PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO
COGEAE - PUCSP
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

VINICIUS ANIBAL AMARAL VIEIRA

BENEFÍCIOS DA ADOÇÃO DA METODOLOGIA ÁGIL *SCRUM*: ESTUDO DE CASO EM UMA EMPRESA DE SEGURANÇA DIGITAL

São Paulo/SP
2013

VINICIUS ANIBAL AMARAL VIEIRA

BENEFÍCIOS DA ADOÇÃO DA METODOLOGIA ÁGIL *SCRUM*: ESTUDO DE CASO EM UMA EMPRESA DE SEGURANÇA DIGITAL

Monografia apresentada ao curso de Especialização em Engenharia de *Software*, da Pontifícia Universidade Católica de São Paulo - COGEAE, como parte dos requisitos para obtenção do título de Especialista em Engenharia de Software.

Orientador: Maurício Nacib Pontuschka

São Paulo/SP
2013

Dedico esse trabalho a todos aqueles que me deram suporte e força para chegar até aqui.

AGRADECIMENTOS

Agradeço a Deus por colocar luz no meu caminho diariamente e sabedoria para ter chegado com força até aqui.

À minha mãe Izildinha pelos ensinamentos da vida, força, amor e luta para conquistar todos meus sonhos.

À minha namorada Larissa pelo amor incondicional, por compartilhar todos os momentos de minha vida, pela paciência e por estar sempre ao meu lado.

Ao Prof. Maurício Nacib por todo o conhecimento que obtive, apoio e incentivo.

“Embora ninguém possa voltar atrás e fazer um novo começo, qualquer um pode começar agora e fazer um novo fim.”

(Chico Xavier)

RESUMO

Este trabalho apresenta um estudo aprofundado sobre a metodologia *Scrum*, aplicada em uma equipe de desenvolvimento que desenvolve produtos e soluções de segurança digital, como cartões bancários, *tokens*, etc. A metodologia ágil estudada, *Scrum*, é caracterizada pelo desenvolvimento de diversas iterações. Cada iteração resulta em um *software*, que é incrementado na iteração subsequente. Com o intuito de demonstrar os benefícios da metodologia, é apresentado o estudo de caso aplicado.

Palavras-chave: *Scrum*. Desenvolvimento ágil de produtos. Gestão de projetos de *software*. Metodologia ágil.

ABSTRACT

This work presents a detailed study of the Scrum methodology, applied to a development team that develops products and digital security solutions, such as bank cards, tokens, etc. The agile methodology studied, Scrum, is characterized by the development of several iterations. Each iteration results in a software that is incremented in subsequent iteration. In order to demonstrate the benefits of the methodology, the case study presented is applied.

Keywords: Scrum. Agile product development. Project management software. Agile methodology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E SIGLAS

CMMI	Capability Maturity Model
FDD	Feature Driven Development
OTD	On Time Delivery
ROI	Return On Investment
RUP	Rational Unified Process
UML	Unified Modeling Language
XP	Extreme Programming

SUMÁRIO

1. INTRODUÇÃO

A indústria de desenvolvimento de *software* tem se tornado uma das mais importantes indústrias do nosso tempo. Empregando milhares de trabalhadores em todos os países do mundo, essa indústria cria alguns dos produtos mais essenciais que nós utilizamos para manter o nosso estilo de vida (LEFFINGWELL; MUIRHEAD, 2004).

O *software* tem se tornado um aliado direto ao controle de produção, da qualidade dos eletrônicos, veículos, automação dos negócios e de muitas outras áreas de nossas vidas.

A habilidade de entregar produtos de *software* que atendem as reais necessidades dos clientes, com qualidade, dentro dos custos e prazo é o diferencial de uma empresa na atual intensa competitividade no ambiente de desenvolvimento de *softwares*.

Em meados da década de 1990, surgiram técnicas de desenvolvimento ágil de produtos de *software*. A utilização dos métodos ágeis é uma maneira para acelerar os prazos de desenvolvimento de projetos e alcançar a satisfação dos clientes agregando maior valor ao produto final.

Neste contexto, destaca-se o *Scrum*, uma abordagem enxuta de gerenciamento de projetos de produtos.

Este trabalho tem por objetivo apresentar os conceitos fundamentais da metodologia ágil de desenvolvimento, aprofundando-se nos fundamentos da metodologia *Scrum* e apresentar o estudo de caso aplicado usando o *Scrum* em uma equipe de desenvolvimento sem metodologia definida.

2. ENGENHARIA DE *SOFTWARE*

Desenvolver um *software* com qualidade não é uma tarefa trivial. No início, o *software* era desenvolvido sem a utilização de métodos sistemáticos e sem nenhuma documentação. Esses *softwares* dependiam dos profissionais envolvidos em seu desenvolvimento para que tivessem sucesso, uma vez que eram os únicos que conheciam seu funcionamento.

Com o passar dos anos e o aumento do uso dos *softwares* em empresas para benefício próprio, a complexidade dos *softwares* cresceu exponencialmente e dentro do novo cenário ficou evidente a necessidade de utilizar metodologias e processos que guiassem o desenvolvimento de *software*, o que levou o surgimento da Engenharia de *Software*.

O uso da Engenharia de *Software* tem sido constante, tanto para o meio acadêmico, onde ela é considerada uma disciplina fundamental, quanto para o meio profissional de desenvolvimento de *software*, que busca sempre a melhor maneira para evitar riscos e garantir a qualidade e satisfação no produto final (SILVA, 2010).

Para Pressman (2003), a Engenharia de *Software* abrange um conjunto de três elementos fundamentais tanto para *hardware* quanto para *software*, são eles: Métodos, Ferramentas e Procedimentos.

- Métodos: Proporcionam os detalhes de "como fazer" para construir um *software*, ou seja, são os procedimentos para produzir o trabalho.
- Ferramentas: Proporcionam aos métodos um apoio automatizado ou semi-automatizado para realizar tarefas da melhor forma possível, de maneira precisa, eficiente e produtiva.
- Procedimentos: Proporcionam o resultado específico através do elo que existe entre os métodos e as ferramentas. Os procedimentos definem a sequência em que os métodos serão aplicados; os produtos que serão entregues assim como seus documentos, relatórios, formulários; os controles que ajudam a assegurar a qualidade e coordenar as mudanças necessárias e os marcos de referência que possibilitam aos gerentes de *software* avaliar o progresso dos projetos.

Este capítulo aborda os conceitos relacionados ao desenvolvimento de *software*, os processos e metodologias de *software*, a fim de apresentar o desenvolvimento ágil com o objetivo de alcançar um aprimoramento contínuo na criação de *softwares*.

2.1 Processo de *Software*

O "Processo de *Software* é um conjunto de atividades e resultados associados que produzem um produto de *software*." Dentro do âmbito de desenvolvimento este processo orienta os passos para a execução de determinada atividade com certo grau de maturidade (SOMMERVILLE, 2003).

> Os processos de desenvolvimento surgiram como parte da engenharia de *software*. Seu principal objetivo é assegurar a qualidade no desenvolvimento de sistemas em todas as suas etapas, para isso eles servem de guia para a execução de atividades, definem quais produtos serão desenvolvidos e quando, coordenam as mudanças necessárias, e auxiliam o acompanhamento do progresso do desenvolvimento do *software* (DAMKE e MORAES, 2008, p.29).

A principal preocupação nesta atividade é atingir com qualidade e dentro dos prazos todos os requisitos definidos pelo cliente, de forma que o *software* possa ser facilmente adaptável e customizado.

O uso de um processo de *software* inadequado pode reduzir a qualidade ou a utilidade do produto de *software* a ser desenvolvido e/ou aumentar os custos de desenvolvimento (SOMMERVILE, 2003). Este fato leva as organizações que produzem *software* a usar processos de desenvolvimento que sejam eficientes e que atendam plenamente suas necessidades.

Pressman (2003) defende que um processo de *software* pode ser entendido como um *framework* [1] para as tarefas que são necessárias para a construção de *software* de alta qualidade. Ainda segundo Pressman o processo é uma das camadas que formam a engenharia de *software* e é tido como apoio. É o

[1] *FRAMEWORK*: um *framework* é uma estrutura de suporte definida em que outro projeto de *software* pode ser organizado e desenvolvido. Um *framework* pode incluir programas de suporte, bibliotecas de código, linguagens de script e outros *softwares* para auxiliar no desenvolvimento e unir diferentes componentes de um projeto de *software*.

responsável pelo atingimento da qualidade e a criação eficiente e oportuna de *softwares*.

> Os processos de *software* formam a base para o controle gerencial de projetos de *software* e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelo, documentos, dados, relatórios, formulários etc.) são produzidos, os marcos estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas. (PRESSMAN, 2003, p.17).

Assim, a Engenharia de *Software* fornece ferramentas e métodos para auxiliar o desenvolvimento do *software*.

Segundo Sommerville (2003), para sistemas críticos é necessário um processo de desenvolvimento muito bem estruturado. Nos sistemas de negócio, com requisitos que mudam rapidamente, um processo flexível e ágil é provavelmente mais eficaz.

Existem vários processos de desenvolvimento de *software*, porém algumas atividades fundamentais são comuns a todos eles (SOMMERVILE, 2003):

- Especificação: define a funcionalidade do *software* e as restrições sobre sua operação.
- Projeto e implementação: o *software* que atenda a especificação deve ser produzido.
- Validação de *software*: o *software* deve ser validado para garantir que ela faça o que o cliente deseja.
- Evolução: o *software* deve evoluir para atender aos novos requisitos que naturalmente surgirão.

Processos de *software* têm como base modelos de processo genéricos. Esses modelos genéricos não são descrições definitivas de processos de *software*. Segundo Sommerville (2003), são abstrações do processo que podem ser usadas para explicar diferentes abordagens para o desenvolvimento de *software*. Eles podem ser considerados como *frameworks* de processo que podem ser ampliados e adaptados para criar processos mais específicos de engenharia de *software*.

2.2 Metodologias Tradicionais de Desenvolvimento

As metodologias tradicionais possuem processos bem definidos, onde todo o escopo das entregas e contexto do projeto é definido no começo do projeto. Dessa forma, toda a documentação precisa ser definida e detalhada, motivo que torna essa abordagem pesada (SILVA, 2010).

O uso desta metodologia é mais adequado em situações onde em geral os requisitos do projeto não sofrem tantas alterações e os mesmos são bem conhecidos.

Segundo Silva (2010), as metodologias tradicionais podem deixar os desenvolvedores amarrados a requisitos desatualizados que muitas vezes não condizem com a expectativa do cliente. No mercado atual de *software*, a competitividade é um fator relevante, sendo assim, a busca por agilidade, qualidade, prazo de entrega e custos são aspectos que determinam a permanência das empresas no mercado. Dessa maneira, cada vez mais as empresas de *software* buscam por metodologias inovadoras, que possam suportar as novas mudanças e necessidades dos clientes. O resultado final de todos esses fatores citados resulta na satisfação do cliente.

2.2.1 Modelo Cascata

O Modelo Cascata, também conhecido como ciclo de vida clássico, descreve um método desenvolvimento sequencial, ou seja, uma etapa deve ser completada antes que a próxima etapa seja iniciada.

Os nomes dados a cada etapa variam, mas basicamente o ciclo de vida que compõe o modelo é mostrado na Figura 1: engenharia de sistemas, análise, projeto, codificação, teste e manutenção.

