Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

METODOLOGIA ÁGIL – UTILIZAÇÃO DO *SCRUM* NO DESENVOLVIMENTO DE SOFTWARE

FILIPE CHRISTOVO DA SILVA

Americana, SP
2013

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

METODOLOGIA ÁGIL - UTILIZAÇÃO DO *SCRUM* NO DESENVOLVIMENTO DE SOFTWARE

FILIPE CHRISTOVO DA SILVA

filipe_ads@hotmail.com

Trabalho de Graduação desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor, Me. Anderson Luiz Barbosa.

Área: Análise e Desenvolvimento de Sistemas

Americana, SP
2013

FICHA CATALOGRÁFICA elaborada pela
BIBLIOTECA – FATEC Americana – CEETPS

S58m	Silva, Filipe Christovo da Metodologia ágil: utilização do Scrum no desenvolvimento de software. / Filipe Christovo da Silva. – Americana: 2013. 53f. Monografia (Graduação de Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza. Orientador: Prof. Me. Anderson Luiz Barbosa 1. Software de gerenciamento de projetos 2. Administração de projetos I. Barbosa, Anderson Luiz II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana. CDU: 681.3.077 658.511-4

Bibliotecária responsável Ana Valquiria Niaradi – CRB-8 região 6203

BANCA EXAMINADORA

Prof. Me. Anderson Luiz Barbosa

Prof. Me. Gabriel de Souza Fedel

Prof. Me. Kleber de Oliveira Andrade

AGRADECIMENTOS

Primeiramente agradeço a Deus, que plantou o sonho em mim e me deu forças para buscar e conquistar todos meus objetivos, sem ele talvez não tivesse força para aguentar toda essa trajetória.

Agradeço muito ao meu orientador e mestre Anderson Luiz Barbosa que me apoio e apostou sua confiança em mim para o desenvolvimento desse trabalho. Nas orientações me forneceu todo o apoio necessário sempre com dedicação e paciência para apontar os erros e apontar melhorias. Tornou-se uma influência para mim nos rumos profissionais que quero seguir depois da faculdade.

Gostaria de agradecer aos professores que compartilharam seus conhecimentos e ajudaram na minha formação e a me tornar um profissional ético e responsável.

Agradeço meus familiares, destacando meu pai, mãe e namorada, pois essas três pessoas foram quem me deram mais apoio durante toda a formação acadêmica, sempre fazendo o possível e impossível para me ajudar.

Por fim, porém não menos importante, todos os meus colegas de curso, principalmente ao Diego e Dionis que estiveram comigo desde o início e com um companheirismo maior desde o terceiro semestre. Foi com os dois que dividi dias de estudo e muitos trabalhos no decorrer do curso.

DEDICATÓRIA

Dedico esse trabalho a minha namorada, devido ao apoio durante toda minha trajetória acadêmica e pelo exemplo de perseverança e companheirismo que sempre me mostrou.

RESUMO

Este trabalho apresenta um estudo sobre a metodologia ágil de desenvolvimento, *Scrum*. É um breve estudo das metodologias ágeis em geral e uma especificação detalhada dos papeis, regras e práticas da metodologia *Scrum*. Além disso, apresenta um estudo de caso, sobre um desenvolvimento de um software sem a utilização de nenhuma metodologia ágil. No mesmo estudo de caso apresenta-se o desenvolvimento utilizando os conceitos do *Scrum,* para demonstrar que com a utilização da metodologia o desempenho da equipe e o produto final tem um resultado mais satisfatório.

Palavras Chave: Engenharia de Software; Metodologia Ágil; *Scrum*;

ABSTRACT

This paper presents a study on the Agile Scrum development. It is a brief study of agile methodologies in general and a detailed specification of the roles, rules and practices of Scrum. This is also presents a case study about development of software without the use of any agile methodology. The same case study will present the development using the concepts of Scrum, to demonstrate that the use of the methodology team performance and the final product has a more satisfactory result.

Keywords: *Software Engineering; Agile Methodology; Scrum;*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

PO – *Product Owner*

PB – *Product Backlog*

SM – *Scrum Master*

TI – *Tecnologia da Informação*

XP – *Extreme Programming*

CASE – *Computer Aided Software Engineering*

1 INTRODUÇÃO

Todo software, quando requisitado, deve atender de forma eficaz às necessidades do cliente. Entretanto, não era isso que se via há alguns anos atrás, pois os softwares desenvolvidos estavam abaixo do desejado pelo mercado e do que necessitavam os clientes.

Existiam problemas constantes com o cliente, pois, geralmente o próprio, não sabia exatamente qual a sua necessidade. Hoje o mercado, devido a diversos fatores, está sujeito a sofrer grandes mudanças, consequentemente, afeta no desenvolvimento de software. Essas mudanças podem afetar diretamente os requisitos e os mesmos se tornam incertos e sempre sofrem alterações durante todo o processo de desenvolvimento. Dessa forma, os softwares que atendiam de uma maneira eficaz às necessidades dos clientes eram apenas uma pequena porcentagem no meio de muitos desenvolvidos.

Diversos problemas foram identificados, como: erro no entendimento e mudanças de requisitos, alteração de cronograma, variação no orçamento inicial do projeto, cansaço dos profissionais por excesso de retrabalho para corrigir falhas e resultados insatisfatórios para o cliente.

> “Projetos importantes apresentavam, algumas vezes, anos de atraso. O software, cujo custo superava as previsões, não era tão confiável, era difícil de manter e seu desenvolvimento era insatisfatório.” (SOMMERVILLE, 2007, pág. 5).

Diante desses problemas e a necessidade da criação de softwares eficazes e aptos à mudança, surgiram às metodologias ágeis. Atualmente existem diversas metodologias e uma delas é *Scrum*, a qual será o tema dessa monografia.

Scrum é uma metodologia ágil, foi formalizada em 1995, por Ken Schwaber, um desenvolvedor de softwares, que também ajudou na sua implantação nos desenvolvimentos. (RODRIGUES, 2007, pág. 5).

A metodologia ágil *Scrum* proporciona benefícios aos projetos, como:

- Agilizar o processo de desenvolvimento;
- Aumentar o desempenho da equipe e diminuir cansaço por excesso de retrabalho;
- Evitar surpresas com resultados;
- Atender completamente as necessidades dos clientes.

No entanto apesar de ser muito usada para desenvolvimento de softwares, para SCHWABER (2009, pág. 3) essa metodologia pode também ser aplicada para qualquer situação onde pessoas necessitam trabalhar juntas buscando o mesmo objetivo com resultados eficazes.

O problema encontrado através de pesquisas foi que grande parte dos projetos sofrem alterações em cronogramas, orçamentos e na qualidade final, causados muitas vezes por um gerenciamento ruim da equipe ou pelo modo em que é empregado nas realizações das tarefas.

Esse trabalho busca responder a seguinte pergunta: O uso do *Scrum* pode evitar os problemas da equipe e a forma de desenvolvimento de um projeto? Assim tornando-o mais eficiente e de maior qualidade.

O objetivo geral dessa monografia é pesquisar os conceitos e valores da metodologia ágil *Scrum* buscando o bom gerenciamento de uma equipe e facilitar o desenvolvimento de software.

Os objetivos específicos é esclarecer que com o uso da metodologia ágil *Scrum* nos projetos de desenvolvimento de um sistema, consegue-se um bom gerenciamento da equipe e se obtém um aumento de qualidade e eficácia nos resultados finais, seja um projeto de pequeno, médio ou grande porte.

Para concretizar o objetivo a metodologia utilizada será um estudo bibliográfico sobre a metodologia e sua aplicação em um estudo de caso.

O desenvolvimento do trabalho foi estruturado em seis capítulos, sendo o capítulo 1 Introdução, no qual abordará uma breve explicação sobre o que será visto nos próximos capítulos. O capítulo 2 destina-se há apresentar uma breve introdução à Engenharia de Software e sobre as metodologias tradicionais e ágeis. No capítulo 3 o assunto abordado é a apresentação da metodologia Scrum, pilares da metodologia, formação da equipe e obrigações de cada membro e detalhamento o Time Scrum. O ciclo de vida do Scrum será abordado e explicado no capítulo 4, o qual também irá expor um estudo das práticas que compõe a metodologia e princípios básicos durante os processos de desenvolvimento do sistema. O capítulo 5 tem como objetivo a apresentação de um estudo de caso sem a utilização do Scrum e após a aplicação da metodologia para exemplificar seu uso no desenvolvimento de sistema. Para finalizar, através das informações obtidas nos estudos realizados nesses cinco capítulos anteriores, o capítulo seis se reserva às considerações finais.

2 ENGENHARIA DE SOFTWARE E METODOLOGIAS ÁGEIS

Engenharia de software é uma disciplina relacionada a todos os aspectos da produção de software, desde sua especificação até a manutenção. Com o uso de tecnologias e práticas de gerência de projeto como: escolha da equipe, planejamento, custos e prazos.

Já as tecnologias englobam linguagens de programação, base de dados, padrões e diversas ferramentas CASE[1], que podem auxiliar durante todo o período de desenvolvimento ou em apenas uma parte especifica, de acordo com a necessidade. Cada ferramenta oferece um serviço diferente, podendo ser para uma ou mais atividades. Alguns serviços oferecidos por essas ferramentas são: diagramas, gráficos, design, codificação e controle de cronograma, todos voltados para a obtenção de resultados satisfatórios junto à necessidade do cliente e manter os sistemas durante todo o seu ciclo de vida. (MODESTO, 2010).

Engenharia de Software também visa sempre à produtividade e melhor qualidade dos projetos, aumentando a produtividade no processo de desenvolvimento. Também foca na especificação, no desenvolvimento e na manutenção de sistemas de software já desenvolvidos, resultando juntamente ao desenvolvimento de uma documentação formal sobre o projeto.

O termo engenharia de software foi definido pelo cientista da computação Friedrich Ludwig Bauer como "o estabelecimento e uso de sólidos princípios de engenharia para obter software economicamente confiável e que trabalhe de forma eficiente em máquinas reais". (WIKIPEDIA, 2011). Entretanto o termo tornou-se conhecido após uma conferência em 1968, quando as dificuldades de projetar sistemas foram discutidas a fim de contornar a crise de software.

