque existe alguma coisa errada e o planejamento deve ser revisto e melhorado.
- Cliente presente: a participação do cliente no desenvolvimento do projeto é fundamental para tirar as dúvidas sobre os requisitos, evitando assim, atrasos ou erros no requisito.
- Código-padrão: adoção de regras na escrita e alguns identificadores facilitam muito, essa padronização permite que todos entendam o código.

Segundo Koscianski e Soares (2007) alguns pontos positivos da metodologia

XP são: A XP é ótima para projetos em que os *stakeholders* não sabem exatamente o que querem e mudam frequentemente de opinião durante o desenvolvimento. O uso do *feedback* constante ajuda realizar os ajustes necessários com mais facilidade. As entregas com o *software* executável é outro ponto positivo, pois os clientes não precisam aguardar um período longo para testá-lo para identificar melhorias. Os testes frequentes e a integração também ajudam na melhoria da qualidade do *software*.

Os pontos negativos são: A falta do uso de diagramas, mesmo que alguns deles não são muito utilizados, mas alguns ainda ajudam muito na hora de entender o programa. A falta de formalidade na hora da definição dos requisitos, pode causar certa insegurança aos clientes. A refatoração do código também pode ser visto pelo cliente como uma forma amadora de desenvolvimento. Outro ponto negativo, é que alguns profissionais não aceitam muito bem algumas práticas dessa metodologia, uma delas é a programação em duplas. A exigência que o time trabalhe no mesmo local dificuldade bastante no caso de empresas que tem outras filiais.

2.5 SCRUM

Segundo Pham e Pham (2012) o termo *Scrum* surgiu em 1986, através de um artigo publicado por Hirotaka Takeuchi e Ikujiro Nonata, intitulado de "*The new new product development game*" na Harvard *Business Review*. Nele os autores descrevem uma abordagem abrangente em que equipes de projetos são compostas por equipes menores multifuncionais trabalhando juntos para um objetivo comum, comparado com a formação do *Scrum* do *rugby*.

Jeff Sutherland, na época então vice-presidente de Engenharia da Easel, Inc., estava trabalhando no desenvolvimento de uma ferramenta de Análise e Projeto Orientados a Objeto (*Object-Oriented Analysis and Design*, ou OOAD), quando ele percebeu que precisava melhorar o desempenho de sua equipe com o Desenvolvimento Rápido de Aplicações (*Rapid Application Development*, RAD), e precisava algo parecido com o *Scrum*.

Na mesma época Ken Schwaber, que trabalhava na empresa *Advanced Development Methods* (ADM), estava pesquisando maneiras de ajudar sua empresa

a melhorar o desempenho no processo de criação de *software*, quando ele percebeu que as empresas vendedoras de *software* independentes bem sucedidas trabalhavam de maneira semelhante, usando processos que exigiam inspeção e adaptação constante.

Em 1995 à pedido da *Object Management Group* (OMG), Jeff e Ken trabalharam juntos e colocaram em prática o que haviam aprendido ao longo dos anos e criaram a metodologia *Scrum*, que foi descrita em um artigo do Schwaber "*Scrum and the perfect Storm*".

2.6 A TRANSIÇÃO PARA O SCRUM

Segundo Cohn (2011) as pessoas costumam ser resistentes a mudança, isso é um fator normal, e com o *Scrum* não poderia ser diferente. Para realizar a transição da sua empresa para o *Scrum*, não pode ocorrer uma mudança inteiramente de cima para baixo, onde um líder ou gerente influente tem uma visão do *Scrum* e resolver mudar toda a estrutura da empresa de uma hora para outra. A mudança também não pode ser inteiramente de baixo para cima, onde alguns membros decidem que é hora de uma mudança e começam a realizá-la. Uma transição bem sucedida de *Scrum* deve ser um meio termo entre esses dois tipos de mudanças.

Outra razão para uma transição bem sucedida é identificar as fraquezas do projeto e utilizar o *Scrum* como forma de melhoria. Muitas pessoas que conhecem o *Scrum*, fixam uma ideia na cabeça e acreditam que terão os mesmos resultados que outras empresas que o utilizam, mas na verdade é o *Scrum* que deve ser adaptado as circunstâncias da sua empresa e não ao contrário.

Uma transição para o *Scrum* é abrangente porque acaba interferindo nas maneiras de trabalho de algumas pessoas, por exemplo o desenvolvedor, que terá que trabalhar em várias tarefas ao mesmo tempo para entregar uma parte executável ao final de cada *sprint*, ou até mesmo os departamentos de finanças ou vendas que terão que se adaptar a maneira como são executados os projetos no *Scrum*, por isso as transições para o *Scrum* costumam ser mais difíceis que outras mudanças, porque existe uma resistência maior das pessoas.

As mudanças estão ocorrendo nos dias atuais de maneira muito mais rápida do que ocorria antigamente, as empresas estão exigindo muito mais e querendo pagar menos, novas tecnologias estão sendo implantadas, e o *Scrum* de certa forma acaba sendo um choque do futuro para as pessoas, porque elas precisam mudar a maneira de trabalho delas.

Apesar de toda dificuldade em realizar a transição para o *Scrum*, os *stakeholders* estão muito contentes com os resultados obtidos, e é possível identificar algumas razões pela qual vale a pena realizar uma transição para o *Scrum* como: maior produtividade e menores custos, maior engajamento satisfação da equipe, *time-to-market* mais veloz, maior qualidade e satisfação dos *stakeholders*.

2.6.1 Adaptação para o Scrum

Segundo Cohn (2011) existem cinco atividades necessárias para uma boa adoção do *Scrum*, que são elas: Reconhecimento (*Awareness*), Desejo (*Desire*), Aptidão (*Ability*), Promoção (*Promotion*) e Transferência (*Transfer*) que representam o acrônimo ADAPT conforme figura 2.

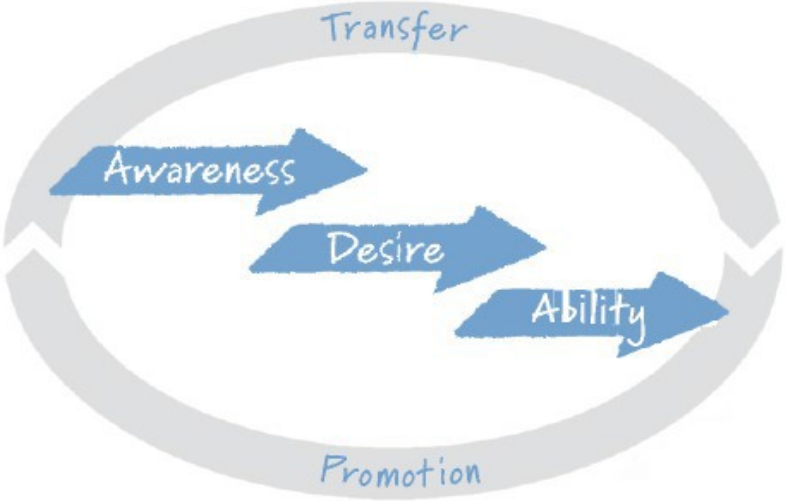

Figura 2: As cinco atividades de adaptação

ScrumFonte: COHN, 2011, p. 44

- Reconhecimento (*Awareness*): O primeiro passo é reconhecer o estado atual, identificar as coisas que não estão funcionando mais e melhorá-las. Muitos funcionários não tem uma visão ampla dentro da empresa, e por isso acabam não percebendo quando uma empresa esta em declínio. Por outro lado, muitos conseguem ver que uma mudança é necessário, mas preferem acreditar que os problemas são temporários. Algumas ferramentas que ajudam no desenvolvimento do reconhecimento são a comunicação, onde a empresa deve sempre divulgar os resultados; utilizar métricas facilita no conhecimento dos funcionários; promoção de novas pessoas e experiências; encorajar os funcionários a realizar treinamento de novas técnicas e práticas; convidá-los para participar de feiras no segmento, afim de conhecer os futuros lançamentos dos concorrentes; deixar os membros da equipe participar de reuniões com o cliente, afim deles ouvirem em primeira mão os recursos necessários e os prazos.
- Desejo (*Desire*): Além da conscientização do problema, as pessoas devem ter o desejo de querer mudar. Algumas ferramentas que ajudam aumentar o desejo são a comunicação, na atividade anterior a comunicação tinha como foco os problemas que a empresa tinha, já nessa atividade o foco da comunicação é o *Scrum*, a empresa deve mostrar como o *Scrum* pode resolver os problemas; criar um senso de urgência, o foco não é pressionar, mas sim deixá-los cientes da atual situação e o fato de que uma mudança é necessária urgente; construir um movimento, ao invés de se preocupar com as pessoas que estão relutantes a adaptação do *Scrum*, dedique-se as pessoas que estão motivados com o *Scrum* , assim conseguimos criar um movimento contínuo, um projeto com sucesso levando a outro; ao invés de tentar discutir teoricamente os benefícios do *Scrum*, deixar os funcionários fazerem um *teste drive*, uma pequena experiência para que eles sintam na prática os benefícios do *Scrum*; procurar eliminar o medo, devido as experiências anteriores com problemas de cronograma, requisitos, entre outros, é normal que as pessoas prefiram uma metodologia na qual os requisitos estejam mais bem detalhados, porém se formos analisar, este medo é totalmente infundado, já que na maioria das vezes, uma fase de *design*, onde são definidos os requisitos detalhadamente, não faz com que o *software* esteja pronto na data determinada; não ridicularizar o passado, pois

independente do processo de *software* utilizado anteriormente, a empresa conseguiu chegar onde esta devido à ele, porém devemos nos desligar do passado para que possamos desejar um novo futuro.