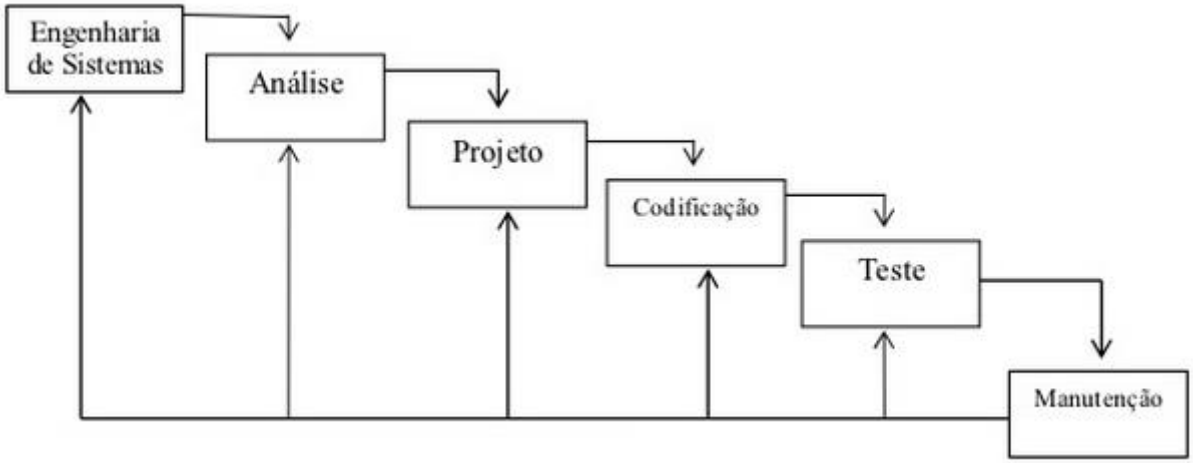

Figura 1 – Modelo Cascata
Fonte: Livro Engenharia de Software. PRESSMAN, 2003

Esse modelo introduziu importantes qualidades ao desenvolvimento:

- O processo de desenvolvimento deve ser conduzido de forma disciplinada e estruturado;
- As atividades devem ser claramente definidas;
- Permite o controle departamental e gerencial; e;
- Separa as atividades de definição e *design* da atividade de programação.

Os pontos negativos do uso deste modelo são:

- É um modelo linear e rígido;
- Não permite a participação do cliente durante o desenvolvimento;
- Não existe como o cliente verificar o andamento do projeto para identificar eventuais problemas;
- Não suporta modificação nos requisitos;
- Não suporta manutenção;
- Não permite a reutilização; e;
- Se ocorrer atraso em uma das etapas, todas as etapas seguintes são afetadas.

Complementando os pontos negativos, Pressman (2003) aponta alguns problemas identificados neste ciclo de vida:

- Os projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Alguma iteração sempre ocorre e traz problemas na aplicação do paradigma;
- Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente. O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projetos;
- O cliente deve ter paciência. Uma versão do *software* não estará disponível durante o projeto. Um erro crítico, se não for detectado até que o *software* seja revisto, pode ser desastroso.

2.2.2 Prototipação

O modelo de prototipação permite que o usuário tenha melhor compreensão do *software* e possibilita melhorias durante o desenvolvimento. Conforme o crescimento do *software*, o usuário tem uma visão sobre os aspectos visuais e os resultados das iterações e assim pode colaborar com o desenvolvimento. Dessa forma, o usuário torna-se parte da equipe, como um colaborador e também é responsável pelo sucesso do *software*.

Segundo Pressman (2003) este modelo pode trazer os seguintes benefícios:

- O modelo é interessante para alguns sistemas de grande porte nos quais representem certo grau de dificuldade para exprimir rigorosamente os requisitos;
- É possível obter uma versão do que será o sistema com um pequeno investimento inicial;
- A experiência de produzir o protótipo pode reduzir o custo das fases posteriores; e;
- A construção do protótipo pode demonstrar a viabilidade do sistema.

Questões a serem consideradas quanto à utilização do modelo:

- A Prototipação deve ser utilizada apenas quando os usuários podem participar ativamente no projeto;
- Durante todo o processo, conduzir uma boa análise;

- Esclarecer aos usuários que o desempenho apresentado pelo protótipo não necessariamente será o mesmo do sistema final;
- Evitar que o sistema final seja um protótipo em que foram implementados todos os requisitos especificados, pois corre-se o risco de ter-se um sistema mal implementado, uma vez que as técnicas utilizadas para desenvolver um protótipo são diferentes daquelas utilizadas na implementação de um sistema; e;
- Durante a etapa de prototipação, documentar todos os pontos levantados e implementados no protótipo, que não constavam dos requisitos iniciais, para incluí-los na documentação final.

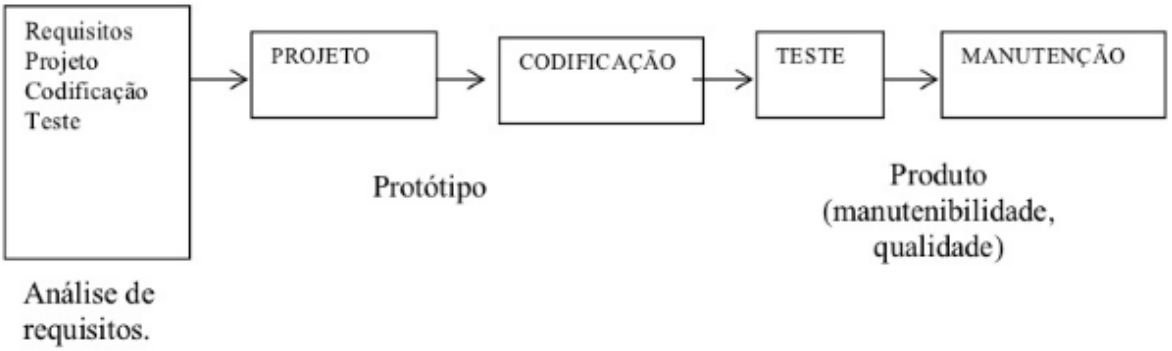

Figura 2 - Modelo de Prototipação
Fonte: Livro Engenharia de Software. PRESSMAN, 2003.

Como pode-se observar na Figura 2, Pressman (2003) coloca que a prototipação tem como opção fazer parte do Modelo em Cascata, como forma de auxiliar a ligação entre cliente e equipe de desenvolvimento em definições de requisitos.

2.2.3 Modelo Espiral

O modelo espiral foi proposto como forma de integrar as melhores características tanto do ciclo de vida clássico como da prototipação, acrescentando a análise de risco, que falta a esses paradigmas.

Os trabalhos principais partem do centro e percorrem para o sentido de fora da espiral conforme o projeto tem andamento.

Pressman (2003) divide o modelo espiral em quatro partes:

- Planejamento: é a busca das definições de riscos, objetivos e restrições;
- Análise de Riscos: busca-se como resolver os problemas e os melhores caminhos para alcançar os objetivos do *software*;
- Engenharia é a parte de desenvolvimento do produto; e;
- Avaliação pelo usuário: é o resultado para o cliente, a avaliação do trabalho.

LESSA e JUNIOR (2012), dizem que esse modelo usa uma abordagem "evolucionária" à engenharia de *software*, capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada fase evolutiva. O modelo espiral usa a prototipação como um mecanismo de redução de riscos. Ele mantém a abordagem de fases sistemáticas sugeridas pelo ciclo de vida clássico, incorporando esse paradigma numa estrutura iterativa, que reflete mais realisticamente o mundo real.

> O modelo espiral usa a prototipação como um mecanismo de redução de riscos. Ele mantém a abordagem de fases sistemáticas sugeridas pelo ciclo de vida clássico, incorporando esse paradigma numa estrutura iterativa, que reflete mais realisticamente o mundo real (PRESSMAN, 2003).

Vantagens:

- Permite uma visão mais completa do sistema ao longo de cada iteração, recorrendo à prototipação para reduzir os riscos;
- Permite usar a abordagem do ciclo de vida clássico de uma forma mais realística ao mundo real, a partir das iterações.

Desvantagens:

- Alto custo;
- Dificuldade em convencer os clientes que a abordagem evolutiva é controlável;
- Exige experiência na avaliação dos riscos. Um risco crítico não descoberto poderá gerar problemas ao projeto;

- Pode levar o desenvolvimento paralelo dos módulos do sistema, sendo cada um abordado de um modo diferenciado e exigindo um cronograma e técnicas específicas.

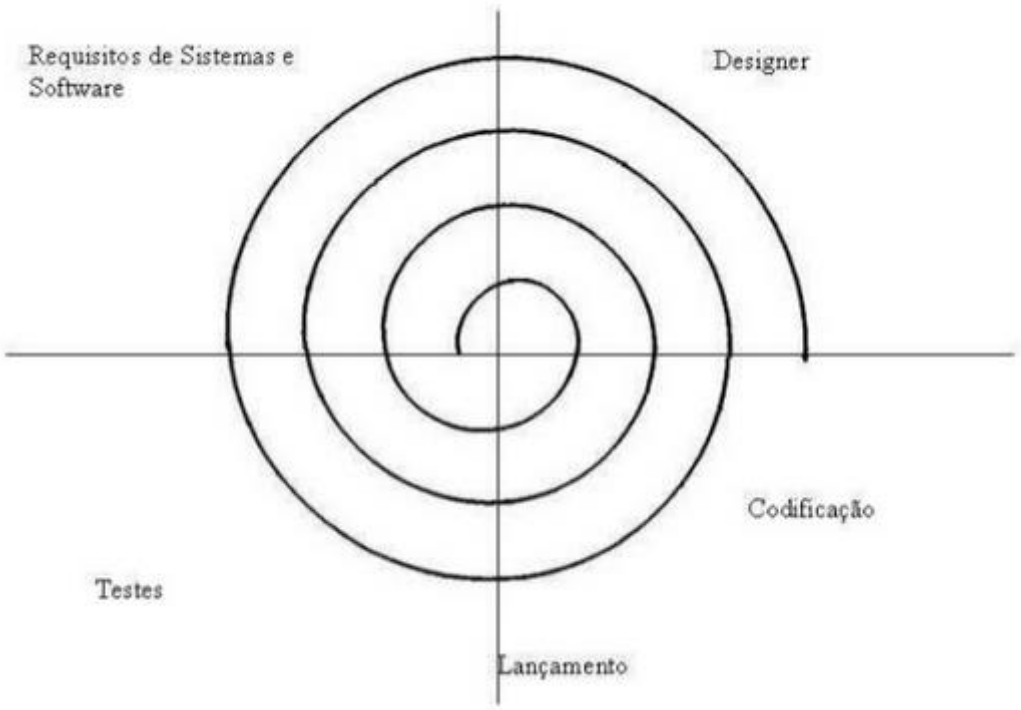

Figura 3 – Modelo Espiral
Fonte: Livro Engenharia de Software. PRESSMAN, 2003

2.3 Metodologia Ágil de Desenvolvimento

Os Métodos Ágeis surgiram como uma alternativa aos Métodos Tradicionais de desenvolvimento.