2.1 CRISE DE SOFTWARE

Crise do software foi um termo utilizado nos anos 70, quando a engenharia de software ainda não existia. Esse termo surgiu para expressar as dificuldades no

[1] *Computer Aided Software Engineering* (Engenharia de Software Auxiliada por Computador).

desenvolvimento frente a grande necessidade das empresas na aquisição de um software de qualidade e a falta de técnicas para o desenvolvimento.

Nessa época, mais precisamente em 1969, o desenvolvimento de software começou a utilizar as linguagens estruturadas e modulares no qual ficou claro que a indústria do software estava falhando, pois pesquisas apontaram que de 50 a 80% dos projetos não foram concluídos ou fracassados por não atingir os objetivos finais. Já os projetos que foram concluídos, atrasaram no prazo ou ultrapassaram o orçamento estipulado.

Esses problemas estavam diretamente ligados à dificuldade no entendimento dos requisitos e modo de trabalho das equipes, fatores que exigiram a criação de novos métodos e modificação de métodos existentes, foi então que surgiu a “Engenharia de Software” que adotou métodos e modelos de desenvolvimento de software e dividiu todas as tarefas em etapas para, no final, obter um sistema eficiente com resultados satisfatórios que solucionasse um determinado problema.

2.2 METODOLOGIAS TRADICIONAIS

Os métodos antigos, conhecidos como pesados ficaram caracterizados por sua rígida regulamentação com uma documentação bastante detalhada, no qual envolve seu código fonte, especificações por escrito em uma forma geral e técnica, manuais de uso do sistema, além de diagramas e até simulações para explicar funcionamento como um todo. Outra característica é a divisão das tarefas em etapas/fases, essas fases são bem definidas e quando terminadas geram um documento, protótipo ou um tipo de versão do software. O foco principal dessas metodologias é a previsibilidade dos requisitos, o que torna o projeto completamente planejado.

No entanto, esses métodos, mais especificamente o modelo cascata, tinham uma espécie de preocupação, pois todos os engenheiros teriam que trabalhar perfeitamente e executar suas tarefas com eficácia devido ao seu desenvolvimento fluir sempre para frente, tendo como padrão seguir as seguintes fases: análise de

requisitos, projeto, implementação, testes e manutenção do software. Como pode ser observado na Figura 1.

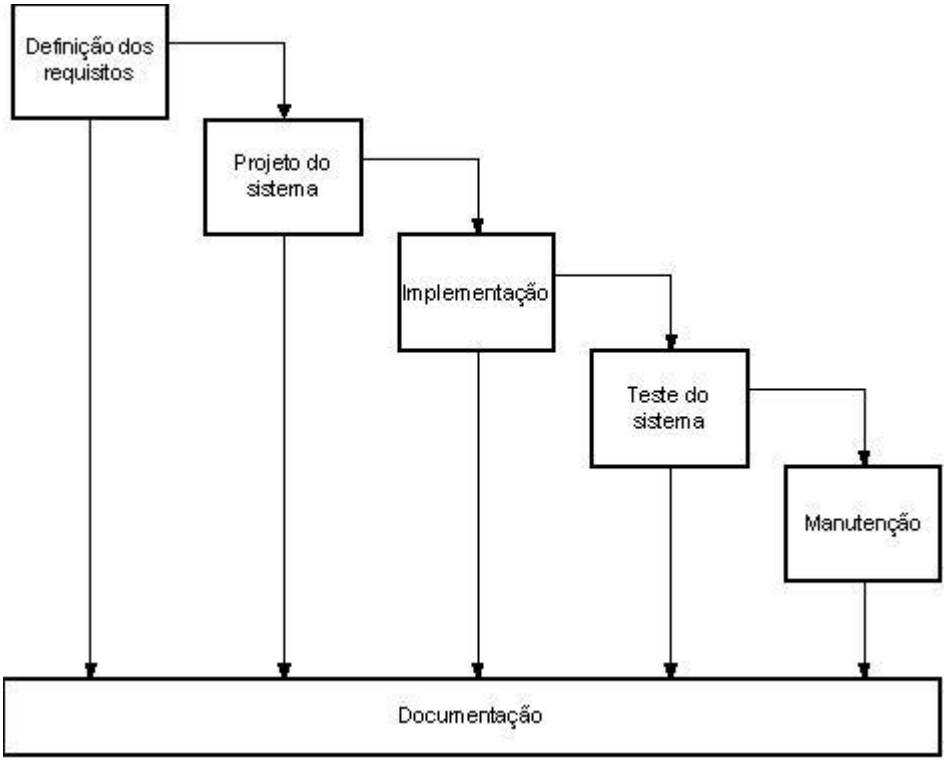

Figura 1 – Modelo Cascata (Fábrica de Software, 2010)

O progresso de uma fase para a outra era puramente de forma sequencial, sempre dependendo da fase anterior para se iniciar a próxima. Por exemplo, para dar início à implementação, as fases análise de requisitos e projeto deveriam estar concluídas. Outra dificuldade nesse modelo é o surgimento ou alteração de algum requisito no meio do desenvolvimento, fator que gera um grande impacto no projeto como um todo. Uma vez que a fase foi concluída não poderia mais sofrer modificações, caso a necessidade existisse, a fase atual teria que ser imediatamente interrompida e retomar o desenvolvimento desde a primeira fase, consequentemente, realizando as mudanças etapa por etapa, gerando um tempo e custo superior ao estimado no início do projeto.

Até meados da década de 1980 o modelo cascata era o único modelo com aceitação geral. Hoje é referência para outros modelos e serve de base para muitos projetos modernos.

> “Dados de 1995 usando como base 8.380 projetos, mostram que apenas 16,2% dos projetos foram entregues respeitando os prazos e os custos e com todas as funcionalidades especificadas. Aproximadamente 31% dos projetos foram cancelados antes de estarem completos e 52,7% foram entregues, porém com prazos maiores, custos maiores ou com menos funcionalidades do que especificado no início do projeto.” (SOARES, 2004, pág. 2)

Além disso, alguns projetos eram planejados e inteiramente desenvolvidos e só no final do desenvolvimento descobria que o mesmo não servia mais para aquele determinado propósito, pois as regras tinham sido modificadas e uma vez desenvolvido se tornaria muito difícil realizar qualquer alteração.

A partir desse cenário, ficou claro a necessidade de um método flexível, que possibilitasse o desenvolvimento de uma forma rápida e com maior adaptabilidade diante da situação e necessidade do cliente e dos desenvolvedores, pois o surgimento de novos requisitos e modificações é inevitável e acontecem a todo instante. Foi então que surgiram os métodos ágeis.

2.3 METODOLOGIAS ÁGEIS

Conforme dito por OLIVEIRA (2009), os métodos ágeis no cenário do desenvolvimento de software surgiram aproximadamente na metade de 1990 como forma reação contra os antigos métodos, conhecidos como pesados, assunto abordado no capítulo 2.2.

Esses métodos abdicam, em parte, à rígida regulamentação e trabalham com *feedbacks* constantes, o que facilita ajustar as eventuais mudanças, sendo então considerados flexíveis. Inicialmente foram conhecidos como *métodos leves* e apenas em 2001 nomeados para *métodos ágeis,* nome dado a partir de uma reunião de membros superiores e a publicação do *manifesto ágil*, documento que contém um conjunto de princípios e práticas para fundamentar o desenvolvimento ágil.

Segundo CRUZ, o manifesto contém quatro valores fundamentais:

- os indivíduos e suas interações acima de procedimentos e ferramentas;
- o funcionamento do software acima de documentação abrangente;
- a colaboração dos clientes acima da negociação de contratos;
- a capacidade de resposta às mudanças acima de um plano pré-estabelecido;

As metodologias ágeis tem uma forma de adaptação altamente valorizada, pois buscam adaptações muito rápidas devido às necessidades e mudanças que acontecem no decorrer do desenvolvimento, permitindo a integração do cliente para opinar nas etapas desenvolvidas, corrigir possíveis erros e solicitar modificações. Outra característica que o diferencia dos métodos antigos é o tempo, o mesmo é medido em curtos prazos sendo em semanas. (TI SIMPLES, 2013). Isso passa uma segurança maior para o cliente, pois os resultados aparecem semanalmente, fazendo com que facilite os ajustes e os feedbacks, trazendo consigo um resultado bem mais eficaz. Atualmente existem diversas metodologias ágeis, diferentes em suas práticas e ênfases, porém todas possuem uma mesma característica, como o desenvolvimento iterativo e incremental.

Um exemplo de como funciona o desenvolvimento utilizando-se uma metodologia ágil, pode ser visualizado na Figura 2.

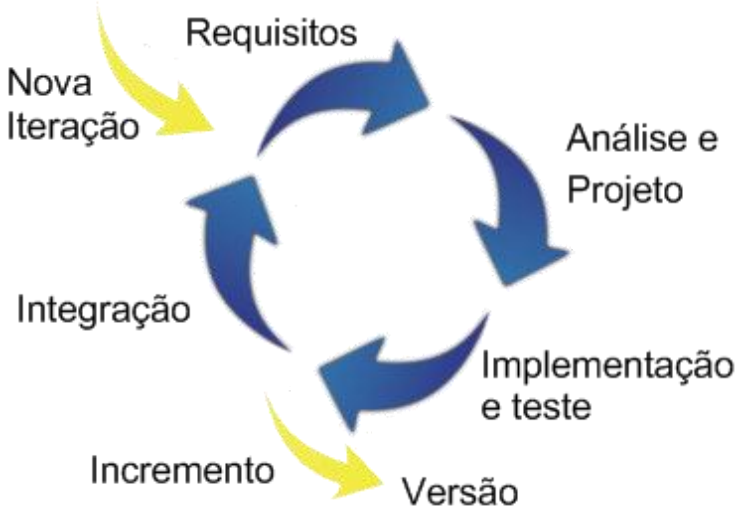

Figura 2 - Desenvolvimento Iterativo e Incremental (Fonte: Makino, 2009, pág. 18)

Conforme publicado em TI SIMPLES (2013), dentre as várias metodologias ágeis existentes, as mais conhecidas são a XP (*Extreme Programming*) e o *Scrum*. Ambas tem o foco de entregar o código funcionando em um curto prazo, devido a isso não se preocupam e gastam pouco tempo com a documentação, pois a maior parte do tempo é gasto em desenvolvimento. Ou seja, *Scrum* e XP focam em ter as coisas prontas e dão menos importância para documentações e diagramas.