- Aptidão (*Ability*): Não é só necessário ter reconhecimento do problema e ter o desejo de mudar, temos que adquirir habilidade no *Scrum*, aprender novas habilidades técnicas, trabalhar em equipe e criar *software* em prazos pequenos são algumas das habilidades do *Scrum*. As ferramentas utilizadas para desenvolver aptidão são: fornecer *coaching* e treinamento adequados para que os funcionários conheçam melhor a metodologia, e possam ver na prática aquilo que estão aprendendo; responsabilizar as pessoas, pois elas devem estar cientes que aquilo que elas estão aprendendo, elas deverão por na prática, que estão sendo responsáveis pela aplicação dessa nova prática; compartilhar informações, pois no começo é interessante as pessoas compartilharem aquilo que estão aprendendo, seja através de *wikis* ou outros meios, ou mesmo deixarem os funcionários participarem das reuniões diárias de outras equipes; definir metas sensatas, pois no começo é normal que todos estejam com medo e não sabem exatamente por onde começar, então é melhor definir metas que são realistas e que a equipe possa cumprir; partir para a ação, a melhor maneira que conseguir habilidade com o *Scrum*, é trabalhar com o *Scrum*.
- Promoção (*Promotion*): Existem três objetivos durante a promoção, primeiro divulgar os sucessos atuais para as próximas etapas, promover o sucesso dentro da equipe, para que eles tenham conhecimento das coisas que conseguiram adquirir e promover o *Scrum* para outras pessoas de fora das equipes, mas que tenham contato indireto com o *Scrum*, como recursos humanos, *marketing*, vendas, operações e infraestrutura. Algumas ferramentas para promoção do *Scrum* são: divulgar as histórias de sucesso, transmitir as histórias dos projetos que tiveram sucesso com a adoção do *Scrum*, realizar apresentações internas com relatos de membros da equipe que já adotaram o *Scrum*; atrair a atenção de todos para essa nova metodologia, quanto mais pessoas souberem, maior vai ser o interesse em conhece-la.
- Transferência (*Transfer*): O uso do *Scrum* não deve limitar-se somente a uma equipe, ele deve ser transferido para toda empresa, caso contrário isso pode

implicar ou eliminar os esforços de transição, a empresa toda não precisa usar o *Scrum*, mas pelo menos deve ser compatível com ela. Enquanto nas outras etapas havia uma lista de ferramentas, nessa a única ferramenta que pode ser utilizada é a comunicação, comunicar-se com outros departamentos da empresa para que haja essa transferência do uso do *Scrum*.

2.7 O USO DO SCRUM

Segundo Pham e Pham (2012) a metodologia *Scrum* começa com o *Product Owner*, que é responsável por obter informações sobre os requisitos com os usuários e *stakeholders* e criar o *Product Backlog*, que é a lista dos requisitos com prioridades. Esses requisitos são coletados dos usuários em reuniões de planejamento anteriores, e coletados como se fossem histórias curtas. Essas reuniões de planejamento ajudam na descrição dos requisitos do produto, pois a equipe consegue conhecer melhor o negócio do cliente. Além da reunião de planejamento, existe também a reunião do planejamento do *sprint*. O tempo do projeto é devido em *sprints*, que deve ser de no máximo quatro semanas, e nas reuniões e planejamento do *sprint*, deve ser definido o *Sprint backlog*, que é uma lista de tarefas que devem ser executadas em cada *sprint*, que são registradas no *Task Board*, um quadro branco, na qual são escritos ou colocados as tarefas. figura 3 mostra a visão geral do *Scrum*.

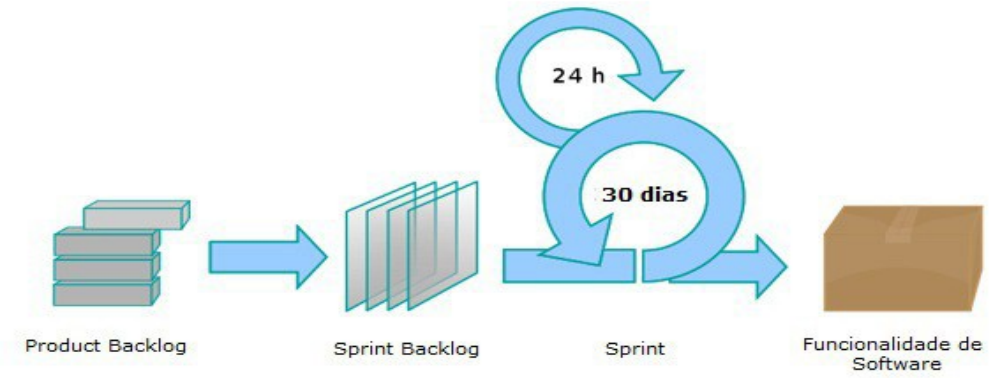

Figura 3: Visão geral do Scrum

Fonte: http://www.devmedia.com.br/desenvolvimento-agil-com-scrum-uma-visao-geral/26343

Definido o *Sprint Backlog* é hora de começar o desenvolvimento do projeto, durante o período de desenvolvimento é feito uma reunião diária de 15 minutos, o *Daily Scrum* ou *Daily Standup* para verificar o andamento do projeto, como na equipe *Scrum* não tem hierarquia, essas reuniões servem para verificar o progresso do projeto e não como uma forma de cobrança.

Para manter o controle do progresso de cada *sprint* é feito um registro em uma gráfico *Burndown*, a própria equipe vai atualizando conforme o andamento do projeto.

No final de cada *sprint* é feita a *Sprint Review*, nela é revisado o que foi feito com a equipe e o *Scrummaster* e é apresentado ao *Product Owner* o que foi feito e é obtido seu *feedback*. É feito também um *Sprint Retrospective*, uma reunião onde o *Scrummaster* e o *Product Owner* fazem uma análise do que foi feito, o que deu errado e o que pode melhorar no próximo *sprint*. Segue a figura 4 com as pessoas e artefatos relacionados ao *Scrum*.