De acordo com Beck (2013), o manifesto ágil foi escrito em fevereiro de 2001, em uma cúpula de 17 profissionais de mentalidade independente de várias metodologias de programação. O grupo chegou ao consenso de que alguns princípios eram determinantes para a obtenção de bons resultados. O resultado deste encontro foi a identificação de 12 princípios e a publicação do Manifesto Ágil (BECK et al., 2013), que os representa com quatro premissas:

- Indivíduos e iterações são mais importantes do que processos e ferramentas;
- *Software* funcionando é mais importante do que documentação completa;

- Colaboração com o cliente é mais importante do que negociação de contratos;
- Adaptação a mudanças é mais importante do que seguir o plano inicial;

Kalermo e Rissanen (2002) apontam os 12 princípios mencionados acima:

1. Prioridade é satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado;
2. As mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
3. Frequentes entregas do *software* funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
4. As pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
5. Os projetos devem ser construídos em torno de indivíduos motivados. Dando o ambiente e o suporte necessário e confiança para fazer o trabalho;
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
7. *Software* funcionando é a medida primária de progresso;
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
9. Contínua atenção a excelência técnica e bom design aumentam a agilidade;
10. Simplicidade para maximizar, a quantidade de trabalho não realizado é essencial;
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis;
12. Em intervalos regulares, a equipe deve refletir sobre como tornar- se mais efetiva, e então, ajustar-se de acordo com seu comportamento.

Segundo Silva (2010), essas mudanças de paradigmas sobre o processo de desenvolvimento e gerenciamento de *software* são o que denotam as principais

diferenças entre metodologias tradicionais e ágeis. As esquipes devem absorver essa nova abordagem e pensar no processo de desenvolvimento de forma diferente da tradicional. Esse paradigma permite o cliente solicitar e priorizar o que deve ser feito pela equipe de *software*, que estará focada nas solicitações, e assim, juntamente ao cliente, validar as contínuas entregas do sistema.

Uma das características das metodologias ágeis, é que elas são adaptativas ao invés de serem preditivas como nos processos tradicionais. Sendo assim, são mais adequadas para situações em que a mudança de requisitos é frequente.

Os processos tradicionais se caracterizam por focar a análise de sistemas, investindo esforços para a obtenção de artefatos de projeto tão completos e precisos quanto possível: relatórios, documentos e diagramas. Grande parte dos processos tradicionais em uso atualmente tem como base os modelos de diagramas da *Unified Modeling Language* (UML), como é o caso do *Rational Unified Process* (RUP). A metodologia ágil é muito adequada para situações em que a mudança de requisitos é frequente, ou seja, para ser ágil, a metodologia deve aceitar a mudança em vez de tentar prever o futuro (PAULA, 2012).

A Figura 4 representa o custo de mudança no *software* entre as metodologias tradicionais e ágeis. A linha pontilha representa essa relação para uma metodologia tradicional, enquanto a linha contínua mostra como se espera que a relação melhore nas metodologias ágeis (SILVA, 2010).

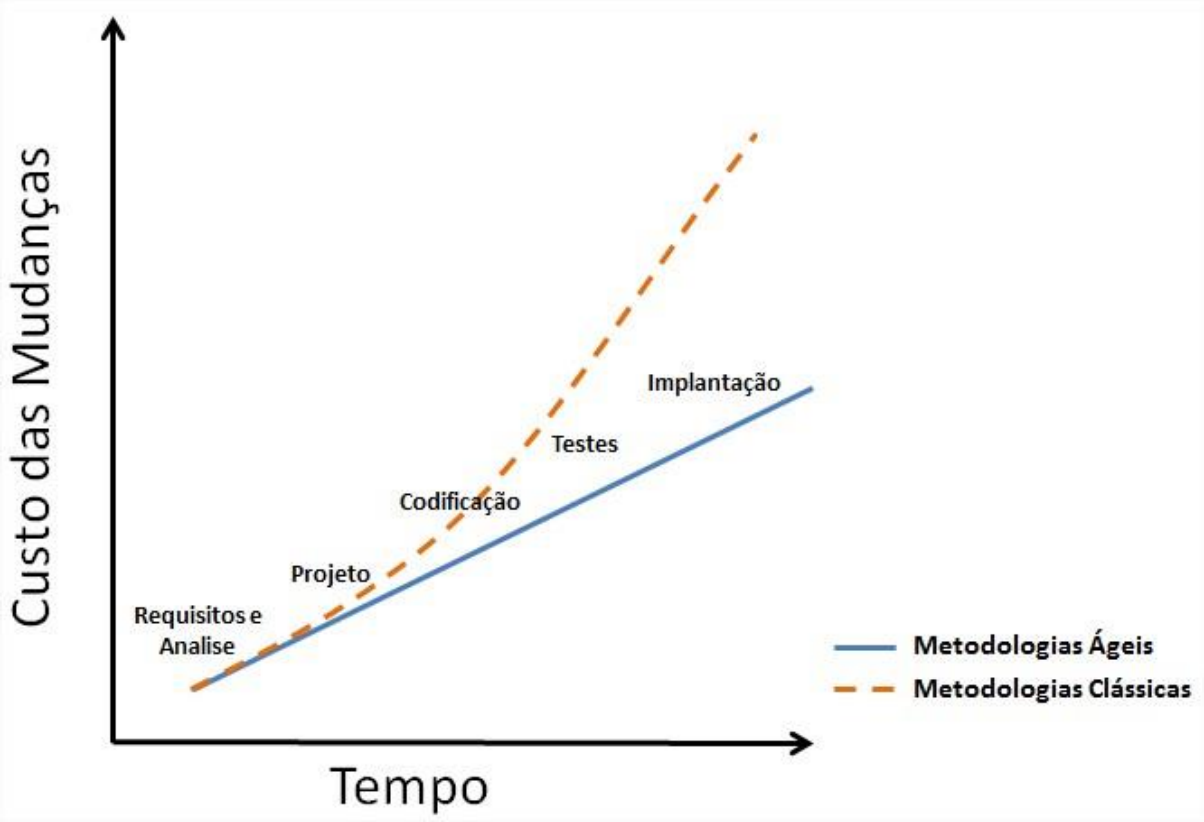

Figura 4 - Comparação do custo de mudanças entre metodologias tradicionais e ágeis.
Fonte: Silva (2010).

Qualquer processo de *software* pode aplicar a abordagem ágil, desde que tenha um bom planejamento, onde a equipe de projeto possa se adaptar as tarefas e as constantes mudanças de requisitos e aperfeiçoá-las, eliminando impedimentos e mantendo os produtos de trabalhos essenciais o mais simples possível. Desta forma, é preciso enfatizar a estratégia incremental onde o cliente possa dar o *feedback* do produto através da visualização, ou do contato do *software* funcionando o quanto antes, se possível, em tempos curtos, isso faz com que o processo torne-se ágil permitindo adaptar modificações rápidas no projeto e nas condições técnicas quando necessário (PRESSMAN, 2003).

Existem diversos métodos ágeis: *Scrum*, *Extreme Programming* (XP), *Feature Driven Development* (FDD), *Dynamic Systems Development Method*, *Adaptive Software Development*, *Crystal, Pragmatic Programming* e *Test Driven Development*. Para o estudo de caso deste trabalho será usado o *Scrum*, que foi aplicado em uma equipe de desenvolvimento de *software* e que será apresentado em detalhes no capítulo a seguir.

3. SCRUM

O nome *Scrum* surgiu da comparação entre desenvolvedores e jogadores de *Rugby. Scrum* é a denominação da rápida reunião que ocorre quando os jogadores de *Rugby* vão iniciar um lance. A primeira utilização deste termo surgiu em um estudo de Takeuchi e Nonaka no artigo "*The new product development game*", em fevereiro de 1986, pela Universidade de Harvard. Takeuchi e Nokada notaram que em projetos com equipes pequenas e multidisciplinares produziam melhores resultados, e associaram isto a formação *Scrum* do *Rugby*. No *Rugby* cada time age em conjunto, como uma unidade integrada. Nele, cada membro desempenha um papel específico e todos se ajudam em busca de um objetivo comum. E assim devem ser os times de desenvolvimento que adotam o método *Scrum* (RISING; JANOFF, 2000).

O método *Scrum* surgiu em 1995 e teve sua definição formalizada por Ken Schwaber, que trabalhou para consolidá-lo como método de desenvolvimento de software por todo o mundo (SCHWABER, 2001).

Apesar de ser uma abordagem relativamente nova, a utilização do método *Scrum* tem aumentado nos últimos anos, impulsionado pelas recentes pesquisas que mostram que seu uso aumenta a satisfação dos clientes e diminui o atraso em projetos em relação aos métodos tradicionais (MANN; MAURER, 2005).

O *Scrum* não define uma técnica específica para o desenvolvimento de *software* durante a etapa de implementação, ele se concentra em descrever como os membros da equipe devem trabalhar para produzir um sistema flexível, num ambiente de mudanças constantes. A ideia central do *Scrum* é que o desenvolvimento de sistemas envolve diversas variáveis (ambientais e técnicas) e elas possuem grande probabilidade de mudar durante a execução do projeto (por exemplo: requisitos, prazos, recursos, tecnologias etc.). Estas características tornam o desenvolvimento do sistema de *software* uma tarefa complexa e imprevisível, necessitando de um processo flexível e capaz de responder às mudanças (FRANCO, 2007).

Scrum é um processo bastante leve para gerenciar e controlar projetos de desenvolvimento de *software* e para criação de projetos de produtos. O *Scrum* segue as filosofias iterativa e incremental, ele se concentra no que é realmente importante: gerenciar o projeto e criar um produto que acrescente valor para o

negócio. O valor decorre da funcionalidade propriamente dita, do prazo em que ela é necessária, do custo e da qualidade (MARTINS, 2007, p. 253).

Sua abordagem é oposta a do modelo em cascata. Inicia-se a análise assim que alguns requisitos estiverem disponíveis, depois que alguma análise tiver sido feita, começam os trabalhos de projeto técnico (*design*), e assim por diante. Em outras palavras, o projeto trabalha em pequenos ciclos de cada vez (*Sprints*) até a conclusão do projeto final. Esta abordagem pode ser chamada de iterativa, e cada iteração consiste na captura de requisitos, um pouco de análise, um pouco de design e mais alguma programação e testes, culminando num ciclo iterativo com várias entregas (MARTINS, 2007, p. 253).

Segundo Schwaber (2001), o *Scrum* não é um processo previsível, ele não define o que fazer em todas as circunstâncias. Ele é utilizado em trabalhos complexos onde não é possível prever os acontecimentos e oferece um *framework* e um conjunto de práticas que torna tudo visível. Isso permite a equipe ter uma visão exata dos fatos ao longo do projeto e se necessário, realizar os devidos ajustes visando alcançar seus objetivos. Este é um dos pontos fortes do *Scrum*: “Adaptabilidade e Flexibilidade”.

O processo de controle do *Scrum* é sustentado por três pilares (SCHWABER, 2001):

- Transparência;
- Inspeção; e;
- Adaptação.

A transparência diz que os aspectos significativos do processo devem ser visíveis para os responsáveis pelos resultados do projeto. Na inspeção, a equipe tem que inspecionar o próprio trabalho durante seu progresso e verificar se o objetivo final será cumprido. Esta tarefa não pode afetar as atividades. A adaptação diz que, se um ou mais aspectos do processo são desviantes além dos limites aceitáveis, e que por consequência o produto final será inaceitável, então o processo ou a atividade devem ser ajustados. Isso precisa ser realizado o mais cedo possível para minimizar mais desvios.