3 *SCRUM*

> "*Scrum*, nome utilizado inicialmente pelos japoneses Hirotaka Takeuchi e Ikujiro Nonaka, que descrevia um tipo de processo de desenvolvimento de produto utilizado no Japão. Também o nome foi escolhido pela similaridade entre o jogo de Rugby, pois tanto o processo de desenvolvimento japonês e o jogo são adaptativos, rápidos e promovem à auto-organização." (CAMARA, 2008).

Scrum é uma metodologia ágil que possui o foco na equipe, a transição para essa metodologia exige um pouco de paciência, pois ela é totalmente diferente dos modelos tradicionais e até mesmo dos outros métodos ágeis, sendo pouco parecido com XP. (COHN, 2011, pág 21). Essa metodologia tem como objetivo estabelecer um processo de desenvolvimento iterativo e incremental, com pequenos ciclos de iteração, chamados de *Sprints*, que permite um maior controle dos riscos.

Os *Sprints* são realizados de acordo com cada atividade ou requisitos, que são definidos pelo cliente do projeto ou um gerente, que é denominado *Product Owner (PO).* Os requisitos são listados, de forma que os itens que tem o custo mais elevado no produto tenham maior prioridade no desenvolvimento. Essa lista de requisitos é chamada de *Product Backlog (PB).*

A equipe utiliza uma rotina, onde, no início de cada iteração, realiza-se uma análise do que devem fazer, após a analise identificam o que pode tornar um incremento de valor ao produto ao final da iteração. A equipe então distribui as tarefas entre os membros, essa etapa é definida como *Sprint Backlog.* Feito a divisão das tarefas, inicia-se o desenvolvimento ou seja, o *Sprint*, e no final os resultados são apresentados para os envolvidos no projeto*,* para que possam analisar e solicitar as mudanças no tempo apropriado. Para cada iteração a equipe também analisa as tecnologias necessárias e habilidades de cada membro, assim conseguem se adequar melhor nas divisões das tarefas e entregar com a melhor qualidade dentro do prazo, tudo de acordo com a maior prioridade.

Essas rotinas podem ser aplicadas no gerenciamento de qualquer atividade, não necessariamente no desenvolvimento de software.

3.1 PILARES DO *SCRUM*

Para o bom funcionamento do *Scrum*, suas práticas de controle são sustentadas por três Pilares, sendo eles: A transparência, inspeção e adaptação. Seguir esses pilares não é garantia de sucesso, mas as chances de conseguir aplicar *Scrum* em sua empresa ou projeto se tornam muito maiores.

3.1.1 TRANSPARÊNCIA – PRIMEIRO PILAR

Transparência abrange todos os acontecimentos que podem afetar no resultado final do projeto ou produto, esses acontecimentos devem estar claramente explícitos para quem gerencia os resultados, incluindo o cliente, tornando claro para todos o que foi já foi realizado, o que está sendo realizado e o que ainda será realizado.

Esse conceito é um dos mais difíceis de ser implantado, principalmente quando a equipe esta acostumada com os métodos tradicionais, pois nesse tipo de método, são eles quem definem quando mostrar, como mostrar, para quem mostrar e o que mostrar, uma vez que a equipe é foco das informações do projeto. (SCHWABER, 2009, pág. 3).

3.1.2 INSPEÇÃO – SEGUNDO PILAR

Na inspeção, todos os envolvidos diretos ou indiretos no projeto devem inspecionar se tudo o que foi desenvolvido e se o que esta sendo feito está de acordo com o planejado, identificando possíveis variações, eliminando riscos ou situações que comprometam a evolução do projeto. (SCHWABER, 2009 pág. 2 e 3).

A rápida identificação de qualquer desvio é fundamental para aplicação dos conceitos apresentados no terceiro pilar, denominado adaptação.

Essas inspeções devem ser frequentes, entretanto cuidadosas, pois podem acabar afetando nas realizações das atividades, gerando atraso no projeto.

3.1.3 ADAPTAÇÃO – TERCEIRO PILAR

A adaptação realiza as adaptações e modificações necessárias para que o projeto continue em bom andamento. Essas modificações são identificadas na inspeção e devem ser realizadas o mais breve possível para evitar desvios durante o desenvolvimento. Essas inspeções e adaptações acontecem frequentemente, a fim de assegurar que as metas do projeto sejam cumpridas. (SCHWABER, 2009 pág. 2 e 3).

3.2 PAPÉIS NO *SCRUM* (TIME *SCRUM*)

O Time *Scrum* é dividido em três papéis, sendo eles: *Scrum Master* (SM), *Product Owner* e Equipe *Scrum*. Esse time é composto por equipes pequenas, normalmente multidisciplinares, capazes de projetar, desenvolver e testar. Também são proativos e auto-organizáveis, pois escolhem a melhor forma de trabalhar e nunca são comandados por alguém de fora. Para LIMA (2011) esse modelo de equipe trabalha a fim de aperfeiçoar a flexibilidade, criatividade e produtividade no decorrer do desenvolvimento.

3.2.1 *SCRUM MASTER*

O *SM* é responsável por verificar se a equipe está adotando e aplicando os valores, práticas e regras da metodologia. Outro papel do SM é fazer a mediação entre os indivíduos de fora da equipe, para que elas entendam quais de suas interações com a equipe são validas e quais não são. Para PEREIRA (2011, pág. 17), ele é considerado um líder facilitador, pois sempre esta disponível para os outros interessados no projeto, visando remover obstáculos que venham comprometer as entregas. Entretanto ele não gerencia o Time *Scrum*, pois o mesmo é auto-organizável.

> "O *Scrum Master* pode ser um membro do Time; por exemplo, um desenvolvedor realizando tarefas do *Sprint*. No entanto, isso frequentemente leva a conflitos quando o *Scrum Master* deve escolher entre remover impedimentos e realizar tarefas. O *Scrum Master* nunca deve ser o *Product Owner*." (SCHWABER, 2009, pág. 7).

O *Scrum Master* possui algumas funções não só dentro da equipe, mas também para empresas e o dono do produto (PB). Alguns desses serviços são:

- **dono do produto/*Product Owner*** – Encontrar melhores técnicas para gerenciar o PO e comunicar claramente os seus objetivos para a equipe.
- **equipe de desenvolvimento** – Orientar sua equipe na auto-organização e ajuda-los a desenvolver um software de alta qualidade.
- **organização/empresa** – Ajudar a empresa na aquisição do *Scrum*, ajudar os envolvidos no projeto a entender e implanta-lo e trabalhar com outros *Scrum Master*s, aumentando a efetividade da sua aplicação.

3.2.2 *PRODUCT OWNER (PO)*

Para SCHWABER (2009, pág. 7) o *Product Owner* é apenas uma pessoa, e não uma equipe. Pode até existir uma equipe para aconselhar ou influencia-lo (a), no entanto é somente ele que define os itens do *Product Backlog* e suas prioridades. Essa pessoa pode ser um membro da equipe de desenvolvimento, porém caso seja atribuída essa função adicional, pode diminuir sua atenção nas atividades principais, podendo afetar o projeto como um todo. O *Product Owner* nunca deve ser o *Scrum Master*.

Para que ele tenha sucesso, todos devem respeitar sua decisão, ninguém tem a permissão de alterar as prioridades de um item, e o time não deve ouvir nenhuma outra pessoa que diga o contrário. Caso a equipe ou alguém queira modificar algum item ou prioridade, deve convencê-lo, para pessoalmente comunicar a equipe.

O PO é responsável por todo o gerenciamento do PB e tem o papel de representar todos que tenham interesse no projeto, mas não efetivamente deve ser o dono do projeto, pode ser um gerente da empresa ou um representante designado para o papel. Esse papel exige disponibilidade para as reuniões de planejamento e revisão, pois ele define as prioridades das tarefas, verifica depois de desenvolvida e decide se será aceita ou rejeitada. Para exercer esse papel não é necessário alto conhecimento sobre o *Scrum*, mas deve conhecer muito bem o trabalho que será desenvolvido.

4 CICLO DE VIDA *SCRUM*

Ciclo de vida no *Scrum* consiste na utilização de equipes pequenas, geralmente sendo no máximo sete pessoas. As iterações são curtas, os requisitos instáveis e o desenvolvimento são divididos no máximo em 30 dias.

No principio a equipe inicia uma visão simplificada do projeto, sendo que algumas ideias ainda não são claras nessa etapa, pois vão se esclarecendo conforme o decorrer do projeto. O Ciclo de vida no *Scrum* é dividido em algumas etapas.

A primeira etapa é o *Product Backlog*, ou seja, a lista de requisitos ou funcionalidades definidas pelo cliente no inicio do projeto. Com o PB definido e em mãos, a equipe inicia o *Sprint Backlog*, etapa em que equipe faz as divisões das tarefas entre seus membros e estima-se o tempo para a conclusão do *Sprint*.

Dividido as tarefas, enfim começa o desenvolvimento, chamado de *Sprint*. Essa fase pode demorar entre um até trinta dias, variando de acordo com a quantidade de atividades envolvida no *Sprint* atual e o grau de dificuldade para cada uma delas. Durante o desenvolvimento, a equipe realiza reuniões diárias, que servem para deixar todos atualizados do que foi desenvolvido naquele dia, expor problemas e ideias para o dia seguinte. (CAMARA, 2008).

Após a conclusão do *Sprint,* a equipe apresenta para o cliente o resultado das funcionalidades ou o resultado final do projeto, caso o mesmo seja o último *Sprint* desenvolvido. Nessa apresentação pode haver exigências por parte do PO, para alterações ou incluir novas funcionalidades. Caso exista a necessidade de alteração ou nova funcionalidade, o ciclo inicia novamente e só é finalizado com a entrega das solicitações concluídas. No entanto esse ciclo não faz mais parte daquele *Sprint* atual, ele é definido como novo, incluindo todas as características de um *Sprint* normal.

Na Figura 3, que segue abaixo, pode ser observado claramente o ciclo de vida do *Scrum*. Sendo o mesmo explicado detalhadamente nos próximos itens.