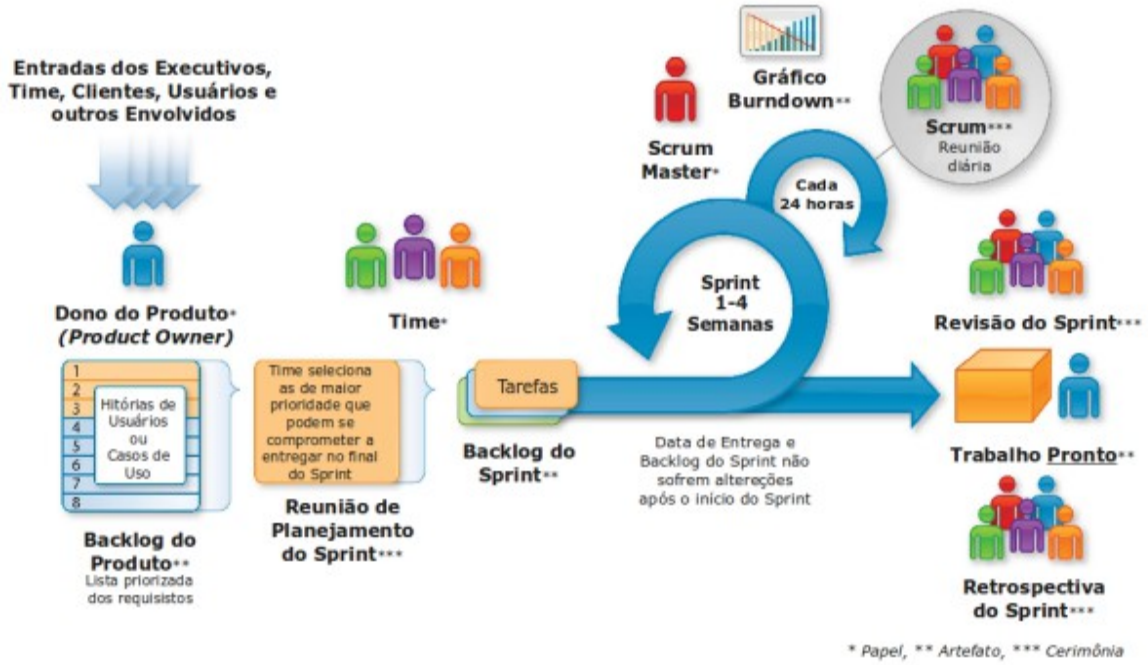

Figura 4: Ferramentas e artefatos do Scrum

Fonte: http://alanbraz.files.wordpress.com

2.7.1 Scrummaster

Segundo Brod (2013) o *Scrummaster* é um membro da equipe que é responsável por garantir o bom andamento do projeto, um coordenador geral, que faz com que as regras sejam seguidas e que as ferramentas estejam sendo utilizadas de maneira correta, a afim de trazer melhorias nos processos. Ele pode representar o *Product Owner*, quando o mesmo não esta presente. Nas metodologias ágeis o *Scrummaster* trabalha como um facilitador, já nas metodologias tradicionais um gestor de projetos é responsável por planejar, instruir e coordenar o serviço de sua equipe.

Segundo Cohn (2011), existem alguns atributos que um bom *Scrummaster* precisa ter: responsável, humilde, colaborativo, comprometido, influente, informado e informado. A melhor opção é escolher um *Scrummaster* dentro da empresa, ao invés de trazer especialistas para projetos de longo prazo, uma opção interessante é trazer um especialista para a capacitação de novos *Scrummasters*.

Algumas equipes não conseguem definir o *Scrummaster*, e optam por alternar essa função entre os membros da equipe, porém a pessoa deve estar preparada e certificada para exercer tal função.

2.7.2 Sprint Backlog e Product Backlog

Segundo Brod (2013) o *Scrum* é uma metodologia que tem como objetivo simplificar um projeto de software, por esse motivo ela é uma metodologia para projetos de curto período e de equipes pequenas, para os projetos longos é necessário a divisão dos projetos em subprojetos e as equipes muito grandes dividí-las em equipes menores, para assim executar a lista de tarefas definidas, o *Product Backlog*, no tempo determinado. No *Scrum* existe o conceito de *Sprints*, que são pequenos períodos, que podem ser de um a quatro semanas, na qual o projeto vai progredindo.

A razão pelo qual o projeto é dividido em *sprint*, é devido ao fato de não poder alterar os requisitos dentro desse período, pois em cada *sprint* uma parte do produto

é projetado, codificado e testado, qualquer alteração no requisito deve ser inserido em um novo *sprint*, para que não implique no tempo de desenvolvimento de cada *sprint*. Dessa maneira, os requisitos acabam sendo bem mais definidos, e a codificação mais refinada.

O *Product Backlog* é a lista de tarefas que são necessárias no projeto, quem define é o *Product Owner*, esta lista compõe o que o cliente deseja e suas prioridades e as tarefas necessárias para que isso seja feito, o documento tem que estar de forma mais clara possível, para que todos os membros da equipe entendam e deve estar acessível para todos. A tabela 1 mostra um exemplo de *product backlog* com os requisitos necessários para realizar o desenvolvimento do software.

Tabela 1: Product Backlog

Produto:	Sistema para uma escola de idiomas			
Data:	30/06/14			
Prioridade	Item	Descrição	Estimativa (Horas)	Pessoa
Início				
Alta	1	Levantamento dos recursos para o projeto.	160	Edvaldo
Alta	2	Contratação da equipe que atuará no projeto.	80	Edvaldo
Alta	3	Reunião inicial do projeto e definição do primeiro sprint backlog.	2	Edvaldo
Design				
Alta	4	Definição do layout, interface, levantamento das necessidades do sistema e ferramentas a serem adotadas para a realização das tarefas.	40	Edvaldo
Organização				
Alta	5	Divisão das tarefas.	2	Edvaldo
Alta	6	Compra do servidor e das ferramentas utilizadas para o desenvolvimento do sistema, instação e registro do domínio.	40	Sandra
Desenvolvimento				
Alta	7	Criação de um piloto com scripts diferenciados para a montagem do projeto de acordo com as características do mesmo.	80	Ricardo
Alta	8	Criação do sistema e conexão com banco de dados	80	Ricardo
Média	9	Criação da interdace do sistema	80	Marcos
Implantação				
Alta	10	Reunião de lançamento com o cliente e equipe de suporte.	2	Toda equipe
Média	11	Implantação do sistema na escola	40	Thomas
Suporte continuado				

Fonte: Autor

Após definido o *Product Backlog,* é hora de definir o *Sprint Backlog*, o projeto pode ser definido em vários *Sprint Backlogs* e pode ser executada por várias equipes. A tabela 2 mostra um exemplo de *sprint backlog.*

Tabela 2: Sprint Backlog

Produto:	Sistema para uma			
Sprint:	Desenvolvimento do sistema			
Período:	30 dias			
Colaborador	Tarefa	Semana	Horas Estimadas	Horas Realizadas
Ricardo	Criação da interface	1	40	35
Thomas	Aquisição dos elementos que integram o layout e se adequam ao formato padrão	1	40	50
Marcos	Organização e montagem da infra	1	40	40
Total			120	125
Ricardo	Integração do layout a interface	2	50	35
Thomas	Desenvolvimento do produto para testar junto ao cliente	2	40	40
Marcos	Criação da estrutura para customização	2	30	30
Total			120	105
Thomas	Realização de testes junto ao cliente	3	30	30
Marcos	Relatório com o feedback do cliente para adequações das funções para o sistema	3	10	10
Total			40	40
Ricardo	Correções de erros e adequação do layout de acordo com o feedback do cliente	4	40	40
Thomas	Entrega da versão funcional par ao cliente para realização de testes e futura capacitação	4	40	35
Marcos	Conclusão da estrutura solicitada pelo cliente	4	40	40
Total			120	115

Fonte: Autor

No exemplo da tabela 2, são detalhados o que cada membro da equipe vai executar no período do *sprint* e quantidade de horas estimadas e realizadas.

2.7.3 Reunião de planejamento do Scrum e reuniões diárias

Segundo Brod (2013) uma das práticas do *Scrum* são as reuniões diárias, que acontecem com todos os membros da equipe e devem ser presenciais, caso haja a impossibilidade de estar presente, pode usar de recursos como vídeo conferência. Existe também a reunião inicial onde é definido o produto desejado ao final do projeto.

Um projeto bem controlável é aquele que dura até três meses, caso ele venha a durar mais, é melhor dividí-lo em mais projetos. No *Scrum* cada *sprint* pode durar no máximo três meses, então são necessários três *sprints* para que o projeto fique pronto.

Através do *Product Backlog*, que são definidos as tarefas a serem executadas, as pessoas responsáveis e tempo gasto para execução e as suas prioridades, com estas informações já é feito a primeira reunião de planejamento do *sprint*. Nesta reunião de planejamento é necessário a presença dos gestores do projeto, do cliente e do usuário, e serão definidos as tarefas que cada um fará, as ferramentas e tecnologias utilizadas e as tarefas do *Product Backlog* que serão atendidas, com isso já é definido o *sprint* e o *sprint backlog*.

Após a reunião de planejamento o projeto já começa ser desenvolvido e as reuniões diárias já começam a acontecer, elas são curtas, em torno de 15 minutos cada, e são levantadas as seguintes questões aos membros da equipe:

1- O que foi feito ontem?

2- O que será feito hoje?

3- Quais fatores impedem seu trabalho?

O *Scrummaster* fica responsável pela eliminação desses fatores, ele tem que conduzir o projeto para que tudo flua bem. Essas reuniões são importantes porque acabam tornando outras reuniões desnecessárias.

Outro fator interessante sobre as reuniões é que elas não tem como objetivo a eliminação dos problemas, no *Scrum* não existe uma hierarquia entre os membros da equipe, então se um desenvolvedor esta com problema em um determinado assunto, ele pode tentar resolver direto com outro analista ou o próprio cliente, sem ter que ficar reportando isso sempre para o *Scrummaster*.