O *Scrum* prescreve quatro fases formais para inspeção e adaptação, tal como descrito na seção *Ciclo do processo Scrum – Sprint*:

- Reunião de planejamento da iteração
- Reunião diária de *Scrum*
- Revisão da iteração
- Retrospectiva da iteração

3.1 Benefícios

A literatura pesquisada mostra que a utilização deste método pode gerar benefícios como (CARVALHO; MELLO, 2012):

- Aumento da satisfação de clientes, por meio da diminuição das reclamações;
- Melhoria na comunicação e aumento da colaboração entre envolvidos nos projetos;
- Aumento do retorno do investimento em projetos de novos produtos;
- Aumento da motivação da equipe de desenvolvimento de produtos;
- Melhoria da qualidade do produto produzido;
- Diminuição dos custos de produção;
- Aumento de produtividade da equipe de desenvolvimento;
- Diminuição no tempo gasto para terminar projetos de desenvolvimento de novos produtos; e;
- Diminuição do risco em projetos de desenvolvimento de novos produtos

3.2 Papéis e responsabilidades

O *Scrum* implementa um esqueleto interativo e incremental através de três papéis: o *Product Owner*, o *Scrum Master* e a equipe do projeto, também chamada de *Scrum Team*.

3.2.1 Product Owner

O *Product Owner* é o responsável pela visão de negócios e por representar os interesses das pessoas que apostam no projeto, provavelmente será um gerente de projeto, ou um patrocinador do projeto, um membro da equipe de marketing ou um cliente interno. É ele quem define e prioriza o *Product Backlog* e consegue a verba. Geralmente é o papel desempenhado pelo cliente (MARTINS, 2007, p. 255).

Suas principais responsabilidades são:

- Definir as funcionalidades do produto;
- Concentrar as informações vindas de usuários, *stakeholders* ou do mercado de maneira que se obtenha uma visão única dos requisitos do sistema;
- Sua maior responsabilidade é o Retorno sobre investimento (ROI - *Return on Investment*) do projeto;
- Priorizar o *Product Backlog*;
- Decidir sobre a data de término;
- Ajustar recursos e priorizar tarefas a cada *Sprint*, como necessário;
- Aceitar ou rejeitar os resultados dos trabalhos.

3.2.2 Scrum Master

O *Scrum Master* é responsável pelo processo *Scrum*, por ensiná-lo a todas as pessoas envolvidas no projeto, por implementá-lo, fazendo dele uma cultura na organização e ainda por garantir que toda a equipe siga as regras e as práticas do *Scrum* (SCHWABER, 2001).

O *Scrum Master* é uma mistura de gerente de projeto, facilitador e mediador. Ele é um líder e não somente um gerente, está sempre em contato com o *Product Owner.*

Qualquer membro da equipe pode ser um *Scrum Master*, entretanto normalmente é exercido por um Gerente de Projetos ou Líder Técnico. Entre as suas principais responsabilidades, temos (PAULA, 2012):

- Assegurar que a equipe de desenvolvimento funcione plenamente e seja produtiva;
- Ajudar na cooperação entre todas as funções e papéis do time;
- Remover obstáculos da equipe e assegurar que as práticas de *Scrum* estão sendo executadas com eficiência pelas pessoas envolvidas. Alguns obstáculos podem ser resolvidos pelo time, outros podem ser resolvidos através de vários times, e outros podem precisar de envolvimento da gerência, pois podem ser problemas relacionados à empresa que bloqueiam qualquer membro do time a fazer qualquer coisa. Como exemplo deste tipo de impedimento externo, temos as questões judiciais;
- Proteger a equipe de interferências externas;
- Assegurar-se de que a metodologia está sendo seguida, incluindo chamadas para reuniões diárias (*Daily Scrum Meeting*), revisões de atividade (*Sprint Reviews*) e reuniões de planejamento das atividades (*Sprint Planning*);
- Identificar quais atividades foram concluídas, quais foram iniciadas, quaisquer novas tarefas que foram descobertas e qualquer estimativa que possa ter mudado. Isto permite a ele atualizar sempre o *Burndown Chart*, que permite mostrar quanto trabalho falta para um *Sprint* acabar, dia por dia. Ele também tem que sempre olhar cuidadosamente para o número de tarefas em aberto. Estes trabalhos em aberto devem ser minimizados o máximo possível para garantir um trabalho sempre limpo e eficiente;
- O *Scrum Master* deve perceber e resolver problemas pessoais ou conflitos entre os integrantes do time. Este tipo de problema deve ser resolvido por diálogo com o time, ou então o *Scrum Master* terá que ter ajuda da gerência ou do departamento de Recursos Humanos. Alguns estudos apontam que 50% dos problemas de desenvolvimento acontecem por razões pessoais. O *Scrum Master* precisa estar sempre atento ao time para fazer dele totalmente funcional e produtivo.

3.2.3 Scrum Team

Por último, a equipe do projeto ou *Scrum Team* é o grupo de pessoas responsáveis por desenvolver ou construir as funcionalidades do produto. Algumas características das equipes de desenvolvimento são (PAULA, 2012):

- São auto gerenciadas, auto organizadas e multifuncionais;
- São equipes pequenas, compostas por cinco a dez membros (o recomendado são sete pessoas). Podem ser compostas por desenvolvedores e participantes externos (marketing, vendas, clientes, etc.);
- Demonstrar o resultado do *Sprint* para o *Product Owner* e outros *Stakeholders*;
- Deve ter a capacidade e o conhecimento técnico sobre todo o processo de desenvolvimento do produto;
- O time deve ter pessoas capazes de analisar a solução, codificá-la e testá-la sem necessitar de outros times ou outras pessoas;
- Definem metas de cada *Sprint*, junto ao *Scrum Master*, e especificam seus resultados de trabalho;
- Têm o direito de fazer tudo dentro dos limites das diretrizes do projeto para atingir a meta de cada *Sprint*;
- Organizam os trabalhos para atingir os objetivos dos *Sprints.*

3.3 Ciclo do processo do *Scrum – Sprint*

O coração do *Scrum* é o *Sprint*. O *Sprint* representa um ciclo, ele pode durar de duas a quatro semanas (RISING; JANOFF, 2000), mas é tipicamente mensal, pois 30 dias fornecem uma melhor visibilidade dos objetivos do projeto e comprometimento da equipe.

Um *Sprint* produz um produto visível, usável e que pode ser entregue, que implementa uma ou mais interações de usuários com o sistema (RISING; JANOFF, 2000). A ideia chave do *Sprint* é entregar uma funcionalidade que agregue valor ao

cliente, uma parte do sistema apresentável. Estas entregas parciais vão sendo implementadas até o produto final estar concluído, e à medida que forem surgindo novas funcionalidades desejadas, novos S*prints* vão sendo realizados.

*Os Sprint*s contêm e consistem na Reunião de Planejamento do *Sprint*, nas Daily *Scrum*s, no trabalho de desenvolvimento, na Revisão do *Sprint*, e na Retrospectiva de *Sprint*. Durante o *Sprint* (Schwaber, 2001):

- Não são feitas alterações que afetem a meta do *Sprint*;
- A composição da equipe de desenvolvimento permanece constante;
- Os objetivos de qualidade não diminuem; e,
- O âmbito do trabalho pode ser clarificado e renegociado entre o *Product Owner* e a equipe de desenvolvimento conforme a aprendizagem sobre o conteúdo.

Somente o *Product Owner* tem a autoridade para cancelar uma *Sprint,* embora a equipe de desenvolvimento e o *Scrum Master* podem influenciar na decisão. A decisão de cancelar ocorre se o projeto torna-se obsoleto, quando a empresa muda de direção. Em geral, devido a curta duração de um *Sprint*, raramente seu cancelamento acontece. Quando um *Sprint* é cancelado, todo o trabalho já terminado do *Product Backlog* é revisto pelo *Product Owner*, pois parte dele poderá ser comercializado. As atividades incompletas são re-estimadas e colocadas novamente no *Product Backlog*. O cancelamento de *Sprints* consome recursos, uma vez que a Reunião de Planejamento deve ser feita novamente.

A Figura 5 demonstra o processo completo do ciclo de uma *Sprint* e pode ser acompanhada para auxiliar no entendimento.

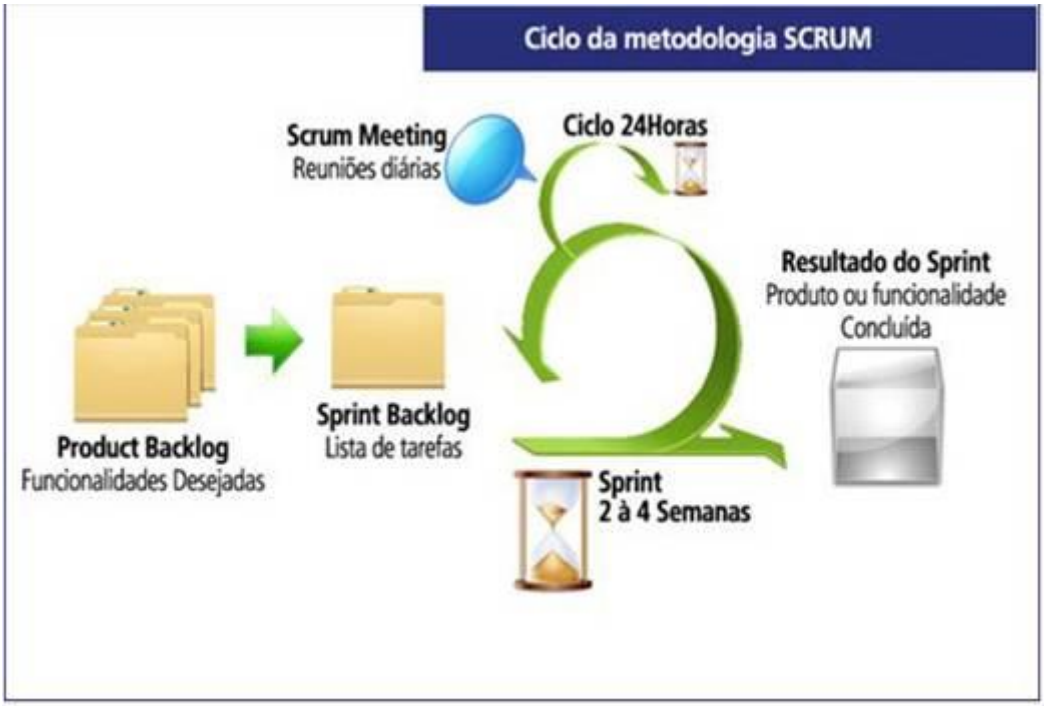

Figura 5 - Ciclo de vida de uma *Sprint*
Fonte: Adaptado de PAULA, 2012.

Um projeto se inicia com uma visão simples do produto que será desenvolvido. A visão pode ser vaga a principio e ir tornando-se clara aos poucos. O *Product Owner* então, transforma essa visão em uma lista de requisitos funcionais e não-funcionais que quando forem desenvolvidos reflitam essa visão. Essa lista, chamada de *Product Backlog* ou "*Backlog* do Produto" é priorizada pelo *Product Owner* de forma que os itens que gerem maior valor ao produto tenham maior prioridade (LIBARDI; BARBOSA, 2010). Esta lista não precisa estar completa logo no começo, ela pode ganhar outros itens no decorrer do projeto.

O ponto de partida é dividir o *Product Backlog* em releases e é esperado que o conteúdo, a prioridade e agrupamento do *Product Backlog* sofram mudanças a partir do momento que o projeto começa. Essas mudanças refletem mudanças nas regras e requisitos de negócios e no quão rapidamente a equipe pode transformá-lo em produto (SCHWABER, 2001).