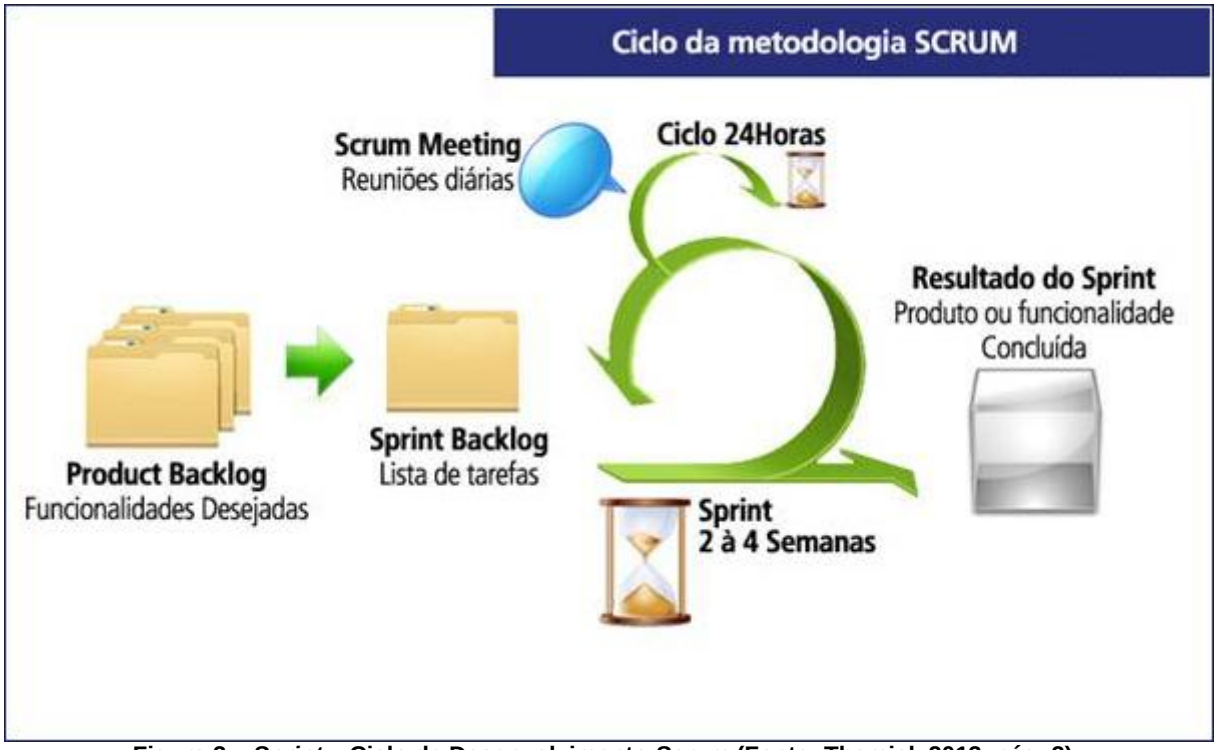

Figura 3 – *Sprint* – Ciclo de Desenvolvimento *Scrum* (Fonte: Thamiel, 2013, pág. 2)

4.1 *PRODUCT BACKLOG (PB)*

Todos os requisitos necessários para o desenvolvimento e lançamento de um projeto devem ser listados e descritos no *Product Backlog*. Geralmente são separados em itens e esses são numerados de acordo com a prioridade, sempre lembrando que os itens de maior prioridade são os que agregam mais valor financeiro dentro do projeto.

A classificação de prioridade é exibida através de um quadro ou uma tabela, sendo a tarefa de maior prioridade exposta no topo ou na primeira linha, no caso de ser uma tabela, seguida por todas outras tarefas, encerrando com a de menor valor.

Esse quadro deve ser fixado ao lado do *Product Backlog*, assim todos os envolvidos no projeto podem ficam ciente dos requisitos e das tarefas mais importantes, bem como as que estão em desenvolvimento, as que já foram desenvolvidas e as que ainda não iniciaram. A responsabilidade de criar e colocar o quadro no lugar correto é do gerente de projetos.

A Figura 4 mostra um exemplo de como as atividades são fixadas no quadro que será exposto para a equipe, esse quadro é semelhante ao utilizado pela metodologia *Kanban*.

Figura 4 - Modelo de um quadro para *Product Backlog* (Fonte: Desenvolvida pelo Autor)

Durante o planejamento é definido as estimativas para entrega dos itens e no decorrer da preparação esses itens são revisados, mas nunca são totalmente estáveis, podendo ser atualizados em qualquer momento. Os responsáveis de todas essas estimativas e mudanças é o time, porém o *Product Owner* pode influenciar ajudando a escolher as mudanças. Entretanto a estimativa final é feita pelo time e a responsabilidade de sua organização, disponibilidade e priorização são do *Product Owner*.

De acordo com SCHWABER (2009, pág. 16), o PB nunca está completo, pois o mesmo evolui de acordo com a evolução do ambiente em que se aplicará o produto, ele também é dinâmico, devido as constantes modificações que sofre para atender as necessidades a fim de ser apropriado para tal finalidade e útil quando completo.

4.2 *SPRINTS* – CICLO DE DESENVOLVIMENTO

No *Scrum*, o desenvolvimento é dividido em iterações, mais conhecido como *Sprints*. Esses *Sprints* normalmente são de uma semana e no máximo trinta dias, pois sendo assim, a equipe pode apresentar para o cliente uma evolução ou uma função finalizada, e curtos períodos de tempo. O tempo pode variar de acordo com o tamanho do projeto ou dificuldade nas atividades contidas na lista de requisitos, (PB). Os *Sprints* são definidos pela equipe, junto com o *Scrum Master* e o *Product Owner*, essa definição é feita na reunião de Planejamento do *Sprint*, a partir do *Product Backlog*, explicado no item 4.1. (SCHWABER, 2009, pág. 10 e 11).

Os *Sprints* ocorrem de forma sequencial, terminando uma já se inicia a outra, sem intervalos. Esses contêm e consiste nas reuniões de planejamento, reunião de revisão e retrospectiva do *Sprint*.

Durante o período de desenvolvimento do *Sprint* definido, caso a equipe identifique a falta de tempo, ou seja, que se comprometeu com algo que não conseguira cumprir no tempo previsto, encontram se então com o PO para remover alguns itens do *Product Backlog*. O mesmo encontro deve ser feito caso a equipe identifique que as atividades serão finalizadas antes do prazo previsto, no entanto essa reunião servirá para acrescentar novos itens. Pode ocorrer também o cancelamento do *Sprint*, isso acontece raramente, pela sua curta duração, só ocorre caso se torne obsoleto ou a empresa resolva mudar a direção devido às condições do mercado, e somente o *Product Owner* tem autoridade para ação. Se ocorrer o cancelamento, todos os itens feitos são revisados de forma que possa servir como incremento para outros *Sprints*, e os itens que não foram desenvolvidos, volta ao *Product Backlog*. (SCHWABER, 2009, pág. 11).

4.3 *SPRINT BACKLOG*

Os *Sprints Backlogs* são definidos como tudo o que é necessário para desenvolver e entregar uma parte do projeto, ou seja, é o objetivo de um *Sprint*. O surgimento dos *Sprints Backlogs* são a partir do que é levantado e listado pelo *Product Owner* na criação do *Product Backlog*. No entanto só será implementado no

momento em que estiver preparado, ou seja, a equipe de desenvolvimento e o PO devem estar de acordo em relação ao item que esteja sendo tratado no tal momento. O mesmo deve estar claramente explicito para a equipe, pois é ela quem determina se o item será incluso de desenvolvido naquele *Sprint*.

A escolha dos itens que serão incluídos no *Sprint Backlog* deve estar de acordo com o que o PO espera que o produto seja capaz de fazer quando chegar o fim daquele ciclo de desenvolvimento. (NASCIMENTO, 2012). Entretanto não é aconselhável que todos os itens sejam ligados diretamente a meta do *Sprint*, pois caso a equipe não conclua um deles, a meta não será comprometida.

Após a definição de quais itens será incluídos no *Sprint Backlog*, o ideal é que sejam divididos em tarefas, para receberem uma estimativa de conclusão, normalmente com um máximo de 8 horas, assim podendo ser finalizadas em um dia.

Um exemplo de *Sprint Backlog* pode ser claramente visualizado na Tabela 1.

Tabela 1 - Modelo de *Sprint* Backlog (Fonte: Desenvolvida pelo Autor)

Tarefas	Responsável	Estimativa(Horas)	Status
Criar serviço de email	Rafael	8	Concluída
Implementar anexos ao email	Patrícia	4	Andamento
Testar serviço de email e anexo	João, Priscilla	2	Não Iniciado

Durante um *Sprint*, o *Scrum Master* é responsável por manter o *Sprint Backlog* atualizando, para a equipe refletir quanto tempo ainda é necessário para finalizar o *Sprint* por completo. A estimativa do que resta a ser desenvolvido são calculados todos os dias e exibida em um gráfico, chamado de *Burndown Chart*.

4.3.1 *BURNDOWN CHART*

O gráfico de *Burndown Chart* é utilizado pelas equipes de *Scrum*, para demonstrar diariamente o que foi desenvolvido. Após o dia de trabalho o gráfico é

atualizado para representar a proporção do que já foi finalizado, comparado com o planejamento total da equipe.

O gráfico é representado por dois pontos, sendo:

- no eixo horizontal os dias do *Sprint*, do primeiro dia até o último.
- no eixo vertical os pontos que foram planejados inicialmente para a *Sprint*.

A Figura 5 apresenta uma modelo real de um gráfico de *Burndown*.