No final de cada *sprint* são feitas duas reuniões: a *Sprint Review*, na qual

participam o *Scrummaster*, os membros da equipe e o *Product Owner*, onde são validados todas as entregas do *sprint*; e a *Sprint Retrospective*, na qual participam o *Scrummaster* e os membros da equipe, onde é feito uma analise do que deu errado e o que pode melhorar.

Nos primeiros *sprints* sempre ocorrem erros, isso é normal, mas deve sempre levar isso como um aprendizado para novos projetos, no caso de haver impacto para o cliente depois de implementado, é necessário uma revisão do *Product Backlog*.

2.7.4 Equipe Scrum

Segundo Brod (2013) a equipe *Scrum* geralmente é composta de sete pessoas, com margem de mais ou menos dois, ou seja entre cinco à nove pessoas, com perfis multidisciplinares. Estas pessoas são capazes de gerenciar suas tarefas, elas não precisam de chefe, gerenciam-se e organizam-se entre si, e fazem os registros de seus trabalhos e entregam o produto funcional ao final de cada *sprint*, elas acionam o *Scrummaster* somente quando encontram algum obstáculo que cause impacto em seus trabalhos.

Não existe hierarquia em uma equipe *Scrum*, o *Srummaster* e responsável por fazer com que a metodologia esta sendo utilizada e um facilitador para resolver os obstáculos da equipe, no principio existe uma certa resistência da equipe em aceitar certas condições da metodologia, mas com o tempo eles mesmos percebem as vantagens em trabalhar em uma equipe *Scrum*.

Sobre a capacitação da equipe, é interessante que eles façam um treinamento de *Scrum* ou tenham uma certificação de *Scrummaster*, no Brasil existe uma grande opção de cursos práticos ótimos, pois existem muitos *Scrummasters* e *Scrum Pratictioners* aqui no país. Não existe muito segredo para encontrar esses cursos, basta você digitar no google “capacitação ou treinamento ou curso de *Scrum*” que aparecerá uma vasta opção de resultados. A figura 7 compõe a equipe *Scrum*.

Figura 5: Equipe Scrum

Fonte: http://www.cafeagile.com.br/category/scrum/

2.7.4.1 Certificação Scrum

Segundo o Scrum.org para ter uma certificação de scrummaster, basta acessar o site scrum.org e realizar o teste, são duas certificações, a primeira de *professional scrummaster* I (fundamental) custa $100,00, e o tempo máximo de prova é de 60 minutos, com perguntas de múltipla escolha, já a segunda de *professional scrummaster* II (intermediate) custa $500,00 e o tempo máximo da prova é de 120 minutos, com perguntas de múltipla escolhas e uma redação. As provas são em inglês e o material pode ser facilmente encontrado na internet, inclusive simulados, para ser aprovado é exigido uma pontuação mínima de 85%.

2.7.5 Product Owner

Segundo Brod (2013) o *Product Owner* é o cliente final do produto, no caso de outra pessoa que seja designada para o projeto *Scrum* ela vai ser o representante ou responsável pelo cliente. No final de cada *sprint*, ele que é responsável por validar as tarefas realizadas, no caso de protótipos que estejam prontos, ele fica responsável também por testar e dar o ok.

O *Product Owner* é quem paga pelo desenvolvimento do projeto, ele que dá a martelada final no final de cada *sprint*, porém o que muitas vezes acontece, o *Product Owner* acaba definindo errado uma tarefa no *Sprint Backlog*, nesses casos a equipe Scrum não deve julgar o *Product Owner* e sim alertá-lo sobre o erro, e fazer o ajuste no próximo *sprint*.

2.7.6 Burndown Chart

Segundo Brod (2013) outra ferramenta importante no *Scrum* é o *Burndown chart*, nele são colocadas as horas estimadas e as horas que foram efetivas para a conclusão do projeto, com isso podemos analisar o quanto a estimativa inicial estava certa e podemos melhorar projetos futuros. Esta ferramenta é essencial para controle de qualidade.

Simplificando, um *sprint* de quatro semanas tem 160 horas, que são consumidas ao longo das semanas de cada *sprint*, se o estimado era consumir apenas 60 horas na primeira semana e consumimos 50, temos que analisar e verificar o que aconteceu para que isso aconteça. A figura 8 mostra um exemplo do gráfico do *burndown chart.*

Tabela 3: Tabela com as horas do sprint

Semana	Horas restantes	Horas estimadas
0	400	400
1	275	280
2	170	160
3	130	120
4	15	0

Fonte: Autor

A tabela 3 mostra a quantidade de horas que são estimadas para cada semana do *sprint* e a quantidade de horas que ainda restam em cada semana para que seja feito um controle através do *burndown chart.*

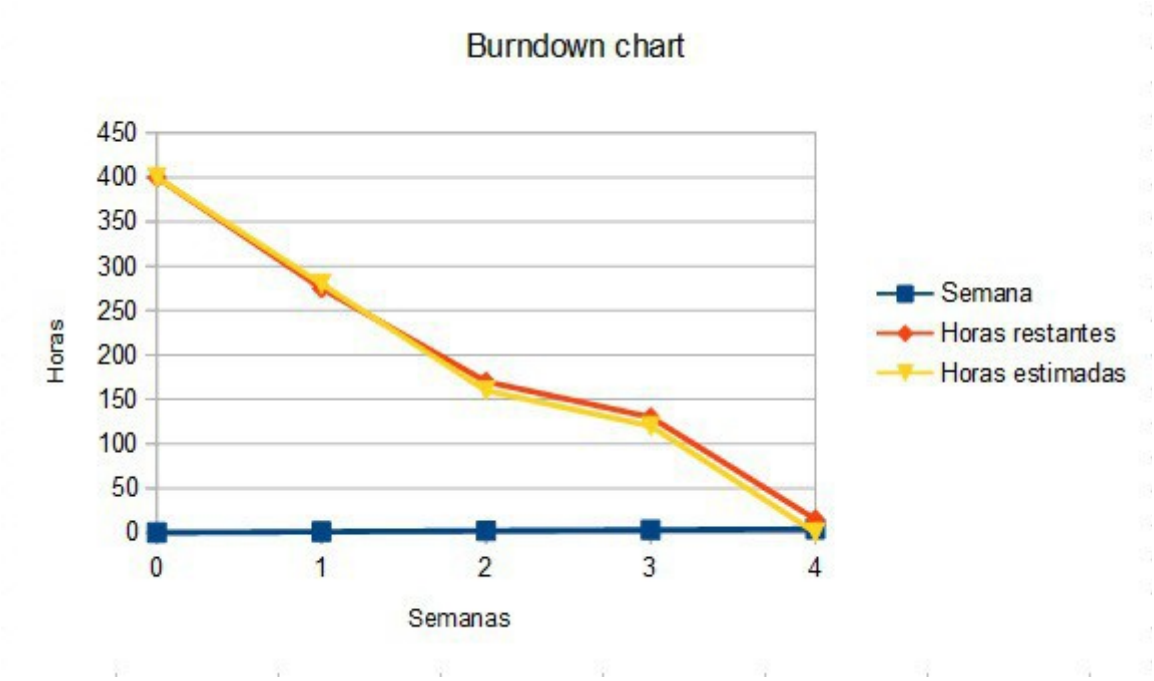

Figura 6: Gráfico Burndown chart

Fonte: Autor

2.7.7 User Stories

Segundo Brod (2013) o *User Stories* é uma ferramenta que envolve o cliente e o desenvolvedor, nela o desenvolvedor guia o cliente na hora da descrição das funções do *software*, para que tudo seja entendido de forma simples e clara. Na maioria das vezes existe um conflito de informações na hora das especificações das funções de um *software* entre o desenvolvedor e o cliente, porque o cliente não entende a linguagem técnica e o desenvolvedor domina a tecnologia a ser utilizada, mas não entende muito do negócio do cliente, por isso é importante para o desenvolvedor sentar com o cliente e o usuário. Realizar entrevistas individuais com os futuros usuários do *software* é interessante, porque muitas vezes o próprio *Product Owner* desconhece as reais necessidades.

O *User Stories* é uma ferramenta importante, nela é descrito todo o trabalho a ser executado pelo desenvolvedor toda semana, como uma história, e o usuário consegue acompanhar o desenvolvimento do projeto através do *User Story* que ele ajudou a descrever, e no final da semana consegue facilmente testar as partes que

ficaram prontas.

Cada *User Story* deve ser independente, breve e testáveis, quando o desenvolvedor auxiliar o cliente na criação das *User Stories*, ele deve ter em mente que essas histórias devem ser as tarefas que serão executadas para a criação do projeto, por isso elas devem ser bem específicas, bem definidas e independentes, caso tenha alguma que pareça não estar relacionada, ela deve ser revista. As *User Stories* devem ser criadas antes da definição do *Product Backlog*. Segue a figura 9 que representa uma *user story*.