3.3.1 Reunião de Planejamento do *Sprint* - *Sprint Planning Meeting*

Cada *Sprint* inicia-se com uma reunião chamada *Scrum Planning Meeting* (Reunião de Planejamento do Sprint). Esta reunião é realizada na presença do

Product Owner (representante do cliente ou o próprio cliente), do *Scrum Master* e do *Scrum Team*.

A Reunião de Planejamento do *Sprint* é limitada a oito horas para um *Sprint* de um mês. Para *Sprints* mais curtos, o evento é proporcionalmente menor (SCHWABER, 2001).

A Reunião de Planejamento do *Sprint* é dividida em duas partes, sendo que cada parte tem a metade da duração total reservada para esta atividade. Essas duas partes da Reunião de Planeamento do *Sprint* respondem às seguintes perguntas, respectivamente, segundo Schwaber:

- O que será entregue no incremento resultante do próximo *Sprint*?
- Como será alcançado o trabalho necessário para entregar o Incremento?

Na primeira parte da reunião, o *Product Owner* apresenta os itens, previamente ordenados, do *Product Backlog* a equipe de desenvolvimento e esta faz as perguntas que sejam suficientes para que eles possam, depois da reunião, definir quais atividades eles irão mover do *Product Backlog* para a *Sprint Backlog*. Apenas a equipe de desenvolvimento pode avaliar o que será possível entregar no próximo *Sprint*.

Após a equipe de desenvolvimento projetar quais itens do *Product Backlog* serão entregues na próxima *Sprint*, a equipe de *Scrum* cria a meta da *Sprint*. A Meta da *Sprint* é um objetivo que será cumprido dentro da *Sprint*, através da implementação do *Product Backlog*, e fornece orientação para a equipe de desenvolvimento sobre o que está sendo construído no incremento (SCHWABER, 2001).

Na segunda parte da reunião, será respondido como o trabalho escolhido irá ser realizado. O *Scrum Team* reúne-se, separadamente, para discutir o que foi dito e decidir o quanto eles se comprometem a fazer durante o próximo *Sprint*. Em alguns casos, haverá negociações com o *Product Owner*, mas será sempre prerrogativa do *Scrum Team* determinar o quanto eles podem se comprometer (PAULA, 2012). Os itens selecionados para a próxima *Sprint* mais o plano de desenvolvimento é chamado de *Sprint Backlog*.

A equipe de desenvolvimento geralmente começa por desenhar o sistema e as tarefas necessárias para implementar as funcionalidades selecionadas. O trabalho pode variar no tamanho ou no esforço estimado.

A equipe se compromete a executar essas tarefas no *Sprint* e o *Product Owner* se compromete a não trazer novos requisitos para a equipe durante o *Sprint*. Requisitos podem mudar (e mudanças são encorajadas), mas apenas fora do *Sprint*. Uma vez que a equipe comece a trabalhar em um *Sprint*, ela permanece concentrada no objetivo traçado para o *Sprint* e novos requisitos não são aceitos (LIBARDI; BARBOSA, 2010).

No final da reunião de Planeamento do *Sprint*, a equipe de desenvolvimento deve ser capaz de explicar ao *Product Owner* e ao *Scrum Master* como pretendem trabalhar enquanto equipe auto-organizada a fim de cumprir a Meta da *Sprint* e criar o incremento previsto (SCHWABER, 2001).

3.3.2 Reunião diária do Sprint - Daily Scrum Meeting

Diariamente o *Scrum Team* se reúne numa reunião de 15 minutos numa reunião chamada *Daily Scrum* para sincronizar o trabalho da equipe toda e informar o *Scrum Master* de eventuais impedimentos no trabalho (LIBARDI; BARBOSA, 2010). Essas reuniões costumam ser rápidas, objetivas e realizadas em pé, preferencialmente pela manhã, no mesmo local e horário, para reduzir a complexidade e evitar perder o foco do que está sendo desenvolvido e possíveis divergências de assuntos.

Esta reunião é eficiente para manter os membros da equipe cientes dos objetivos e evitar que o projeto perca o foco. Elas não representam uma forma de cobrança vinda de um gerente de projetos, mas é uma maneira de sincronizar a equipe às tarefas e relatar os impedimentos que podem estar interferindo no bom andamento do Sprint (PAULA, 2012).

Nas reuniões diárias, os únicos autorizados a falar são o *Scrum Master* e a equipe envolvida. Os demais participantes devem apenas ouvir, se falarem algo devem ser convidados a se retirar da reunião, pois podem querer mudar o rumo do projeto, tentando priorizar itens que são importantes para eles e não para o projeto ou seu objetivo imediato. O *Scrum Master* é o responsável por manter a ordem

nessa reunião, dar a palavra a um membro por vez, evitar desvio de foco do assunto e conversas paralelas (MARTINS, 2007, p. 274).

Cada membro do *Scrum Team* deve então responder cada uma destas três perguntas:

- O que você fez ontem?
- O que você fará hoje?
- Há algum impedimento no seu caminho?

Com essas informações, a equipe ganha conhecimento do que está acontecendo no projeto e o que cada membro da equipe esta fazendo em relação ao projeto. Qualquer impedimento relatado deve ser tratado o mais breve possível.

As Daily Scrums melhoram a comunicação, eliminam outras reuniões, ajudam a identificar e remover obstáculos ao desenvolvimento, a destacar e promover a rápida tomada de decisões e a melhorar o nível de conhecimento que equipa de desenvolvimento tem do projeto. Esta é uma reunião chave para inspecionar e adaptar (SCHWABER, 2001).

3.3.3 Reunião de revisão do Sprint - Sprint Review Meeting

No final do *Sprint* ocorre a *Sprint Review Meeting*, com o intuito de inspecionar o incremento e adaptar o *Product Backlog*, se necessário. Nesta reunião, o *Scrum Team* apresenta para o *Product Owner* o que foi desenvolvido durante o *Sprint*. O *Product Owner* é responsável por validar e/ou solicitar ajustes para que o projeto se torne adequado aos anseios do cliente.

Os participantes desta reunião incluem tipicamente: o *Product Owner*, o *Scrum Team*, o *Scrum Master*, a diretoria, clientes e engenheiros de outros projetos.

O ideal é que a equipe tenha concluído todos os itens do *Product Backlog* alocados para o *Sprint*. Segundo MARTINS (2007, p. 276) é uma reunião informal de quatro horas de duração, para *Sprints* de um mês, que ajuda a equipe de forma colaborativa, a obter um *feedback* e determinar quais os objetivos da próxima iteração, assim como a qualidade e as capacidades das funcionalidades produzidas.

Ao final da apresentação das funcionalidades desenvolvidas, os *stakeholders* são questionados, um a um, quanto às suas impressões, mudanças necessárias e alterações de prioridades. Possíveis rearranjos nos itens do *Product Backlog* ou funcionalidades que não foram entregues como esperado podem ser incluídas no próximo *Sprint*.

O resultado da Revisão do *Sprint* é um *Product Backlog* revisado que define prováveis itens para o próximo *Sprint*.

3.3.4 Retrospectiva do Sprint – Sprint Retrospective Meeting

Depois da *Sprint Review Meeting* e antes do próximo *Sprint* o *Scrum Master* se reúne com o *Scrum Team* numa última reunião: a *Retrospective Meeting*.

Na *Retrospective Meeting*, o *Scrum Master* encoraja o *Scrum Team* a revisar as práticas do *Scrum* e refletir sobre o que precisa ser feito para melhorar no próximo *Sprint* (LIBARDI; BARBOSA, 2010). Nessa reunião é debatido o que funcionou na *Sprint*, o que precisa ser melhorado e quais ações devem ser tomadas para colocar as melhorias em prática. Geralmente o *Sprint Retrospective* tem de três a quatro horas de duração (PAULA, 2012), para *Sprints* de um mês.

O objetivo da Retrospectiva do *Sprint* é, segundo SCHWABER:

- Verificar como correu o último *Sprint* no que diz respeito às pessoas, relações, processos e ferramentas;
- Identificar e ordenar os itens que correram melhores e potenciais melhorias; e;
- Criar um plano para implementar melhorias no modo como a equipe *Scrum* faz o seu trabalho.

No final da Retrospectiva do *Sprint*, a equipe *Scrum* deve ter identificado melhorias que irão implementar no próximo *Sprint*. A implementação destas melhorias no *Sprint* seguinte é a forma de adaptação à inspeção que a equipe faz a si própria. Embora as melhorias possam ser implementadas a qualquer momento, a Retrospectiva do *Sprint* fornece uma oportunidade formal para nos focarmos na inspeção e adaptação (SCHWABER, 2001).

Juntas, a *Sprint Planning Meeting*, a *Daily Scrum Meeting*, a *Sprint Review Meeting* e a *Sprint Retrospective Meeting* implementam as práticas de inspeção e adaptação empíricas (SCHWABER, 2001).

A Figura 6 mostra o Ciclo de Trabalho do *Scrum* explicado até aqui.

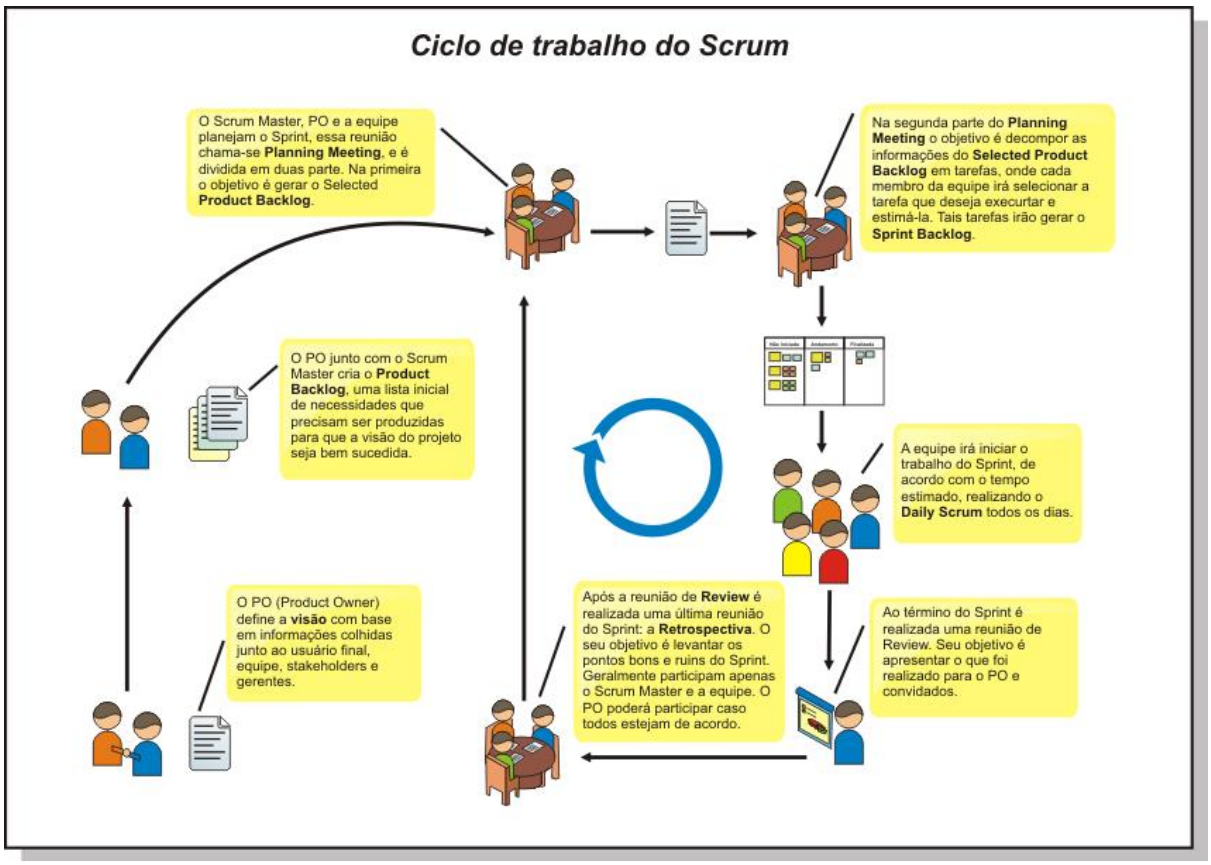

Figura 6 – Ciclo de Trabalho do *Scrum*
Fonte: http://evolvebr.wordpress.com/2011/01/21/voce-conhece-a-metodologia-scrum/

3.4 Artefatos *Scrum*

Os principais artefatos produzidos no processo do *Scrum* são o *Product Backlog, Sprint Backlog, Sprint Burndown Chart* e o *TaskBoard*. Eles serão exemplificados a seguir e detalhes dos mesmos serão explicados no estudo de caso.