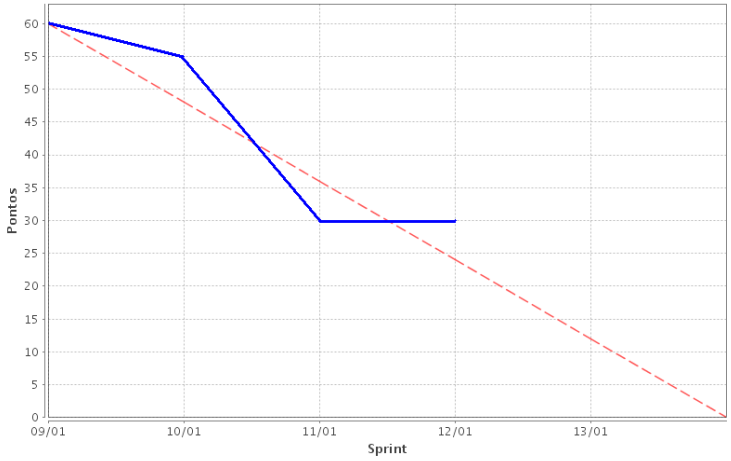

Figura 5 – Gráfico de Burndown Chart (Fonte: Adaptada de My*Scrum*half, 2012)

No entanto de acordo com LIMA (2012), muitas equipes não dão valor necessário nas informações do gráfico, simplesmente traçam a linha por traçar, mas não se atentam aos detalhes que o gráfico mostra. Quando observado detalhadamente, pode-se identificar o que está errado com o processo de trabalho adotado pela equipe e efetuar as mudanças necessárias para obter melhores resultados e evitar obstáculos futuros, podendo gerar retrabalho, no qual consequentemente gera perdas, seja de tempo ou financeiramente.

4.4 *SCRUM MEETING*

Scrum Meeting são as reuniões diárias, realizadas no mesmo local e no mesmo horário do expediente, porém antes da equipe iniciar o desenvolvimento. Essas reuniões têm durações de aproximadamente 15 minutos no máximo e não são necessárias à presença do *Product Owner*, pois serve apenas para aqueles que estão transformando os itens em incremento, ou seja, o Time.

O responsável por essa reunião é o *Scrum Master*, ele é encarregado de manter a reunião em curta duração, fazendo com que os membros sejam rápidos e objetivos nas suas falas.

Nas reuniões diárias servem como uma inspeção do que foi desenvolvido por cada membro desde a última reunião diária e o que será desenvolvido no dia atual, até a próxima reunião. Também são expostos dificuldades e obstáculos que tenham surgido no desenvolvimento ou algo que possa dificultar o próximo desenvolvimento.

A *Scrum Meeting* ajuda melhorar a comunicação entre os membros da equipe, fazendo com que identifique e removam futuros impedimentos. Também promovem a tomada rápida de decisões e fortalece o nível de conhecimento de todos. Entretanto essas reuniões são simples e bem objetivas, pois o planejamento detalhado das atividades é realizado na *Sprint Planning Meeting*.

4.5 *SPRINT PLANNING MEETING*

Sprint Planning Meeting é a reunião da equipe, *Scrum Master* e o cliente. Essa reunião acontece toda vez antes de iniciar um *Sprint* e geralmente é divida em duas etapas.

A primeira etapa é obrigatoriamente a presença do cliente, pois essa etapa consiste na revisão do *Product Backlog* por parte do mesmo. Ainda nessa etapa o PO deve confirmar o próximo *Sprint* a ser realizado, conforme descrito no PB. Caso houver alteração é nessa etapa que o cliente deve descrevê-las as apresentando o que ele quer que seja desenvolvido no próximo *Sprint*.

Já na segunda etapa não é necessário à presença do cliente, embora algumas empresas ou equipes que fazem questão de sua presença. Nessa etapa a equipe decide como o *Sprint* será desenvolvido, realizando as divisões de tarefas e estimando aproximadamente a quantidade de horas necessárias para o desenvolvimento completo.

> “Geralmente, somente 60-70% do total do Backlog da *Sprint* será definido na Reunião de Planejamento da *Sprint*. O restante é deixado para ser detalhado mais tarde ou são dadas estimativas grandes que serão decompostas mais tarde na *Sprint*.” (SCHWABER, 2009, pág. 13).

4.6 *SPRINT REVIEW* (REVISÃO DA *SPRINT*)

Sprint Review ocorre no final de cada *Sprint*, essa é uma reunião informal e sua duração é calculada de acordo com o tempo de duração daquele *Sprint*, normalmente 5% do total de horas exigidas para finaliza-lo. Essa reunião tem como finalidade a apresentação da funcionalidade desenvolvida e promover a colaboração para o que será feito em seguida.

Segundo CRUZ (2013), a reunião de Revisão do *Sprint* tem presença da equipe e do *Product Owner*, sendo o segundo, responsável por verificar o que foi e o que não foi feito, ele também faz algumas projeções de datas para os próximos *Sprints*, baseado na velocidade e qualidade que foi desenvolvido o último. Já a equipe discute informalmente o que correu bem e quais obstáculos foram identificados e como foram resolvidos.

Essa reunião apesar de ser informal, pode trazer informações importantes para a *Sprint Retrospective*, reunião na qual será discutido mais detalhadamente cada processo ocorrido durante o *Sprint*.

4.7 *SPRINT RETROSPECTIVE* (RETROSPECTIVA DO *SPRINT*)

A reunião de Retrospectiva do *Sprint* é realizada antes da próxima reunião de planejamento, tem duração fixa de três horas. Essa reunião deve conter a presença da Equipe de Desenvolvimento, do *Product Owner* e do *Scrum Master*, sendo o último, o responsável por fazer com que a equipe revise dentro do modelo de trabalho e das práticas do *Scrum*, os seus processos de trabalho, afim torná-los mais eficaz para o desenvolvimento do próximo *Sprint*. É importante nessa reunião que todos tenham a oportunidade de expor suas conclusões retiradas no *Sprint* e expor suas ideias para melhorar o próximo. (TOPICS, 2012, pág. 1).

Conforme descrito por SCHWABER (2009, pág. 15), a Retrospective tem como objetivo inspecionar o que ocorreu na última *Sprint*, se tratando de pessoas. Essa inspeção deve avaliar a relação entre as pessoas envolvidas e os processos, ferramentas utilizados durante o desenvolvimento, a fim de verificar e priorizar os itens que ocorreram bem e aqueles que poderiam ser feitos de outra forma para obter um melhor resultado. Esses itens englobam: preparativos para reuniões realizadas durante o *Sprint* anterior, formação da equipe nas divisões das tarefas, formas de comunicação entre todos os envolvidos no projeto e ferramentas utilizadas. No final da reunião o Time *Scrum* deve ter identificado as melhores medidas e adaptá-las durante o desenvolvimento da próxima *Sprint*, fazendo com que se obtenha um melhor resultado do que o anterior e assim sucessivamente para todos os *Sprints*.

4.8 FERRAMENTAS DE GERENCIAMENTO *SCRUM*

Para o gerenciamento de projetos utiliza-se de algumas ferramentas para conectar a equipe e os trabalhos realizados simultaneamente. Essas ferramentas ajudam no acompanhamento das tarefas, equipe e defeitos em todas as fases do ciclo de vida do desenvolvimento e manutenção de sistemas. Atualmente existem várias ferramentas no mercado, cada uma delas com suas características inerentes, porém todas com os mesmo objetivos, facilitar o gerenciamento do projeto e melhor expor as informações. Algumas das ferramentas são: PangoScrum, JIRA e FireScrum.

4.8.1 FIRESCRUM

O FireScrum[2] é uma ferramenta desenvolvida no modelo *open source*, sob a licença GPL (General PublicLicense), essa ferramenta que tem como o objetivo o gerenciamento de projetos que usam o *Scrum,* utiliza tecnologias que aumentam o nível de experiência dos usuários no uso das interfaces, otimiza os processos e possibilita o uso do *Scrum* por diversos times, além de gerar automaticamente gráficos, métricas e históricos. (CAVALCANTI, 2009, pág. 39)

[2] Link para ferramenta FireScrum: http://sourceforge.net/projects/firescrum/

4.8.1.1 CARACTERÍSTICAS

A ferramenta possui um esforço para garantir a usabilidade e simplicidade, facilitando o seu uso e uma maior aceitação. FireScrum é uma ferramenta Web, no qual pode ser acessível através do navegador, em um ambiente internet quanto intranet. No quesito usabilidade, propõem interfaces completas e ao mesmo tempo simples, prática e focada em um objetivo. Também é extensível, possui uma arquitetura modular que possibilita a criação de novos módulos para expandir as funcionalidades. (CAVALCANTI, 2009, pág. 41).

4.8.1.2 FUNCIONALIDADES

As funcionalidades do FireScrum são divididas em módulos, o que facilita ainda mais o seu uso, podendo utilizar somente as funcionalidades necessárias. Os módulos que a ferramenta oferece, pode-se incluir:

- *CORE* – Responsável por todo o controle operacional do *Scrum*, possui as seguintes funcionalidades: controle de acessos, cadastros de usuários e mais de um projeto, criação e priorização de itens *Backlogs*, criação de *Sprints,* alocação de usuários em uma tarefa e criação dos gráficos de *Burndown.* Esse módulo é o único obrigatório na utilização da ferramenta. Na Figura 6, exibe uma tela de como o usuário cadastra um novo projeto. (CAVALCANTI, 2009, pág. 42).

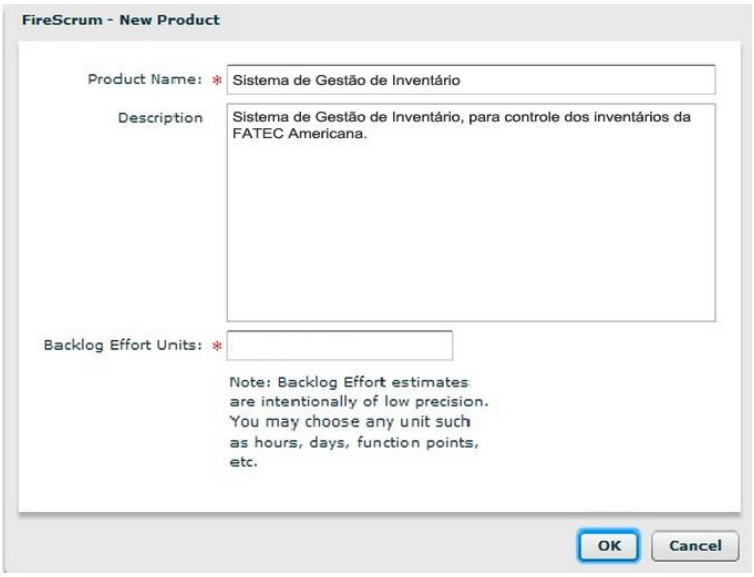

Figura 6 – Tela de Cadastramento de Sistemas (Fonte: Ferramenta FireScrum)

> “Uma vez que o módulo Core é considerado o módulo principal do sistema e responsável por toda a parte operacional básica do *Scrum*, o mesmo deve ser auto-suficiente para contemplar todo o ciclo de vida básico de um projeto *Scrum*, sendo possível utilizá-lo de forma independente em projetos *Scrum*. Os demais módulos são considerados módulos de apoio, contemplando necessidades e ferramentas que possam tornar o Fire*Scrum* uma ferramenta ainda mais atrativa, excluindo a necessidade de ferramentas de terceiros que apoiem o desenvolvimento de software como um todo.” (CAVALCANTI, 2009, pág. 47).