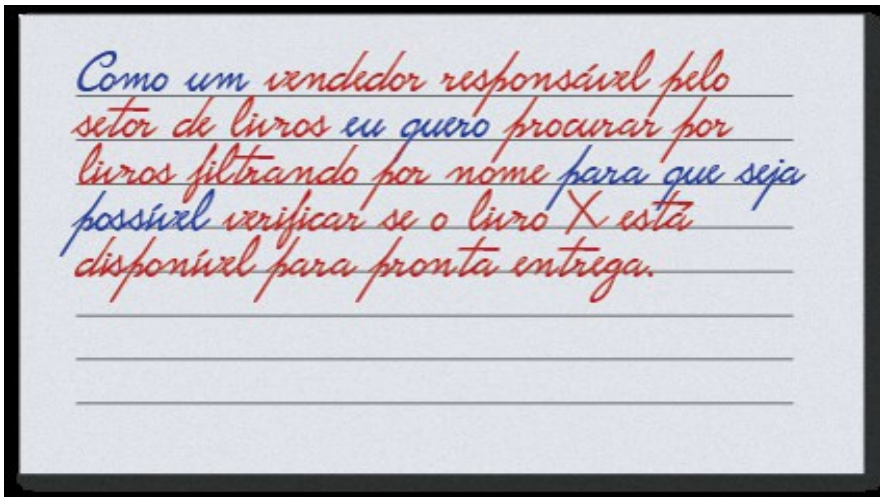

Figura 7: User story

Fonte: Fonte: http://blog.myscrumhalf.com/2011/10/user-stories-o-que-sao-como-usar/

2.7.8 Mapa Mental

Segundo Brod (2013) o uso de mapas mentais na criação do *Product Backlog* é um artefato importante, nele é possível imaginar os resultados de um projeto ou itens que irão compor no sistema, fica bem mais visível como será o projeto. Cada uma das tarefas de trabalho pode ser detalhada com mais facilidade e a equipe consegue priorizar as tarefas melhor. Segue a figura 10 que representa um mapa mental sobre o *Scrum*.

Figura 8: Mapa mental sobre Scrum

Fonte: http://www.xmind.net/m/EpTP/

2.7.9 Planning Poker

Segundo Brod (2013) o *planning poker* é uma maneira legal de entrosar a equipe e ensinar sobre o *Scrum*, e o *Scrummaster* e o *Product Owner* conseguem identificar também o perfil de cada um dos membros. O *planning poker* é utilizado para priorizar as tarefas e para realizar a estimativa do esforço da equipe para executá-las.

Há várias versões do *planning poker* e uma delas é a figura 6, que é a sequência de Fibonacci e alguns elementos estranhos no meio, que na prática funciona da seguinte maneira, é distribuído cartas para todos os membros da equipe, onde eles devem dar prioridades para cada uma das tarefas, sendo que 0 significa que a tarefa deve ser deixada de fora do projeto e ∞ significa que a tarefa é de extrema importância, a interrogação significa que a pessoa esta em dúvida em relação a prioridade da tarefa e o café significa uma pausa. Depois que todos os membros derem as prioridades para as tarefas é feito uma média e assim é definido as tarefas que terão maior prioridade no *product backlog*. A figura 11 representa um planning poker utilizado no *Scrum*.

1	2	3
5	8	13
21	0	?
1/2	∞	

Figura 9: Planning poker

Fonte: Fonte: BROD, 2013, p. 76

2.7.10 Scrum board

Segundo Brod (2013) o *Scrum board* é um quadro de avisos ou tarefas, e o modo como ele é utilizado hoje no Scrum , ele é dividido em três colunas: a fazer, em execução e feito. Em cada coluna é colocado etiquetas adesivas com as tarefas e suas anotações, e a cada etapa concluída as etiquetas vão avançando. A figura 12 mostra um exemplo de *Scrum board.*

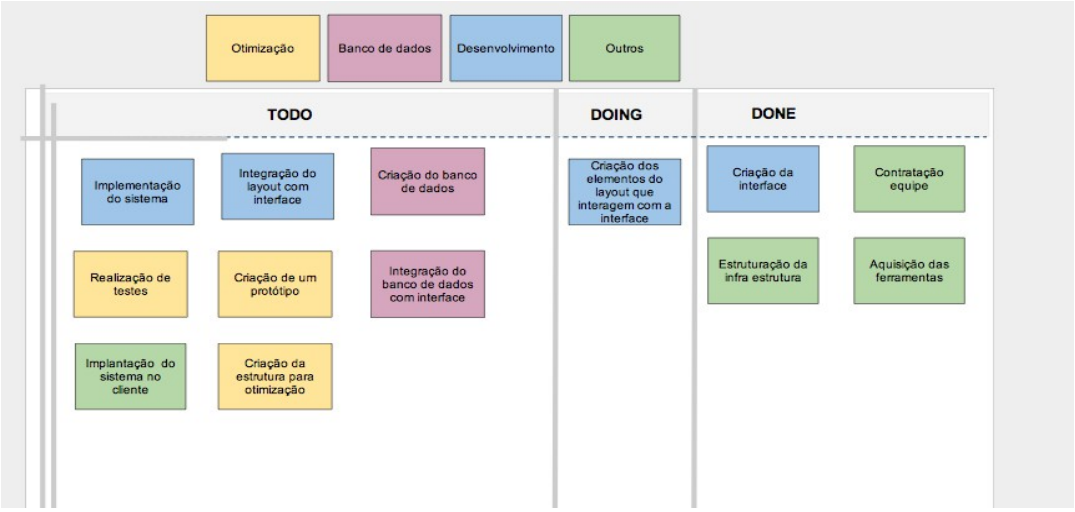

Figura 10: Scrum Board

Fonte: Autor

2.8 OS CÁLCULOS DOS CUSTOS DO PROJETO

Segundo Pham e Pham (2011) para convencer o cliente a utilizar o *Scrum* em um projeto, você precisa mais que argumentos, você precisa de números que comprovem sua eficiência. Devemos começar calculando o custo do projeto, e para isso é necessário saber a velocidade da equipe, que é o número de *user stories* ou *product backlog* em formas de pontos, que equipe consegue entregar durante um *sprint*.

Como exemplo vamos deduzir que a velocidade da equipe de um projeto seja de 20 pontos de *user stories* por *sprint*, sendo que cada *sprint* tem 4 semanas, ou seja o projeto esta com 160 pontos, com isso saberemos que a equipe levará 8 *sprints*, ou 32 semanas para concluir o projeto. Assumindo que os custos do projeto sejam de R$150.000,00 por ano com salários e benefícios, o custo da equipe desse projeto vai ser de R$92.308,00 ((R$150.000,00 x 32 semanas)/52 semanas).

Somando os custos com equipamentos de computador e telecomunicação, é possível obter o custo total do projeto.

2.8.1 O investimento do projeto

Segundo Pham e Pham (2011) com o custo do projeto em mãos é mais fácil calcular o investimento do projeto e deixar que o cliente decida se vai investir nesse projeto com *Scrum* ou não.

Alguns dos jeitos de calcular o retorno no investimento mais utilizados são o *payback*, comprar versus construir, valor presente líquido (VLP) e o retorno sobre investimento (ROI, *Return on investiment*).

- Período de *payback*: conhecido também como ponto de equilibrio, o período de *payback* é o tempo gasto para recuperar o dinheiro investido. Pega o custo total gasto para o projeto e é feito uma análise com projeções do fluxo de caixa que a empresa terá nos próximos meses. Como exemplo, uma empresa

que gastou cerca de R$200.000,00 com o custo de um projeto de *software*, e que conforme projeções o seu fluxo de caixa trimestral será de R$28.572,00. O cálculo será R$ 200.000,00 / R$28.572,00 = 7 trimestres, ou seja será necessário 7 trimestres ou 21 meses para a empresa recuperar o dinheiro investido.

- Comprar versus Construir: deve seguir essa técnica da seguinte maneira, primeiro deve-se calcular a diferença do preço fixado, subtraindo o preço fixado para construir e o preço fixado para comprar, chamando-o de diferença de preço. Depois deve-se calcular a diferença do gasto mensal, subtraindo o gasto mensal de compra pelo gasto mensal de construção. E finalmente deve-se calcular o número de períodos de tempo, dividindo a diferença de preço pela diferença de gasto mensal.