Os artefatos representam um trabalho ou valor, de várias maneiras, que são úteis no fornecimento de transparência e oportunidades para a inspeção e adaptação (SCHWABER, 2001).

3.4.1 Product Backlog

O *Product Backlog* é uma lista contendo todas as funcionalidades desejadas para um produto. O conteúdo desta lista é definido e priorizado pelo *Product Owner*.

Este artefato é totalmente dinâmico, ele é modificado toda vez que se identifica algo que o produto precisa para ser mais apropriado, competitivo ou proveitoso (SCHWABER, 2001), por isso ele nunca está completo, ele sempre contém os requisitos mais conhecidos e melhores entendidos. O inicio do desenvolvimento do mesmo apenas estabelece os requisitos previamente conhecidos e melhor compreendidos no momento.

O *Product Backlog* existe enquanto um produto existe. Ele lista todas as características, funções, requisitos, melhorias e correções que constituem as mudanças a serem feitas para futuras versões do produto (SCHWABER, 2001). Todos os itens listados no *Product Backlog* possuem uma ordenação, uma estimativa e uma descrição.

Usualmente a ordenação dos itens do *Product Backlog* é feita baseado no valor, risco, prioridade e necessidade. Os itens do topo da lista são os mais críticos e necessitam de atividade de desenvolvimento imediato.

Mudanças nas regras de negócio, no mercado ou tecnologia podem causar alterações no *Product Backlog*.

3.4.2 Sprint Backlog

O *Sprint Backlog* é uma lista de tarefas que o *Scrum Team* se compromete a fazer em um *Sprint* como um potencial incremento de produto a ser entregue. Os itens do *Sprint Backlog* são extraídos do *Product Backlog* pela equipe, com base nas prioridades definidas pelo *Product Owner* e a percepção da equipe sobre o tempo que será necessário para completar as várias funcionalidades. A quantidade de itens do *Product Backlog* que serão trazidos para o *Sprint Backlog* é definida pelo *Scrum Team* que se compromete a implementá-los durante o *Sprint*. Os itens do *Sprint Backlog* são divididos em tarefas com detalhes suficientes para que possam ser executadas em até 16 horas (LIBARDI; BARBOSA, 2010).

O *Sprint Backlog* é um plano com detalhes suficientes para que as mudanças no progresso possam ser entendidas na *Daily Scrum* (SCHWABER, 2001). Apenas

a equipe de desenvolvimento pode mudar seu *Sprint Backlog* durante um *Sprint*, usualmente resulta uma realocação de atividades, ou seja, uma delegação de responsabilidades.

Durante um *Sprint*, o *Scrum Master* mantém o *Sprint Backlog* atualizando-o para refletir que tarefas são completadas e quanto tempo a equipe acredita que será necessário para completar aquelas que ainda não estão prontas. A estimativa do trabalho que ainda resta a ser feito no *Sprint* é calculada diariamente e colocada em um gráfico, resultando em um *Sprint Burndown Chart* (LIBARDI; BARBOSA, 2010).

3.4.3 Sprint Burndown Chart

O monitoramento do progresso do projeto é realizado através de dois gráficos principais: *Product Burndown Chart* e *Sprint Burndown Chart*. Estes gráficos mostram ao longo do tempo a quantidade de trabalho que ainda resta ser feito, sendo um excelente mecanismo para visualizar a correlação entre a quantidade de trabalho que falta ser feita (em qualquer ponto) e o progresso do time do projeto em reduzir este trabalho (LIBARDI; BARBOSA, 2010).

3.4.4 Taskboard

O *taskboard* (Figura 7) é um grande painel onde podem ser colocadas várias informações importantes para o acompanhamento do *Sprint*. O *Sprint Backlog*, ou seja, as atividades não iniciadas, as que estão em andamento e as concluídas ficam sempre visíveis e disponíveis para todos os interessados no projeto.

Algumas características do *taskboard* são (PAULA, 2012):

- Normalmente é desenhado em uma parede e as atividades são descritas em *post-its*;
- Apresenta uma visão geral do *Sprint*; e;
- Fica acessível a todos os interessados no projeto.

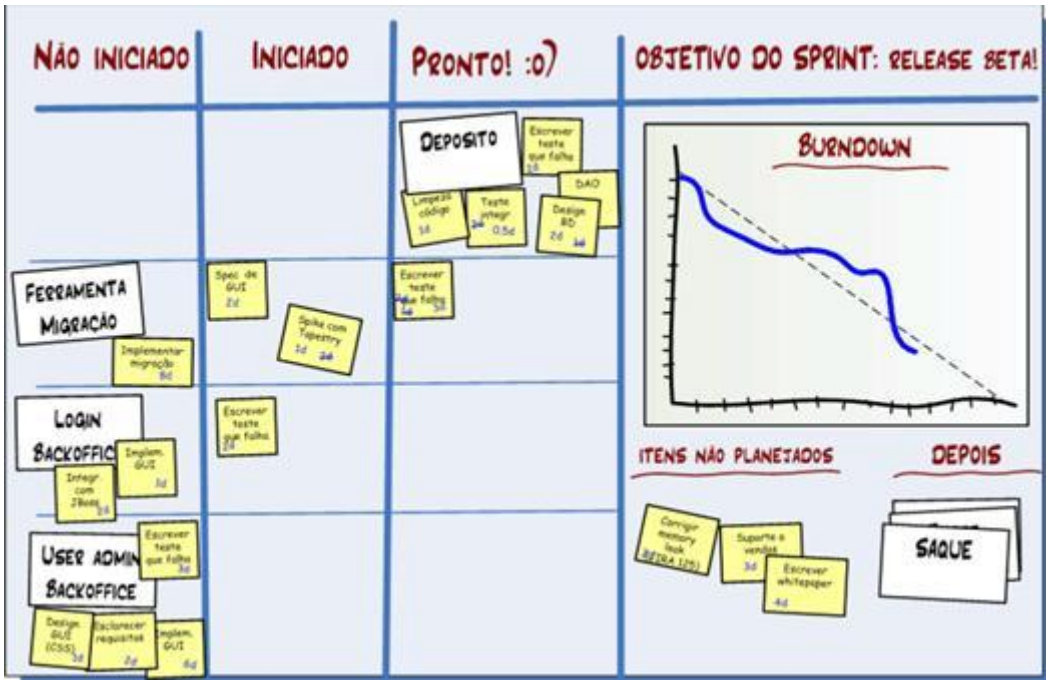

Figura 7 - *Taskboard*
Fonte: Adaptado de PAULA, 2012.

3.5 O *Scrum* para melhoria no gerenciamento de projetos de *software*

O *Scrum* é baseado nas metodologias ágeis voltado especificamente para o gerenciamento de *software* com base na colaboração com o cliente, trabalho em equipe, desenvolvimento iterativo e incremental, e com respostas às mudanças (RICO et al., 2009).

Tais características objetivam eliminar a dependência em realizar um extenso planejamento inicial e documentação de todos os requisitos do sistema (FERREIRA E COHEN, 2008), assim como é observado na utilização de métodos tradicionais.

Os processos tradicionais de desenvolvimento de *software* além de serem burocráticos muitas vezes são limitadores aos desenvolvedores. O fato de algumas empresas de *software* não possuir recursos ou experiência para processos pesados de produção de *software* pode ser a razão da baixa qualidade nos produtos, atrasos na entrega e muitas vezes o cancelamento do projeto.

Como alternativas a esse cenário surgem as metodologias ágeis, que apesar de possuírem documentação, são mais flexíveis, orientadas a entregas e possuem uma maior iteratividade no processo de desenvolvimento e codificação do produto. Preocupa-se mais com funcionalidade do que com documentação.

São adaptativas ao invés de serem preditivas, elas se adaptam a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento (PAULA, 2012).

O que diferencia as metodologias ágeis das tradicionais são o enfoque e os valores. As metodologias ágeis tem foco no cliente ao invés do processo.

Para ser considerada ágil a metodologia deve aceitar a mudança ao invés de tentar prever o futuro. Elas variam em termos de práticas e ênfases, mas compartilham algumas características, como desenvolvimento iterativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva (PAULA, 2012). Dessa forma, existem maiores possibilidades de atender aos requisitos do cliente, que estão sempre sendo alterados.

As metodologias convencionais devem ser aplicadas apenas em situações em que os requisitos do *software* são estáveis e requisitos futuros são previsíveis. Estas situações são difíceis de serem atingidas, uma vez que os requisitos para o desenvolvimento de um *software* estão a todo o momento sofrendo alterações. Dentre os fatores responsáveis por alterações nos requisitos estão a mudança de regras de negócio e dinâmica das organizações.

O *Scrum* não tem como objetivo rejeitar os processos e ferramentas e sim mostrar que a importância primária são as pessoas, o *software* executável, colaboração com o cliente e a resposta as rápidas mudanças.

As características do projeto irão determinar entre o uso das abordagens tradicional ou ágil. Os projetos que têm como natureza a inovação tecnológica inviabilizam o uso da abordagem tradicional, pois o risco de ser necessário alterar um produto depois da conclusão de uma fase de seu ciclo de vida é alto. A abordagem ágil consegue uma adaptação mais fácil nesses casos de mudança. A abordagem tradicional é mais adequada para projetos que necessitam de um forte planejamento e muita disciplina no processo, mas ela não promove tão ampla comunicação entre os membros de equipes e seus gerentes (PAULA, 2012).

Um processso *Scrum* bem implementado poderá alcançar o efeito *Toyota*: quatro vezes mais produtividade e 12 vezes mais qualidade. Um dos motivos principais no ganho de produtividade é que sua organização faz com que um *Scrum* funcional produza resultados de ótima qualidade de negócio na primeira vez,

evitando que se façam componentes que nunca serão usados pelo usuário. Esse problema atinge muitas organizações que não implementam *Scrum* (SCHWABER, 2006).

Outra vantagem de se utilizar o *Scrum* é a produtividade aliada ao conforto. Ele permite aos colaboradores desenvolverem *software* na medida em que reduzem a pressão de administração, pois oferece liberdade para que cada membro da equipe selecione seus próprios trabalhos e organizem entre si (PAULA, 2012).