- *TASKBOARD* – Objetivo desse módulo é a criação do quadro físico utilizado por usuários do *Scrum*, o qual é fixado na parede para a avaliação do andamento do projeto, como explicado no item 4.1. A Figura 7 representa uma tela desse módulo. (CAVALCANTI, 2009, pág. 43).

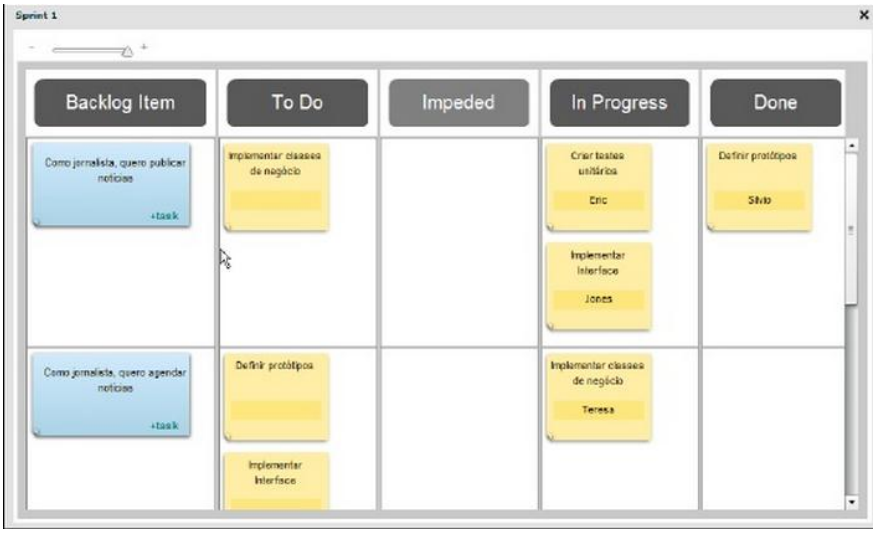

Figura 7 – Exemplo do Quadro *TASKBOARD* (Fonte: CAVALCANTI, 2009, pág. 43)

- *PLANNING POKER* – Esse módulo tem como funcionalidade a visualização dos itens do *Backlog* e suas estimativas. Possibilita também um *chat* entre os membros do projeto com áudio e vídeo para a melhor comunicação e gerenciamento das estimativas. (CAVALCANTI, 2009, pág. 43).

- *TEST MANAGEMENT* – Esse módulo basicamente possui a função de criar o plano de testes, evitando o uso de ferramentas externas para tal atividade. (CAVALCANTI, 2009, pág. 45).

- *DESKTOP AGENT* – Esse módulo é apenas uma instalação offline no sistema operacional, no qual é alimentada quando a máquina esta conectada na internet e quando desconectada, pode-se visualizar todas as informações contidas no FireScrum. (CAVALCANTI, 2009, pág. 46).

- *BUG TRACKING* – O objetivo desse módulo é controlar os *bugs*, permite o registro de todos os *bugs* durante o ciclo de vida do projeto, define o responsável pelo *bug* e associa o mesmo a um item do *Backlog*, como pode ser visualizado na Figura 8. (CAVALCANTI, 2009, pág. 45).

Figura 8 – Painel de Registros de Bugs (Fonte: CAVALCANTI, 2009, pág. 45)

5 ESTUDO DE CASO: UTILIZAÇÃO DO *SCRUM* NO DESENVOLVIMENTO DE SOFWARE

Nesse capitulo será apresentado um estudo de caso realizado dentro da FATEC Americana, tendo em vista a demonstração de um desenvolvimento de software simples, porém sem a utilização de qualquer metodologia ágil, mais especificamente sem nenhuma utilização dos conceitos e valores do *Scrum,* assim tornando os processos de desenvolvimento confuso e com inúmeros problemas.

5.1 ESTUDO DE CASO – O SISTEMA

O estudo de caso a ser apresentado, envolve o desenvolvimento de um sistema para controle de gestão de inventário. O projeto de desenvolvimento surgir a partir da necessidade de se construir um software, para finalizar a matéria Laboratório de Engenharia de Software, ministrada pelo professor Me. Anderson Luiz Barbosa, no curso de Análise e Desenvolvimento de Sistemas, período vespertino na FATEC de Americana.

Dado a proposta, os alunos teriam que se reunir em dupla e encontrar um problema real ou fictício que tivesse a necessidade da construção de um software, como forma de corrigir ou minimizar os problemas de um determinado local ou empresa. No entanto o software não poderia ser grande, devido ao curto tempo de aproximadamente quatro meses para a apresentação final do projeto. Foi então que surgiu a proposta de criar um software que facilitasse a gestão do inventário da própria faculdade, uma vez que foi apontado o problema de não existir controle.

Tendo um curto tempo para planejamento, o foco principal do sistema era controlar todos os equipamentos de informática contidos no campus da faculdade e talvez em longo prazo a ideia fosse realizar melhorias no sistema, incluindo novas funcionalidades para que o mesmo tivesse a capacidade de controlar não apenas os equipamentos de informática, mas sim todo o inventário. Com o problema identificado, o tempo praticamente definido e com a autorização do professor para esse sistema, foi dado inicio ao desenvolvimento.

5.2 O DESENVOLVIMENTO

Para o tal projeto a dupla optou por utilizar o método tradicional de desenvolvimento, tendo como base o modelo cascata, item abordado no inicio desse trabalho. O método escolhido é dividido em etapas, o qual foi seguido pela dupla.

5.3 LEVANTAMENTO DE REQUISITOS

A primeira etapa era o levantamento de requisitos, o que normalmente é feito através de reuniões com o cliente, porém a equipe enfrentava problemas para se reunir com representantes de TI da faculdade, que pudesse ajudar a levantar os requisitos necessários. A partir dessa dificuldade foi então que um dos membros da dupla e ex-funcionário da faculdade levantou superficialmente como era a administração até tal momento e o que era necessário em um software, para facilitar o trabalho dos funcionários em administrar os inventários de informática.

5.4 PROJETO DO SISTEMA

Levantado as informações à equipe elaborou um escopo dos requisitos funcionais e não funcionais, incluindo diagramas, tabelas e interfaces do sistema.

Na primeira reunião com o cliente, sendo um representante da faculdade, a dupla apresentou o que tinha elaborado até o momento. Algumas modificações foram sugeridas pelo cliente e outras funcionalidades deveriam ser acrescentadas, mas no geral a forma que a equipe tinha iniciado os seus trabalhos estava no rumo correto.

Com o escopo realizado pela dupla, junto com as modificações solicitadas a equipe finalizou dentro de um mês, todo o levantamento de requisitos, criação diagramas e modelagem do banco de dados.

5.5 CODIFICAÇÃO

Após a finalização das primeiras etapas de planejamento, iniciou-se a codificação. Essa etapa foi dividida entre os dois desenvolvedores, um ficou responsável por desenvolver as interfaces e as funções para validações e mascara de campos, a responsabilidade o outro integrante foi de realizar a codificação de cadastros e consultas. Semanalmente os dois membros discutiam sobre como estava o andamento um do outro, sugerindo alguma modificação e alguma ajuda caso fosse necessário. Essas atividades de tiveram duração de aproximadamente um mês e meio. Terminado as tarefas separadas, a dupla se encontrou durante duas semanas para enfim unir codificações com interfaces e funções Java script, chegando ao fim de toda codificação.

5.6 TESTES

Depois de codificado o software, iniciou-se os processos de testes, tendo como primeira etapa um plano de testes, seguido por outro teste, realizado pelo professor, orientador da matéria. Encontrado alguns erros e algumas sugestões de melhoria, a dupla teve uma semana para realizar as correções e modificações. No entanto algumas modificações e sugestões não puderam ser realizadas, devido ao desenvolvimento seguir os conceitos das metodologias tradicionais, alguns requisitos geram muita dificuldade e outros não se consegue mais realizar mudanças, pois esse modelo não se preocupa com a adaptação do sistema e os requisitos devem ser fixos do inicio ao fim.

5.6.1 PLANO DE TESTES

O Plano de testes foi desenvolvido pelos próprios desenvolvedores do sistema, o plano abordava testes como cadastramentos, consultas, navegabilidade e interação do usuário, totalizando em 40 testes. Os testes foram executados por pessoas que não fizeram parte do sistema, e houve marcações como: Numero do caso, objetivo do teste, pré-requisitos para funcionar, dados para o teste, etapas dos testes, resultados, data da realização, Status e descrição da falha, caso ocorresse.

Na Tabela 2 podem ser observados dois exemplos de como foi realizado o plano de testes. O primeiro teste o sistema não apresentou nenhum erro, o resultado foi o esperado, o segundo teste o sistema apresenta uma falha, como pode ser visto no campo “Descrição da Falha”.

Tabela 2 – Caso de Testes Usado no Sistema (Parcial)
(Fonte: Plano de Testes - Sistema Gestão de Inventário, 2013)

Caso de Teste	Objetivo	Pré-requisitos	Dados de Teste	Etapas do teste	Resultados esperados	Data Realização	Status	Descrição da Falha
CT1	Verificar se o sistema cumpre as obrigatoridades para o cadastramento de componente.	Deixar campos em brancos ou informações em formatos inválidos.	bp = "617084" Local_idLocal = "2" Tipo_idTipo = "1" numeroSerie = " " descricaoComponente = "CPU Intel Core2Duo, 4 GB, 500GB"	Entrar com dados do componente.	Sistema exibe uma mensagem de alerta, com o aviso de campos inválidos ou em branco.	21-May	OK	Sem Falhas
CT2	Busca por componente.	Entrar com o bp que não tem cadastrado.	bp = "698854"	Entrar com bp do componente.	Sistema exibe uma mensagem de alerta, com o aviso de componente não cadastrado.	5/21/2013	X	Não exibe alerta, apenas lista vázia.