  Como exemplo imagine-se dois projetos, A e B, onde A tem o custo fixado de construção é de R$1.000.000 e o gasto mensal de construção é R$50.000 e o B tem custo fixado de compra de R$900.000 e o gasto mensal de compra é de R$100.000. Utilizando as fórmulas tem o seguinte:

  Diferença de preço = custo fixado de construção - custo fixado de compra = R$1.000.000 - R$ 900.000 = R$ 100.000

  Diferença de gasto mensal = gasto mensal de compra - gasto mensal de construção = R$100.000 - R$50.000 = R$50.000

  Agora dividindo R$100.000 por R$50.000, tem-se 2 meses que é o ponto que ambos terão o mesmo custo (custo fixado + gastos mensais agregados).

  Portanto para manter um *software* por mais de 2 meses a melhor opção seria comprar o *software*, pois o custo total de compra (R$900.000 + R$100.000 = R$1.000.000) é menor que o custo total de compra (R$1.000.000 + R$50.000 = R$1.050.000), porém como na maioria dos casos, a empresa resolve manter o *software* por mais de 2 meses então, a melhor opção seria construir o *software*, pois o custo total da construção (R$1.000.000 + R$50.000 + R$50.000 + R$50.000 = R$1.150.000) é menor que o custo total de compra (R$900.000 + R$100.000 + R$100.000 + R$100.000 = R$1.200.000).

- Valor presente líquido (VPL): para entender melhor o VPL, tem que saber o VP, que é uma técnica mais aperfeiçoada que leva em consideração o valor temporal do dinheiro, pois uma quantia que receberá futuramente, vale menos que a mesma quantia hoje.

VP é calculado da seguinte maneira : $VP = VF/(1 + i)^n$, sendo VP valor presente, VF valor futuro, i taxa de inflação ou interessante e n número de períodos, onde o interesse é pago. Como exemplo pega-se o valor de R$4.000, e queremos saber qual é será seu valor daqui a 3 anos, com uma taxa de inflação ou interesse de 5%. O resultado é o valor de R$4.630,50 que é o produto R$4.000 multiplicado por $(1,05)^3$, ou R$4.000 multiplicado por (1,157625).

O VPL é o valor atual da receita total subtraindo o valor atual do custo de investimento durante um determinado tempo, onde tem as seguinte situações para escolher.

- Caso o VPL de um projeto seja maior que 0, o projeto deve ser aceito.
- Caso o VPL de um projeto seja menor que 0, o projeto deve ser negado.
- Em casos, onde houver mais de um projeto, escolher o com VPL maior.

Como exemplo um projeto com interesse de 10% durante um período de 3 anos com as estimadas receitas de R$25.000, R$100.000 e R$200.000 respectivamente para primeiro, segundo e terceiro ano e um custo anual do projeto de R$100.000.

Diferença do VPL = VPL da receita - VPL do custo do projeto

Diferença do VPL = R$252.046 - R$281.654 = - R$29.068

Como neste caso o VPL deu negativo, esse projeto deveria ser recusado.

- ROI (*Return on investiment*): calcular o ROI é um pouco complexo, mas de forma simples é necessário saber a velocidade do projeto e a margem de lucro, geralmente o departamento financeiro que cuida disso, e caso precise eles provavelmente fornecerão, a fórmula do ROI é velocidade x margem. Como exemplo um projeto que possa gerar de lucro R$4 milhões e um custo de investimento de R$2 milhões, com margem de lucro da empresa de 10%, então utilizando a fórmula tem-se a seguinte resposta:

  ROI = (4.000.000 / 2.000.000) x 10% = 20%

  Comparando com outros projetos, devemos escolher o projeto que tem o ROI mais alto.

2.9 FERRAMENTAS

Segundo Brod (2013) para a prática do *Scrum* não são necessárias muitas ferramentas, nada mais que um quadro branco e etiquetas adesivas são utilizadas. Porém para existem algumas opções de ferramentas que são muito utilizadas.

O quadro branco ou parede branca é a principal ferramenta, pois nela são coladas as etiquetas com as *users stories* que se movem entre *sprint backlog*, em execução e pronto. O quadro branco também é utilizado nas reuniões diárias, na qual os participantes apresentam o que estão fazendo e sua evolução. O *burndown chart* também deve estar no quadro branco ou próximo, para que todos vejam as atividades e o seu progresso. No quadro branco deve ter um espaço para avisos e outras anotações.

As etiquetas são elementos essenciais no *Scrum*, são utilizadas desde a definição do *product backlog*, a priorização das atividades e dos recursos até a exeução das atividades. é interessante o uso de etiquetas coloridas para que possa haver uma identificação das tarefas, pessoas, grau de importância, entre outros.

As ferramentas Google são muito importante no caso dos participantes que não podem estar presentes nas reuniões, através do *Drive* as pessoas podem colocar todo material das reuniões diárias, e as pessoas podem acompanhar de qualquer lugar, assim como podem editar também, no *Drive* é possível encontrar *templates* prontos, basta acessar o site (*https://drive.google.com/templates#*) e procurar por *Scrum*, *product backlog* e *sprint backlog*.

O *Hangout* é outra ferramenta do Google, utilizada para realizar vídeoconferência, onde as pessoas que não estejam fisicamente no local, conseguem participar das reuniões diárias.

O *Murally* (*http://mural.ly*) visa substituir o quadro branco com as etiquedas por um ambiente virtual que tem o mesmo jeito, com ele você pode criar salas e nelas colocar o *backlog*, e outros documentos. Com a versão gratuita você pode criar somente três salas, a partir desse número é necessário adquirir o pacote pago.

O *Virtual Kanban* (*http://virtualkanban.net/*) é uma ferramenta utilizada somente para criação de *Scrum board*, na qual você pode acompanhar o progresso das tarefas realizadas em cada *sprint*. Sua vantagem é a simplicidade, pois o projeto é composto somente de um arquivo que pode ser salvo no seu computador, na rede

da empresa ou em um servidor *web*.

O *Scrinch* é uma ferramenta que ajuda no controle e implantação dos processos do *Scrum*, ele pode ser encontrado através do site (http://sourceforge.net/projects/scrinch/), ele possui todos os artefatos e controles de processo do *Scrum*.

2.10 O PRODUTO FINAL

Segundo Brod (2013) a definição de pronto seria “pronto para implantar no ambiente de produção”, porém o pronto pode ter outras definições de acordo com a situação, isso vai depender muito do tipo de projeto e do acordo que foi estabelecido no fechamento do contrato.

Alguns projetos são eternos, sempre existem alterações para serem feitas, nesses casos os softwares prontos tem versões, como versão1, versão2, etc.

2.10.1 Testes

Segundo Pham e Pham (2011) no *Scrum*, os testes são realizados ao longo das iterações e *sprints*, se a empresa tiver uma infraestrutura com testes automatizados, mais rápido e eficiente será realizados os testes. Os testes são parte do processo de desenvolvimento e não são realizados somente quando o produto estiver “pronto”.

Como o *Scrum* é uma metodologia voltada para gerenciamento de projetos, ela não diz nada sobre os testes, mas existem dois tipos de testes que melhoram a prática do *Scrum*:

- Testes automatizados: são feitos de forma rápida, com ele a equipe *Scrum* conseguirá manter um ritmo acelerado, mas com muita suavidade, nele um

software executa todos os testes de forma automatizada. É necessário adquirir ferramentas específicas para isso, e no primeiro momento deve ser criado todos os casos de testes, porém isto é uma questão de tempo, a equipe *Scrum* ganha muito mais tempo do que realizar tudo isso manualmente.

- Testes de integração contínua: este teste é muito importante também, porque a equipe se certifica que o *software* esta sempre em um estado disponível para a entrega.

A empresa também deve estar com uma infraestrutura preparada para realizar os testes, existem três ambientes que devem estar interligados: desenvolvimento, testes e produção.

Segundo Cohn (2011) para realizar os testes depois que o produto estiver pronto não funciona mais, porque é mais dificil melhorar a qualidade de um produto depois que ele já estiver pronto, isso é o que acontece quando é testado um *software* depois de pronto e é necessário que corrigir os erros. Com os testes somente no final do projeto os erros acabam não sendo notados e eles são cometidos repetidamente. Outro problema encontrado com a realização dos testes no final do projeto é que muito *feedback* acaba sendo perdido, pois é difícil identificar os problemas que podem ter sido causados meses atrás. Devido atrasos durante o desenvolvimento do projeto e a cronogramas, muitas vezes o teste acaba sendo cortado ou feito de qualquer jeito, quando é executado no final do projeto.