Como toda metodologia ou processo, sempre existem vantagens e desvantagens advindas da sua adoção. Paula (2012) cita essas diferenças. Algumas das desvantagens observadas no *Scrum* são:

- Não é fácil de ser implementada, principalmente devido a resistência de mudanças culturais;
- Ausência de práticas de Engenharia de *Software*, pois é voltada para o gerenciamento do projeto e não para o desenvolvimento, podendo necessitar da associação de outras metodologias de Engenharia de *Software* simultaneamente;
- Pode ser mais difícil de aplicar em grandes equipes ou em equipes geograficamente distribuídas;
- A documentação do *software* pode acabar sendo desprezada, tornando insuficiente quando forem necessárias manutenções futuras; e;
- Sensação de informalidade, pois a documentação formal do escopo do *software* só é criada caso os envolvidos considerem necessário.

Todavia, diante de tantas vantagens já discutidas nos parágrafos anteriores, o *Scrum* ainda se mostra como uma opção capaz de driblar vários problemas atuais no gerenciamento de projetos de *software*. Os principais benefícios obtidos com a prática do *Scrum* na intenção de obter uma melhoria nos processos de gerenciamento de software podem ser assim resumidos:

- Melhor adaptação no gerenciamento de projetos com constantes mudanças, pois a grande maioria dos projetos se inicia sem total

detalhamento e sempre existem alterações nos requisitos ao longo do desenvolvimento;

- Encoraja a comunicação entre os membros da equipe e o espírito colaborativo. As chances de sucesso em projeto são muito maiores em times onde há uma boa comunicação, onde há entendimento e cooperação mútua;
- Por ser iterativo e incremental, permite a entrega antecipada de uma parte funcional do *software* ao cliente, que já poderá validar se está conforme o esperado. Caso não esteja, a equipe não perderá o mesmo tempo que perderia se esse *feedback* só fosse feito após a conclusão do *software*. Além de impedir que se implemente funcionalides que nunca serão utilizadas pelo cliente;
- A entrega do *software* utilizando Scrum é mais rápida do que se utilizasse alguma outra metodologia tradicional, uma vez que a codificação já se inicia logo nos primeiros *Sprints*, e não se espera ter antes uma extensa documentação do *software*;
- A qualidade no *software* utilizando Scrum é promovida pelos seguintes fatores: as iterações, a remoção de impedimentos, inspeção e adaptação, times multifuncionais e autonomia da equipe, o que significa menos pressão sobre cada membro;
- O *Scrum* traz um ROI mais rápido, pois minimiza a burocracia dos processos e se aproxima do cliente. Este está sempre envolvido, participando da demonstração do *software* ao final de cada *Sprint*, oferecendo *feedback* e atento às mudanças e suas consequências.

4 ESTUDO DE CASO

4.1 A empresa

Formada em Junho de 2006, através da fusão entre as empresas Axalto e Gemplus International S.A, a Gemalto é líder em segurança digital. Uma empresa multinacional, com sede em Amsterdã e que possui subsidiárias no mundo todo, que fornece soluções para segurança digital. A empresa desenvolve *softwares* seguros que são incorporados em dispositivos de segurança que a própria empresa personaliza, tais como cartão de pagamento, bilhetes de viagem, *tokens* de autenticação, passaportes eletrônicos, documentos de identificação entre outros.

Mais de um bilhão de pessoas ao redor do mundo utilizam os produtos e serviços da empresa em Telecomunicações, Serviços Financeiros, Governo Eletrônico, Gestão de Identidade, Conteúdo Multimídia, Gestão de Direitos Digitais, Segurança de TI, transporte público e tantas outras aplicações.

No Brasil, a empresa possui dois centros de personalização, um na cidade de Barueri em São Paulo, onde são personalizados cartões bancários e onde foi aplicado o estudo de caso deste trabalho e o outro na cidade de Pinhais em Curitiba, onde são fabricados os cartões bancários e também *Sim Cards*.

4.2 A equipe

A equipe na qual foi aplicado a metodologia *Scrum* foi a equipe de desenvolvimento de *software* para cartões bancários. A equipe é composta por cinco profissionais e um coordenador.

As principais atividades da equipe são:

- Desenvolver *softwares* que fazem a decodificação do arquivo do cliente e gere um arquivo binário no formato que a máquina entende, para a personalização do cartão;
- Desenvolver relatórios usando a ferramenta *Jasper Report;*
- Automatizar processos por meio de *procedures* usando a ferramenta *PL/SQL Developer* e utilizando a base de dados Oracle;

- criar *softwares* internos que atendam a necessidade dos funcionários.

4.3 Necessidade

A equipe de desenvolvimento não possui uma metodologia de trabalho. Os projetos são de curta duração (de uma a quatro semanas). Os projetos chegam dos consultores técnicos, os quais tem contato direto com os clientes (bancos da América Latina), entendem suas necessidades e criam o documento de especificação.

Antes da implantação da metodologia *Scrum*, a equipe de desenvolvimento trabalhava da seguinte forma: toda segunda-feira a coordenadora da equipe fazia uma reunião com o coordenador dos consultores técnicos. Nessa reunião era definido o que seria desenvolvido durante a semana, para qual desenvolvedor a atividade seria direcionada e a data de entrega da solução. Uma planilha era montada com essas atividades e passada para os desenvolvedores. Os mesmos tinham a missão de se auto planejar para cumprir o objetivo dentro do prazo esperado.

A mesma equipe que faz o desenvolvimento do *software* presta suporte à fábrica para qualquer problema durante a personalização. Isso significa que é rotina ter que parar o desenvolvimento para corrigir o problema. Isso causa dois impactos visíveis:

- O profissional perde o foco do que esta sendo desenvolvido;
- Na maioria das vezes, o projeto atrasa; e;
- Falta de prioridade no que esta sendo desenvolvido.

A necessidade da implantação do *Scrum* surgiu no momento em que havia um claro diagnóstico sobre a realidade da equipe de desenvolvimento. A ideia seria de organizar melhor tanto o método de trabalho quanto o planejamento das atividades a serem realizadas.

4.4 Planejamento e implantação do *Scrum*

O planejamento iniciou-se com uma reunião interna para definir como seria a adoção da metodologia *Scrum*. Nesta reunião, foram definidos dois membros da equipe para a realização de um curso sobre *Scrum*, que seria pago pela empresa. Pelo tempo de empresa, um engenheiro e a coordenadora da equipe foram os escolhidos. O curso ocorreu na empresa *Adaptworks*, uma empresa focada em treinamentos e no desenvolvimento de *software* utilizando metodologias ágeis. O curso teve duração de 16 horas e foi dividido em dois dias.

Após a realização do curso, os dois membros da equipe foram responsáveis por replicar o conhecimento para os demais, treinamento que durou mais um dia.

Com a equipe inteira tendo conhecimentos sobre *Scrum*, os papéis foram definidos para a adoção.

A equipe de desenvolvimento decidiu que seria interessante implantar totalmente as seguintes práticas ágeis do *Scrum*:

- *Scrum Master;*
- *Product Owner;*
- *Scrum Team*
- *Sprint*;
- *Product Backlog*;
- Estimativas;
- Reunião de planejamento do *Sprint*;
- Reunião diária;
- Reunião retrospectiva;
- *Taskboard*.

O *Scrum Master* escolhido foi um dos engenheiros. Apesar de normalmente este papel ser exercido por um líder técnico ou um gerente, esta escolha foi feita porque este era o membro da equipe que tinha mais conhecimento do método *Scrum* e metodologias ágeis.

O *Product Owner* escolhido foi o coordenador dos consultores técnicos, visto que ele era o membro que tinha uma visão mais ampla sobre todos os projetos que

seriam desenvolvidos. Preferencialmente, este papel deveria ser desenvolvido pelo próprio cliente. Entretanto, por regras de segurança da empresa, onde apenas pessoas autorizadas podem ter acesso às áreas seguras onde são desenvolvidos os softwares e por hierarquia, ficou definido em eleger o *Product Owner* sendo um profissional da empresa.

O *Scrum Team* (time) foi composto pelos demais membros da equipe, incluindo a coordenadora da equipe.

A *Sprint* ficou definida como sendo de uma semana, tendo em vista o curto prazo definido pelo cliente para o desenvolvimento dos projetos, de acordo com a necessidade e regras de negócio.

Para a “Estimativa”, a coordenadora da equipe define, no inicio de cada *Sprint*, uma porcentagem para a meta que deverá ser alcançada no final da *Sprint*, levando em consideração alguns fatores.

Para implementar o *taskboard* um quadro branco foi comprado. Uma vez que a equipe de desenvolvimento trabalha com mais de um projeto ao mesmo tempo, o *taskboard* foi adaptado com linhas, sendo que cada linha é um projeto. Dentro das linhas, um *post-it* é colado para cada atividade a ser realizada. O *post-it* na cor amarela claro é a atividade em si, na cor amarela escuro significa uma parada que tivemos que realizar para atender um suporte. Na fábrica, o suporte sempre tem prioridade em relação ao desenvolvimento. O *post-it* na cor rosa significa um impedimento do projeto, algo que necessite ser realizado primeiro para que o projeto tenha andamento.

Na primeira coluna da *taskboard* são os projetos da *Sprint*. A segunda coluna são as atividades do projeto, inicialmente em “*to do*” (para fazer). Conforme o *Scrum Team* inicia uma atividade, a mesma é passada para a terceira coluna “*doing*” (fazendo) e quando esta termine, o *post-it* é transferido para a quarta coluna “*done*” (feito). Para os impedimentos, *post-it* de cores rosas, são colados do lado direito da tabela com uma breve descrição do problema, que membro prestou o suporte, a data e o tempo para que o problema fosse corrigido. Isso foi definido para que, no final da *Sprint* fosse possível calcular com maior precisão o OTD (*On time delivery*) que deve, no mínimo, ser igual a meta definida no começo da *Sprint*. O OTD mensura a porcentagem dos projetos entregues no prazo. Se um projeto não foi entregue no prazo, mas teve um impedimento, é analisado caso a caso para a medição do OTD.

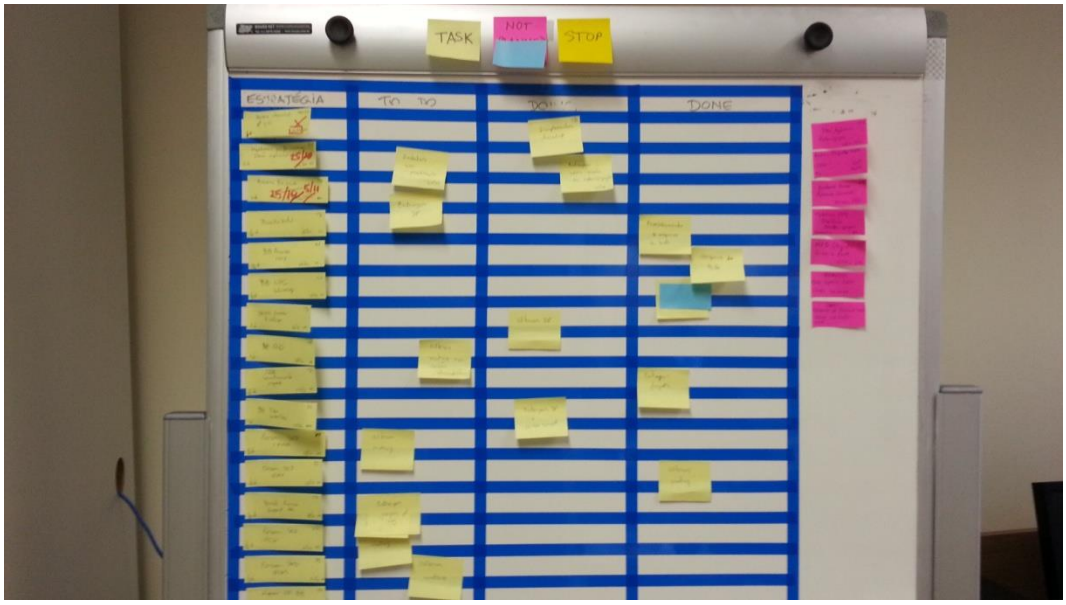

Figura 8 - *Taskboard* utilizado pela equipe.