5.6.2 TESTE COM PROFESSOR

Após ser realizado o plano de teste, os próximos testes, foi executado pelo professor orientador do projeto. Esses testes foram mais rígidos, sendo testadas as codificações, validações de campo e armazenamento de informações no banco de dados, ou seja, todos os pontos que poderiam afetar diretamente o objetivo principal do sistema. Nessa etapa também houve vários pontos apontados como críticos e várias solicitações de melhorias, as quais seriam realizadas juntamente com as mudanças propostas no plano de testes.

5.6.3 MODIFICAÇÕES E MELHORIAS

Os pontos identificados, quais necessitavam de alterações e melhorias, foram discutidos entre a dupla e alguns ficaram claros a impossibilidade de alteração. Entretanto várias modificações foram realizadas, as tarefas de correções e melhorias foram dividas entre a dupla, no qual um ficou responsável por todas as melhorias e o outro para executar as devidas correções.

Com essa fase terminada, enfim a equipe apresentou o projeto para o cliente, pois o mesmo deverias realizar os testes adequados, como se estivesse realmente utilizando o software, chamado de teste *Beta*.

A Figura 9 representa umas das alterações realizadas, podendo ser observado uma melhoria na interface do software, organizando e facilitando a navegação.

Figura 9 – Modificações na Interface do Software
(Fonte: Sistema de Gestão de Inventário, 2013)

5.6.4 TESTE COM O CLIENTE

Passado por todas as etapas de testes com o professor e correções, a equipe iniciou junto com o cliente os testes finais. A equipe se reuniu com usuários, representantes da faculdade para realizar a apresentação do projeto considerado final, sendo apresentado no ambiente de utilização.

Nessa apresentação ficou claro que o projeto não estava apto para ser implantado. Os usuários identificaram problemas com as interfaces, sendo o mesmo não podendo ser corrigido, pois as interfaces já estavam conectadas com as funções do sistema, identificaram também problemas como falta de relatórios e consultas, funcionalidades que interferiam a utilização do sistema. A funcionalidade principal do sistema foi atendida, no entanto alguns pontos se tornaram críticos, devido à falta de adaptabilidade.

Então ficou decidido que o software deveria passar por uma reformulação no desenvolvimento, sendo um maior planejamento inicial, um desenvolvimento adaptativo e com maior participação do cliente, para assim os resultados finais serem positivos, o software atender completamente as necessidades e a satisfação do cliente.

5.7 PROBLEMAS IDENTIFICADOS

Durante todo o processo de desenvolvimento do software descrito no estudo de caso acima, foram encontrados diversos problemas. Alguns desses problemas foram de facilmente resolvido, outros mesmo que com alguma dificuldade também foram solucionados, porém alguns problemas identificados no decorrer dos processos, não obtiveram sucesso nas correções e outros ainda mais graves, não tinham como serem modificados e acabaram afetando diretamente na aceitação do cliente e no objetivo principal do produto, consequentemente prejudicando a entrega final.

5.8 APLICAÇÃO DO *SCRUM* NO ESTUDO DE CASO

Esse item propõe a aplicação dos conceitos do *Scrum*, para mostrar em quais pontos são importantes durante o desenvolvimento do software, mais especificamente no estudo de caso acima, mostrando que se o mesmo fosse desenvolvido aplicando o *Scrum*, o projeto não teria fracassado.

Os problemas que foram identificados e explicados no estudo de caso acima são problemas que podem ocorrer quando se utiliza os métodos tradicionais de desenvolvimento. O software de Gestão de Inventário não apresentou um bom planejamento e não teve boa comunicação entre os desenvolvedores e o cliente, fatores que foram fatais para o fracasso na aceitação e implantação do projeto.

5.8.1 ESTUDO DE CASO COM *SCRUM*

Como uma forma de comparação entre os métodos utilizados no estudo de caso e os conceitos que a metodologia estudada até o momento propõe, abaixo será explicado como seria aplicado os valores do *Scrum*, para o mesmo desenvolvimento do Sistema de Gestão de Inventário. Para maior entendimento da aplicação dos valores, será mostrado por etapas.

5.8.1.1 ETAPA 1 - EQUIPE

A primeira etapa a ser modificada seria a formação da equipe. O estudo de caso apresentava a equipe com dois desenvolvedores, um professor como orientador do projeto e representantes do cliente. A equipe não possuía um líder, o representante do cliente não era definido, uma vez que poderia ser qualquer funcionário.

No *Scrum* a equipe deve ser formada por uma equipe de desenvolvimento, podendo ser com dois membros, porém deve conter um membro chamado *Scrum Master*, para ser o líder da equipe e também denominar uma única pessoa para ser o *Product Owner*, no qual será o cliente ou um representante, assim formando o Time *Scrum.*

5.8.1.2 ETAPA 2 – PILARES DO *SCRUM*

A segunda modificação com os conceitos do *Scrum* é a utilização dos três pilares proposto para o bom funcionamento da metodologia, Transparência, Adaptação, Inspeção. Esses pilares definem que todos os requisitos do sistema devem estar claros para todos os envolvidos, os mesmos devem ser inspecionados por todos a fim de encontrar modificações e alterações necessárias a tempo de ser resolvido antes de causar transtornos.

No Sistema de Gestão de Inventário, apenas os dois desenvolvedores conheciam os requisitos, realizavam inspeções e adaptações, porém os requisitos não estavam claros e as inspeções e adaptações foram feitas sem a participação do cliente, o que não se deve fazer, pois o cliente que irá utilizar o sistema, assim deve ter participação durante todo o desenvolvimento e saber o que deve ser modificado para atender suas necessidades.

5.8.1.3 ETAPA 3 – REUNIÕES

No estudo de caso ficou explicito a falta de reuniões, tanto de planejamentos, como, acompanhamento, finalização de etapas e revisões. O *Scrum* aborda os conceitos de quatro tipos de reuniões, sendo duas sem a necessidade do cliente (*Scrum Meeting* e *Sprint Review*), e as outras duas com sua presença obrigatória (*Sprint Planning* e *Sprint Retrospective*). No estudo nenhuma dessas reuniões foram feitas, pois a metodologia adotada não aborda esses conceitos. Com a utilização do *Scrum*, reuniões diárias deveriam ser feitas, assim cada membro da equipe poderia saber como está o desenvolvimento dos outros integrantes, podendo auxiliar em caso de dúvidas ou qualquer impedimento. Reuniões de revisões com o cliente são fundamentais, pois cada etapa desenvolvida o cliente interage com o sistema, verificando possíveis modificações a tempo de serem realizadas. No desenvolvimento citado a equipe não se utilizou desse recurso, pois todo o planejamento foi feito no inicio do desenvolvimento, o qual se aplicaria as reuniões de planejamento, para saber o que o cliente necessita para as próximas etapas, tarefas que são definidas no inicio do projeto, como nos métodos tradicionais, porém são revisadas antes de iniciar as próximas etapas.

Essas reuniões contidas no *Scrum* faria com que o desenvolvimento do sistema de Inventário, tomasse outro rumo, pois com uma participação mais ativa do cliente, as tarefas e resultados seriam mais claros, verificando cada etapa do desenvolvimento, a fim de manter o projeto com as características necessárias para atingir seu objetivo.

5.8.1.4 ETAPA 4 – UTILIZAÇÃO DO FIRESRUM

Alguns módulos da ferramenta FireScrum poderia ser utilizado no desenvolvimento do software de Gestão de Inventário, como forma de apoio para os desenvolvedores. Além do módulo CORE, o qual é obrigatório na utilização da ferramenta, a dupla poderia utilizar *Planning Poker e Bug* Tracking, sendo o primeiro para facilitar o controle das atividades desenvolvidas por cada membro e aproximar a comunicação entre os desenvolvedores e cliente, fazendo com que todos tivessem conhecimento do que esta sendo desenvolvido. Já com o módulo *Bug*

Tracking os membros teriam um maior controle sobre possíveis *bugs* no projeto, a fim de concerta-los antes de afetar etapas importantes do desenvolvimento.

Apesar de ser um desenvolvimento de software de pequeno porte, a utilização da ferramenta seria viável, pois assim a equipe manteria um controle maior das suas atividades, gerando mais confiança para o desenvolvimento de cada atividade e nas demonstrações dos resultados para o cliente.

5.9 CONSIDERAÇÕES FINAIS DO ESTUDO DE CASO

Mesmo com um bom planejamento, um desenvolvimento correto seguindo todas as etapas e uma boa relação da equipe, os projetos que ainda utilizam desses métodos tendem a não atender todas as necessidades do cliente. Esses problemas ocorrem devidos os requisitos atualmente serem muito instáveis. Um software planejado e iniciado o desenvolvimento hoje, para estar em uso em seis meses, provavelmente quando pronto talvez já não seja suficiente para atender a necessidade do cliente, pois as necessidades normalmente aumentam ou se modificam durante o tempo.

O estudo de caso apresentado deixou claras as dificuldades de se trabalhar com os métodos tradicionais. Os desenvolvimentos do software de gestão de inventário foram encontrados vários problemas, que afetaram diretamente o resultado final, problemas que poderiam ser mais graves e gerar um impacto muito maior, quando se falamos de softwares de grande porte.

Os problemas identificados como mais graves aconteceram, principalmente, pela falta de comunicação, falta de planejamento e falta de adaptação para mudanças futuras. O projeto não foi planejado como deveria, uma vez que o cliente teve pouca participação no decorrer do desenvolvimento e não teve como expressar sua opinião e acompanhar o que estava sendo feito.