2.10.2 Qualidade

Para Cohn (2011) a qualidade no *Scrum* esta relacionada aos testes, pois são através deles que é identificado os erros, fazendo que o *software* seja entregue com qualidade. Os produtos que não são entregues com qualidade, significa que os testes não foram executados ou não foram executados com os níveis certos.

A qualidade esta relacionada a equipe também, pois uma equipe boa equipe *Scrum* procura sempre novas habilidades de testes e maneiras de introduzi-las dentro prazos rigorosos do *Scrum*. A equipe também deverá trabalhar em conjunto visando um objetivo comum que é a entrega do *software* com qualidade, o programador e o testador devem estar sempre se comunicando, e a quantidade de testes deve ser a mesma tanto no primeiro quanto no último dia de cada *sprint*.

3 ESTUDO COMPARATIVO

O estudo comparativo deste trabalho tem como objetivo comparar o uso da metodologia *Scrum* e o modelo Clássico para o desenvolvimento de *software*.

3.1 SCRUM X MODELO CLÁSSICO

Devido ao grande ao crescimento do número de empresas que estão utilizando a metologia ágil *Scrum* para o desenvolvimento de *software*, foi feito uma comparação das vantagens e desvantagens entre o *Scrum* e o modelo Clássico mostrando a eficiência da metodologia *Scrum*.

3.1.1 Vantagens e Desvantagens do Scrum

As vantagens identificadas em relação ao Scrum foram:

- Integração do usuário: O usuário participa ativamente no projeto, participando de reuniões para a definição dos requisitos no *product backlog*, criando as *user stories*, no final de cada *sprint* para validar as tarefas que foram executadas, testar e avaliar os protótipos e no final do projeto quando o projeto já esta pronto para implantação.
- Baixo Custo: Os custos com as ferramentas são menores, todos as ferramentas utilizadas para o uso da metodologia *Scrum* é *open source*, não é necessário pagar licensa para utilizar. Apesar de exisitir várias opções de ferramentas que tem funcionalidades melhor, que são pagas, a empresa pode escolher as ferramentas free, e conseguirá executar todas as funções da metodologia da mesma maneira. Devido a equipe ser menor e multidisciplinares acaba gerando um custo menor para a empresa.
- Maior interação do grupo de desenvolvimento: Por se tratar de uma equipe pequena, onde os membros são multidisciplinares e se auto gerenciam, e

devido as reuniões diárias, a comunicação entre os participantes da equipe flue melhor, devido ao objetivo comum da equipe, todos conhecem suas responsabilidades e trabalham em conjunto para que o projeto seja entregue com sucesso.

- É dinâmico e não estático: A interação com o cliente e equipe, faz com que o projeto seja dinânimo, tudo acontece de forma rápida.
- Integração maior da equipe para a resolução do problemas: Devido as reuniões diárias e exigência de que os membros estejam fisicamente no mesmo local, é muito mais fácil corrigir os erros identificados no projeto. Assim que os problemas são identificados pelos membros da equipe, eles já são apontados na próxima reunião diária, e os próprios membros devem discutir o que deve ser feito para correção, e podem sentar juntos para trabalharem na correção do problema, e caso encontrem alguma barreira devem reportar para o *Scrummaster* para que ele corra atrás resolver.
- Parte do projeto entregue ao final de cada sprint: Com um protótipo entregue ao final de cada *sprint*, fica muito mais fácil receber *feedbacks* do cliente, podendo identificar o que esta errado e deve ser consertado, o que esta certo e o que pode ser melhorado na próxima *sprint*.
- Foco na equipe de desenvolvimento: A metologia *Scrum* preza muito os funcionários, os sprints são planejados com as atividades e com os horários que podem ser cumpridos, e não com prazos absurdos que não podem ser cumpridos. No *Scrum* é esperado que tudo seja cumprido conforme determinado, por isso existe as reuniões diárias e o *burndown chart*, para que não haja atrasos, de forma a não exigir que os funcionários façam quantidade de horas extras absurdas, causando um tremendo desgaste.

As desvantagens identificadas em relação ao *Scrum* foram:

- A resistência da equipe para a utilização da metodologia: Caso seja a primeira vez que a equipe vai trabalhar com o *Scrum*, existe sempre uma resistência por parte da equipe em querer utilizar a metodologia, o fato de muitos não conhecerem acaba gerando um certo medo e dúvida na equipe se o projeto vai dar certo ou não.
- A necessidade da equipe estar no mesmo local: Apesar das vantagens da equipe estar presente no mesmo local, muitas empresas não aceitam muito

bem essa exigência, pois tem funcionários que trabalham em filiais diferentes, ou pessoas que trabalham *home office*.

- A presença da equipe nas reuniões diárias: Imprevistos podem acontecer a todo momento, e a necessidade da presença de todos os membros da equipe todos os dias acaba pode gerar um certo desentendimento na equipe. As reuniões são realizadas todos os dias de manhã, e atrasos e imprevistos podem comprometer esse artefato.
- Custo com treinamentos: Caso seja o primeiro projeto da equipe com *Scrum*, a empresa deve realizar pelo menos um treinamento para que a equipe conheça a metodologia, apesar de ser necessário somente para o primeiro projeto, muitas empresas entendem isso como um custo adicional.
- Falta de documentação: O foco principal da metodologia não é a documentação, porém muitas empresas veem isso como amadorismo.

3.1.2 Vantagens e desvantagens do modelo Clássico

As vantagens identificadas em relação ao modelo Clássico:

- Serve como referência para as empresas: Por ser o modelo mais utilizado nas empresas, ele acaba servindo como referência
- Documentação Abrangente: Todos os processos são documentados, além dos documentos escritos, existem os diagramas que ajudam a entender melhor o projeto.

As desvantagens identificadas em relação ao modelo Clássico

- A interação com o usuário: A interação com o usuário é muito limitada, o contato é feito no início do projeto, quando o usuário define os requisitos para o projeto e no final, quando o *software* já esta pronto para implantação. Dependendo do período do projeto, um requisito pode acabar sendo desnecessário, com os avanços tecnológicos, o que pode funcionar bem hoje, talvez se torne obsoleto meses depois.
- O usuário só vê o *software* no final do projeto: Caso o usuário não tenha especificado um requisito corretamente, ele só irá verificar quando o software

for implantado, o que pode gerar certo desconforto. Os requisitos não podem ser alterados.

- Alto custo: Alto custo com a modelagem, as ferramentas utilizadas para a modelagem necessitam de licença. O custo com a equipe é maior, devido ao trabalho específico ser executado por professionais específicos, como arquitetos, analistas, programadores, gestores de qualidade, entre outros.
- Problemas para cumprir os prazos: Se uma etapa atrasar, irá atrasar todo o projeto, devido ao fato de uma etapa só começar quando a etapa anterior estiver pronta. É muito difícil cumprir com os prazos determinados.
- Problemas com a interação da equipe: Cada um é responsável por sua função, então não há uma interação entre a equipe. Somente o gerente do projeto tem uma visão geral do *software,* e quando existe um problema, ele acaba sendo o responsável, já que cada time é responsável por sua tarefa e não pelo projeto como um todo.
- Custo de manutenção: Se o requisito não for interpretado, conforme a real necessidade do usuário, a falha no sistema será identificada somente na fase de implementação. Desta forma, leva-se um tempo maior para identificar um problema no desenvolvimento do software, o que proporciona um maior custo de manutenção.

3.2 ANÁLISE

Através da análise feita foi possível identificar que a metodologia ágil *Scrum* tem mostrado melhor resultado em comparação ao modelo Clássico em relação a custo, qualidade e cumprimento dos prazos.

O custo de um projeto utilizando a metodologia *Scrum* é menor do que de um modelo Clássico, devido ao fato do *Scrum* não utilizar ferramentas específicas para a modelagem, o que não gera custo com licenças caras. O custo com os professionais é menor, já que um projeto utilizando o modelo Clássico necessita de professionais específicos para cada atividade, ao contrário dos professionais de um projeto *Scrum* que são multidisciplinares. Apesar do custo com treinamento sobre *Scrum* para os membros do projeto, a empresa gastará somente no primeiro projeto,

e olhando futuramente, para os novos funcionários que entrarem para os projetos que utilizarão a metodologia, porém o valor gasto com os treinamentos pode ser recuperado com a redução dos gastos utilizando a metodologia.

O *Scrum* leva vantagem em relação a qualidade também, pois devido a grande interação entre o cliente e a equipe, é possível entender melhor os requisitos. A entrega de protótipos para o cliente testar e avaliar ao final de cada sprint ajuda identificar as falhas e fazer as correções para o próximo *sprint*. No modelo Clássico a interação com o cliente é feito somente no começo e no final, então se algum requisito não foi bem especificado, a equipe só irá saber no final na hora da implantação, o que acaba comprometendo a qualidade de um projeto utilizando o modelo Clássico.

Em relação ao cumprimento dos prazos, em um projeto utilizando a metodologia Scrum é possível identificar através das reuniões diárias, quando há um problema, e devido a interação da equipe ser melhor, é possível corrigi-lo sem atrasar o cronograma. No Scrum é possível acompanhar o andamento das atividades através do *burndown chart*, o que ajuda identificar quando há atrasos. Em um projeto utilizando o modelo Clássico, uma etapa só é iniciada quando a outra termina, se uma etapa atrasa, todo o projeto terá consequências no seu cronograma de planejamento. Devido as pessoas serem específicas para cada tarefa, não existe uma interação geral com a equipe, por esses problemas de comunicação, um problema pode demorar mais para ser resolvido, atrasando o cronograma. Muitas empresas determinam um cronograma, e para que ele seja cumprido, quando existem problemas que atrasam algumas etapas, as últimas etapas acabam ficando prejudicadas, sendo necessário a realização de muitas horas extras, aumento o custo do projeto e desgastando os funcionários.

Um fator que o modelo Clássico leva vantagem em relação ao *Scrum* é a credibilidade, o fato da metodologia ser nova, acaba gerando uma certa dúvida nos membros da equipe em relação ao sucesso do projeto, e criam uma certa resistência em aderir a metodologia. As empresas também tem dúvida sobre a metodologia, já que o foco dela não é documentação e como a metodologia é utilizada para pequenos projetos, as empresas acham difícil comparar, já que o modelo Clássico atende todas as necessidades. O modelo Clássico é o mais utilizado e serve de referência para outros modelos de desenvolvimento de *software.*

A metodologia *Scrum* vem crescendo e ganhando muito espaço no mercado

de desenvolvimento, porém ainda tem que melhorar algumas coisas e eliminar algumas barreiras que foram impostas pelo mercado, isso sem torná-la uma metodologia pesada, como o modelo Clássico, um dos desafios é ser utilizada em projetos maiores e para grandes empresas, outro desafio é a comunicação entre a equipe, apesar de ser um fator positivo para a metodologia, existe sempre a necessidade de reuniões diárias e que o time esteja fisicamente no mesmo local, o desafio seria manter a comunicação para times que estejam separados geograficamente, que acontece muito no caso de grandes empresas, assim a metodologia poderá ganhar melhor visibilidade entre elas.

4 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O principal motivo pelo grande número de empresas utilizarem o *Scrum* como metodologia para desenvolvimento de *software*, se deve ao fato das metodologias tradicionais serem consideradas pesadas e não estarem funcionando tão bem nos dias de hoje, devido ao alto custo com ferramentas, prazos não cumpridos, entre outros problemas. Através da análise feito no estudo comparativo deste trabalho, foi possível identificar as vantagens do uso da metodologia *Scrum* em um projeto de *software* em relação ao modelo Clássico.

A metodologia *Scrum* é mais adequada para projetos pequenos, pois é descartado toda aquela parte de diagramas e documentação, o que torna um projeto pesado e foca mais no projeto, na execução e na qualidade. Com o *Scrum* ficou mais fácil identificar o andamento do projeto e cumprir os prazos, o que no modelo Clássico fica mais difícil cumprir o cronograma, pois não existe um acompanhmento diário das atividades que estão sendo executadas.

Um outro fator que podemos identificar foi o baixo custo, para a utilização da metodologia *Scrum* é necessário somente um quadro branco, post-its e o baralho, o restante das ferramentas utilizadas para criação do *product backlog, sprint backlog* e *burndown chart* é *open source*, não sendo necessário a compra de licenças caras para ferramentas para criação de diagramas e documentação como no modelo Clássico.

Uma outra vantagem do uso do *Scrum*, é a comunicação entre o time, devido as reuniões diárias, fica mais fácil acompanhar o progresso do projeto, o que cada participante esta fazendo e os problemas encontrados. As reuniões no final de cada *sprint* com o *product owner* ajudam a identificar o que o cliente esta achando, e caso seja identificados problemas, eles podem ser corrigidos na proxima *sprint*, melhorando a qualidade do produto.

Apesar de muitas vantagens do *Scrum* que identificamos com o estudo comparativo, existem alguns problemas também. As *daily meetings* são um fator importante para o projeto, porém é necessário a participação de todos os membros da equipe, o não comparecimento de algum membro acaba prejudicando o projeto, e devido à vários fatores, é complicado que todos os membros participem de todas as reuniões todos os dias.

De um modo geral, a metodologia *Scrum* atingiu todas as expectativas deste projeto, apesar de alguns pontos que podem melhorar, ela conseguiu atender todas as necessidades da equipe e do cliente com qualidade e no prazo determinado.

Para trabalhos futuros é sugerido a utilização da metodologia *Scrum* em novos projetos de ambientes corporativos. Os problemas encontrados no estudo comparativo, poderão ser corrigidos e gerar artefatos a serem utilizados, focando principalmente na satisfação do usuário e redução de custos com manutenção nos sistemas.

A utilização do *Scrum* em futuros projetos é uma ótima opção para melhora contínua nos projetos da empresa, e com os resultados obtidos a empresa poderá também divulgar a metodologia para outras empresas.

REFERÊNCIA BIBLIOGRÁFICA

BROD, Cesar. **Scrum**: Guia prático para projetos ágeis. 1º. ed. São Paulo: Novatec, 2013.

COHN, Mike. **Desenvolvimento de software com Scrum**: Aplicando métodos ágeis com sucesso. 1º. ed. Porto Alegre: Bookman, 2011.

Equipe Scrum, Disponível em: <http://www.cafeagile.com.br/category/scrum/> Acesso em: 10 MAI 2014 10h46

Ferramentas e artefatos do Scrum, Disponível em: <http://alanbraz.files.wordpress.com> Acesso em: 10 MAI 2014 13h05

KOSCIANSKI, Andre; SOARES, Michael Dos Santos. **Qualidade de software**. 2º. ed. São Paulo: Novatec, 2007.

LAKATOS, E. M.; MARCONI, M. A.: **Fundamentos de Metodologia Científica**. Disponível em: <http://www.webartigos.com/artigos/conceitos-em-pesquisa-cientifica/10409/> . Acesso em 04 JAN 2014 10h23

LAUDON, Kenneth; LAUDON, Jane. **Sistemas de informação gerenciais**. 9º. ed. São Paulo: Pearson, 1995.

LIJPHART, Arend: Método Comparativo. Disponível em: <http://pt.scribd.com/doc/59014737/Metodo-Comparativo>. Acesso em 01 JUL 2014 20h15

Mapa mental sobre o Scrum, Disponível em: <http://www.xmind.net/m/EpTP/> Acesso em: 25 MAI 2014 09h20

MINAYO, M. C. **O desafio do conhecimento: pesquisa qualitativa em saúde**. Disponível em: <http://www.webartigos.com/artigos/conceitos-em-pesquisa-cientifica/10409/> . Acesso em 04 JAN 2014 10h03

PHAM, Andrew; PHAM, Phuong-Van. **Scrum em ação**: Gerenciamnto e desenvolvimento ágil de projetos de software. 1º. ed. São Paulo: Novatec, 2011.

PMTECH, Capacitação em projetos, Disponível em: <http://www.pmtech.com.br/scrum_certificacao.html> Acesso em: 23 FEV 2014 17h30

PRESSMAN, Roger S.. **Engenharia de software**. 3º. ed. São Paulo: Makron Books, 1995.

SCRUM.ORG, Improving of profession of software development. Disponível em: < https://www.scrum.org/> Acesso em: 23 FEV 2014 15h43

SOMMERVILLE, Iam. **Engenharia de software**. 8º. ed. São Paulo: Pearson, 2007.

TURBAN, Efraim; JR., R. Kelly Rainer,; POTTER, Richard E.. **Introdução a sistemas de informação**: Uma abordagem gerencial. 4º. ed. Rio de Janeiro: Elsevier, 2007.

User story, Disponível em: <http://blog.myscrumhalf.com/2011/10/user-stories-o-que-sao-como-usar/> Acesso em 15 ABR 2014 11h15

Visão geral do Scrum, Disponível em: <http://www.devmedia.com.br/desenvolvimento-agil-com-scrum-uma-visao-geral/26343> Acesso em: 25 JAN 2014 09h55

YOURDON, Edward. **Análise estruturada moderna**. 3º . ed. Rio de Janeiro: Campus, 1990.