Todos esses problemas poderiam ser minimizados ou até evitados com a aplicação dos conceitos da metodologia *Scrum* junto com a Ferramenta FireScrum. Metodologias ágeis vieram para suprir a dificuldade de tornar o desenvolvimento mais prático, com um maior controle de riscos e maior adaptação diante das

mudanças necessárias no decorrer do desenvolvimento. Conforme visto no tópico 5.5 a utilização *Scrum* consideravelmente evitaria os transtornos enfrentados pela equipe e cliente, fazendo com que o software atendesse melhor as necessidades da faculdade e o mesmo tivesse sido implantado e em funcionamento.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um breve estudo sobre as metodologias ágeis, com um maior detalhamento sobre o *Scrum,* uma das metodologias mais utilizadas atualmente para gerenciamento e desenvolvimento de projetos em geral, porém mais focada em desenvolvimento de software. O trabalho pode ser utilizado para auxiliar aos gerentes de projeto ou até mesmo as organizações entenderem um pouco melhor sobre essa metodologia e verificar a possibilidade de aplica-la em projetos ou dentro de empresas, seja ela de pequeno, médio ou grande porte.

A grande competitividade na área de desenvolvimento faz com que as organizações busquem sempre melhorar os seus serviços para se destacar dos concorrentes. Alguns fatores que beneficiam nesse diferencial são prazo de entrega e qualidade, além de uma adaptação a mudanças constantes. A utilização do *Scrum* pode facilitar as empresas a atingir esses objetivos mais facilmente, basta ser bem aplicado seja qualquer o ambiente.

A metodologia não é a solução para todos os problemas, porém mostra uma maneira de trabalhar bastante organizada e iterativa, contribuindo também com um ambiente de trabalho mais amigável, facilitando a comunicação, resultando em um resultado final muito mais satisfatório.

Primeiramente este trabalho apresentou alguns conceitos básicos da engenharia de software, que apesar de antigos ainda são utilizados com muita frequência, apresentou também conceitos das metodologias tradicionais, as quais foram base para a criação de muitas metodologias ágeis e ate hoje empregam um pouco dos seus valores.

Em seguida concentrou-se em estudar fundamentos teóricos, valores e práticas sobre o *Scrum,* principalmente no que diz respeito a desenvolvimento de software, detalhando cada processo da metodologia, equipe, conceitos e aplicabilidade.

Posteriormente foi visto um estudo de caso, no qual houve um desenvolvimento de software sem a utilização de metodologia ágil, usando apenas valores básicos da engenharia de software e seguindo as etapas que fazem parte do estilo tradicional de desenvolvimento.

Esse mesmo estudo de caso foi utilizado para demonstrar a diferença do resultado final, caso o mesmo fosse desenvolvido aplicando de toda teoria estudada nesse trabalho.

Após a realização deste trabalho, conclui-se que é perfeitamente viável a utilização do *Scrum*, no desenvolvimento de software. Neste contexto o trabalho comprovou que inúmeros benefícios podem ser adquiridos com o seu uso, podendo-se destacar: um maior controle da equipe de desenvolvimento e o envolvimento entre todos os envolvidos no projeto. Entretanto é necessário saber como aplicar os conceitos vistos nesse trabalho, para cada projeto em particular.

Para finalizar, fica a sugestão de trabalhos futuros, o desenvolvimento de uma aplicação fundamentada na metodologia estudada, para auxiliar no desenvolvimento de projetos no setor de Tecnologia da Informação e o desenvolvimento do estudo de caso aplicando os conceitos da metodologia, a fim de mostrar na prática que o uso do *Scrum* minimiza os problemas com as equipes e faz com que o desenvolvimento seja mais eficaz, tornando o software mais eficiente e de melhor qualidade, para atender melhor as necessidades dos clientes.

REFERÊNCIAS

CAMARA, Fabio. **Metodologias e Processos:** Uma metodologia ágil - *SCRUM*. Artigo Disponível em: <http://www.linhadecodigo.com.br/artigo/2084/uma-metodologia-agil-*Scrum*.aspx>. Acesso em: 05 jun. 2013.

CARLOS, Antônio et al. **Engenharia de Software:** Modelo Cascata. Apresentação Univesidade do Vale do Sapucai. Disponível em: <http://modelocascata.blogspot.com.br/2010/05/modelo-cascata-apresentacao.html>. Acesso em: 06 maio 2013.

CARLOS, João. UML: **Ferramentas CASE**. Publicado em: 08/03/2005. Disponível em: <http://i*Master*s.com.br/artigo/3048/uml/ferramentas-case/>. Acesso em: 20 jul. 2013.

CAVALCANTI , Eric de Oliveira. **Fire*Scrum*: Ferramenta de Apoio à Gestão Ágil de Projetos Utilizando *Scrum*.** Dissertação de Mestrado. Publicado em: 27/09/2009. Disponível em: <http://pt.scribd.com/doc/36064003/FIRE*SCRUM*-FERRAMENTA-DE-APOIO-A-GESTAO-DE-PROJETOS-UTILIZANDO-*SCRUM*/>. Acesso em: 04 nov. 2013.

COHN, Mike. **Desenvolvimento de Software com *Scrum* – Aplicando métodos Ágeis com Sucesso**. Tradução: Aldir José Coelho Corrêa da Silva. Revisão Técnica: Rafael Prinkladnicki - Dados Eletrônicos - Porto Alegre: Bookman, 2011.

CRUZ, Fábio. ***Scrum*:** *Sprint Review*. Disponível em: <http://www.fabiocruz.com/outros/*Sprint*-review/>. Acesso em: 02 out. 2013.

CRUZ, Fábio. **Manifesto Ágil.** Disponível em: <http://www.fabiocruz.com/outros/manifesto-agil/>. Acesso em: 30/05/2013.

FÁBRICA de Software – **Definir processo de desenvolvimento**. Disponível em: <http://voat.com.br/rdal/?tag=cascata>. Acesso em: 02 jun. 2013.

GRÁFICO de **Burndown Chart** Disponível em: Publicado em: 20/09/2013. <http://blog.my*Scrum*half.com/wp-content/uploads/2012/01/Burndown_31.png>. Acesso em: 30 set. 2013.

LIMA, Ester. **Interseções dos Papéis no *Scrum*.** Publicado em: 23/08/2011. Disponível em: <http://blog.my*Scrum*half.com/2011/08/intersecoes-dos-papeis-no-*Scrum*/>. Acesso em: 24 ago. 2013.

LIMA, Ester. ***Scrum* Half:** Burndown chart. Publicado em: 20/09/2013.Disponível em: <http://blog.my*Scrum*half.com/2012/01/burndown-chart-medindo-o-progresso-de-sua-*Sprint*-e-trazendo-indicativos-do-processo-de-trabalho-da-equipe/>. Acesso em: 20 set. 2013.

MAKINO, Amanda. **Abordagem da Metodologia RUP no Desenvolvimento de um Sistema de Gestão Comercial.** Monografia Faculdade de tecnologia de Taquaritinga. Disponível em: <http://www.ebah.com.br/content/ABAAAAsdkAG/monografia>. Acesso em: 01 jun. 2013.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Fundamentos de Metodologia Científica.** São Paulo: Altas S.A., 2010.

MODESTO, Jéssica; OLIVEIRA, Caique. **Noções de Engenharia de Software.** Publicado em: 11/03/2010. Disponível em: <http://nocoesengsw.blogspot.com.br/2010/03/ferramentas-case.html>. Acesso em: 25 abr. 2013.

NASCIMENTO, Paula. **O que é *Sprint* Backlog ?.** Publicado em: 13/02/2012. Disponível em: <http://blog.my*Scrum*half.com/2012/02/o-que-e-o-*Sprint*-backlog-%E2%80%93-faq-*Scrum*/>. Acesso em: 30 set. 2013.

OLIVEIRA, Thiago L. **Desenvolvimento Ágil de Software: Uma Abordagem com Scrum e XP**. Monografia Faculdade de Tecnologia de Mococa.Publicado em 2009. Acesso em 30/04/2013. Disponivel em:<http://www.slideshare.net/kakocaju/desenvolvimento-gil-de-software-uma-abordagem-com-scrum-e-xp>

PEREIRA, Alessandro F.. ***Scrum Master*:** Agilidade é adaptar-se rapidamente. Publicado em: 09/09/2011. Disponível em: <http://www.cova.com.br/afp/node/17>. Acesso em: 10 set. 2013.

RODRIGUES, Rafael; ROST, Rafael. ***SCRUM*: (Metodologia para o Desenvolvimento Ágil de Software)**. Monografia UNICAMP. Disponível em: <http://rafaelrgi.files.wordpress.com/2007/11/*Scrum*.pdf>. Acesso em: 02 jun. 2013.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do *Scrum***: Um guia definitivo para o *Scrum*: As regras do jogo. Publicado: 05/2009. Disponível em: <http://www.*Scrum*.org/Portals/0/Documents/*Scrum*%20Guides/*Scrum*%20Guide%20-%20Portuguese%20BR.pdf>. Acesso em: 25 abr. 2013.

SOARES, Michel Dos Santos. **Metodologias Ágeis Extreme Programming e *Scrum* para o Desenvolvimento de Software.** Artigo Universidade Presidente Antônio Carlos. Publicado em: 2004. Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/view/146/38>. Acesso em: 01 jun. 2013.

SOARES, Michel Dos Santos. **Metodologias Ágeis Extreme Programming e *Scrum* para o Desenvolvimento de Software.** Artigo Universidade Presidente Antônio Carlos. Publicado em: 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>. Acesso em: 01 jun. 2013.

SOMMERVILLE, Ian. **Engenharia de Software**. 8ª ed. Tradução: Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa. Revisão técnica: Keichi Kirama. São Paulo: Pearson Addison Wesley, 2007.

THAMIEL, Thiago. **Entendendo o *SCRUM*.** Publicado em: 17/07/2013. Disponível em: <http://thiagothamiel.com/category/desenvolvimento-agil/page/2/>. Acesso em: 09 set. 2013.

TI SIMPLES. **Metodologias ágeis X Metodologias tradicionais.** Publicado em: 18/04/2013. Disponível em: <http://tisimples.wordpress.com/2009/04/18/metodologias-ageis-x-metodologias-tradicionais/>. Acesso em: 20 jul. 2013.

TOPICS in *Scrum*: ***Sprint*** **Retrospective**. Publicado em 2012. Disponível em: <http://www.mountaingoatsoftware.com/*Scrum*/*Sprint*-retrospective/>. Acesso em: 25 set. 2013.

WIKIPEDIA. **Scrum**. Publicado em 2011. Acesso em: 15/03/2013. Disponível em:<http://pt.wikipedia.org/wiki/Scrum>