4.5 Primeira iteração

A primeira iteração utilizando a metodologia *Scrum* iniciou-se com o *Product Owner* fazendo uma reunião com todos os consultores técnicos. Essa reunião serviu para levantar todos os projetos que deveriam ser desenvolvidos, as prioridades de cada um e a data de entrega.

Uma segunda reunião foi feita, a “Reunião de planejamento do *Sprint*”, entre o *Product Owner*, o *Scrum Master* e o *Scrum Team*, onde o *Product Owner* mostrou para os demais toda a lista de projetos a ser desenvolvido nas semanas seguintes. Nesta mesma reunião, o *Scrum Master* junto com o *Scrum Team*, dentro das limitações do momento, definiram os projetos que seriam possíveis cumprir, criando-se assim o *Product Backlog* e as datas de entrega, que deveriam ser cumpridas.

Atividade	**Prioridade**	**Data de entrega**
Matriz de produtos Itaú	1	7/10/2013
Matching Itaú	2	7/10/2013
Arquivo de retorno Itaú	3	7/10/2013
Lista de postagem Itaú	4	8/10/2013
Automatizar o FTP	5	8/10/2013
Checklist de arquivo	6	9/10/2013
Matriz de produtos Bradesco	7	9/10/2013
Matching Bradesco	8	10/10/2013
Arquivo de retorno Bradesco	9	10/10/2013
Lista de postagem Bradesco	10	10/10/2013

Modelagem da base	11	11/10/2013
Atualizar o Chord	12	11/10/2013

Tabela 1 - *Product backlog* da equipe.

Atividade	**Prioridade**	**Custo horas**
Matriz de produtos Itaú	1	8
Matching Itaú	2	4
Arquivo de retorno Itaú	3	6
Lista de postagem Itaú	4	4
Automatizar o FTP	5	8
Checklist de arquivo	6	3
Matriz de produtos Bradesco	7	6
Matching Bradesco	8	4
Arquivo de retorno Bradesco	9	6
Lista de postagem Bradesco	10	4
Modelagem da base	11	10
Atualizar o Chord	12	15

Tabela 2 - *Product backlog* da equipe com o custo de horas de cada atividade.

A estimativa foi calculada pela coordenadora da equipe, definindo a porcentagem da meta que deveria ser atingida no final da iteração.

As reuniões diárias não aconteceram diariamente. Alguns dias não ocorreram a reunião, devido a problemas urgentes na fábrica a serem corrigidos, que demandaram tempo. Notou-se também um receio e falta de vontade da equipe em ter que parar o trabalho para fazer a reunião diária. A reunião seguiu, na maioria dos dias, a metodologia, onde apenas o *Scrum Master* falava e os membros do *Scrum Team* apenas respondiam as perguntas.

O *taskboard* foi utilizado de forma correta, a equipe movimentou os *post-it* no momento certo, antes de iniciar uma atividade, ao finalizar uma atividade e nos impedimentos. A novidade ficou neste último, uma vez que foi possível comprovar o atraso na entrega da atividade. Item que antes, era bastante criticado.

A reunião de retrospectiva ocorreu de forma correta. A coordenadora da equipe, após calcular o OTD, apresentou para a equipe o resultado. Foi revisado, ainda que de forma rápida, todos os impedimentos da *Sprint* e possíveis ações para que os mesmos não ocorram mais. Duas ações foram levantadas e definiu-se que seriam implementadas na próxima *Sprint*. As atividades foram rapidamente comentadas, tendo uma maior ênfase nas que atrasaram a entrega. A equipe

também ficou surpresa com o baixo rendimento, levanto em consideração OTD x meta.

4.6 Segunda iteração

Para a segunda iteração, decidiu-se priorizar os itens que não foram completamente executados na iteração anterior, como a reunião diária.

A reunião de planejamento da *Sprint* continuou sendo executada da mesma forma.

A estimativa foi melhorada em relação a iteração anterior, levando em consideração os impedimentos de forma mais realística.

Houve um envolvimento maior da equipe para cumprir a meta. Com o painel de prioridades, foi visível para todos se determinada atividade atrasasse e notou-se uma preocupação dos membros da equipe determinados a ajudar a todos para que nada atrasasse. Notou-se também uma autossuficiência da equipe no gerenciamento, o que diminuiu a quantidade de horas extras. No meio da *Sprint* já era possível ver no painel de atividades que a maioria das tarefas já estavam concluídas.

A reunião diária foi melhor executada nesta iteração. Um horário foi definido e, independente dos impedimentos, a regra era para executar o item. A reunião aconteceu todos os dias no mesmo horário e seguiu corretamente a metodologia.

Na reunião de retrospectiva, foram levantados todos os impedimentos ocorridos durante a *Sprint*, porém, não foi criada nenhuma ação para trata-los, visto que, as ações da iteração passada não foram executadas até o fim. Foi definido que as ações seriam as mesmas, até porque parte dos impedimentos ocorridos podem também ser tratados pelas ações levantadas na *Sprint* anterior.

4.7 Demais iterações

As reuniões diárias passaram a ser mais produtivas quando o time começou a se auto gerenciar corretamente.

Durante as *Sprints* também se passou a fazer o uso do *backlog* de impedimentos. Sua utilização possibilita mitigar os riscos na prática e ter um maior

controle sobre eles. Ações sobre cada risco foram criadas e priorizadas pelo *Scrum Master*.

Os itens foram executados corretamente e notou-se um envolvimento maior da equipe. A confiança na metodologia também cresceu quando foi possível visualizar os resultados obtidos com o passar das iterações.

Sprint	**Meta**	***Stops***	***Not Planned***	**Atividades**	**Atividades concluídas**
1	100%	7	12	20	20
2	43%	1	19	14	6
3	62%	1	15	13	8
4	53%	5	23	15	8
5	71%	2	16	14	10

Tabela 3 - Resumo das S*prints*.

Conforme observado na Tabela 3, a quantidade de paradas da atividade devido a algum impedimento técnico (*stops*) diminuiu drasticamente em relação à primeira *Sprint*, devido às ações tomadas pelo *Scrum Master*, que foram ficando mais eficientes com o passar das iterações. As ações para os impedimentos de suporte (*not planned*) foram eficientes, apesar dos números dizerem o contrário. Neste item, só não melhorou por conta de novos impedimentos de diferentes tipos terem acontecido.

Notou-se uma diminuição nas atividades (*tasks*) da *Sprint*, devido ao melhor gerenciamento do time com relação ao que é possível cumprir dentro do prazo.

Sprint	**O que foi bom?**	**O que foi ruim?**	**Como melhorar?**
1	**Meta 100% atingida**	- 4 paradas por troca de prioridade na atividade - 3 paradas aguardando retorno do consultor técnico	- Adicionar linhas no quadro para visualizar melhor as *tasks* - OK
2	**Nada e meta muito abaixo da expectativa**	- Faltou fazer o *review* na segunda-feira - Retrabalho de *softwares*	- Adicionar no *checklist* ação para entregar matriz de *matching* e matriz de *keys* - Treinar consultor técnico para usar o *Split Tool* - OK - Identificar solução para remover atributo de *input file* do Banco do Brasil
3	**Nada**	- Muitos retrabalhos de matriz e software	- Concluir ação da matriz de *matchings* e *keys* - OK - Definir fluxo de desenvolvimento

			de arquivos de retorno - *Reworks* de software vão para o fim da lista de estratégias
4	**Nada**	- Muitos retrabalhos de matriz	- Entregar somente componentes alterados - Entregar informações de próximas atividades por e-mail
5	**Somente um retrabalho de atividade durante a semana**		- Definição de "*Ready*" para projetos - Entrega de documentação de desenvolvimento e manual de usuário. - Implementar *checklist* de *output files*

Tabela 4 - Resumo das S*prints* e as ações.

A Tabela 4 mostra as ações levantadas pela equipe para tratar os pontos negativos de cada *Sprint*. Nota-se uma melhora dos pontos negativos da *Sprint* 5 em relação a *Sprint* 1.

4.8 Melhorias observadas

Apesar das metodologias ágeis serem razoavelmente novas, os resultados obtidos são favoráveis. Paula (2012) cita um exemplo interessante comparando as metodologias ágeis e tradicionais: um estudo mostrou que os projetos usando os métodos ágeis obtiveram melhores resultados em termos de cumprimento de prazos, de custos e padrões de qualidade. Além disso, o mesmo estudo mostra que o tamanho dos projetos e das equipes que utilizam as metodologias ágeis têm crescido.

Apesar de serem propostas idealmente para serem utilizadas por equipes pequenas e médias (até 12 desenvolvedores), aproximadamente 15% dos projetos que usam metodologias ágeis estão sendo desenvolvidos por equipes de 21 a 50 pessoas, e 10% dos projetos são desenvolvidos por equipes com mais de 50 pessoas, considerando um universo de 200 empresas usado no estudo.

Para equipes grandes, também é possível obter sucesso com o uso da metodologia *Scrum*, mas para isso a equipe precisa ser dedicada e motivada para aplicar a mudança. São muitos e visíveis os benefícios do uso da metodologia

Scrum e para que uma organização adote o processo, é necessária uma mudança na cultura, muito empenho e recursos.

Foi observado também que as práticas do *Scrum* podem ser aplicadas em qualquer contexto, no qual pessoas precisem trabalhar juntas para atingir um objetivo comum. Ele é recomendado para projetos nas áreas de *software*, automotiva, telecomunicações e, principalmente, de pesquisa e inovação. Apresenta-se ideal para projetos dinâmicos e suscetíveis a mudanças de requisitos, sejam eles novos ou apenas modificados (PAULA, 2012). No entanto, para aplicá-lo, é preciso entender seus conceitos, ciclos de vida das atividades, papéis, responsabilidades.

A abordagem ágil assim como a abordagem tradicional possui características positivas e negativas, sendo que a principal diferença entre as duas está no conjunto de pressupostos de cada uma. É possível afirmar, ainda, que existe uma sinergia muito grande entre as duas metodologias, ou seja, uma pode complementar a outra (PAULA, 2012).

Assim como as metodologias convencionais de *software*, a metodologia ágil não é perfeita, existindo pontos negativos e positivos, assim como não há uma única solução para todos os casos. As metodologias ágeis agregam bastante aos projetos de *software*, com alto grau de mudanças, escopos não muito bem definidos que podem ser alterados a qualquer momento ou que precisam de resultados em um curto prazo.

Por outro lado, quando a organização já possui experiência e maturidade em situações que existem fortes requisitos, amarrados com prazos e escopo, pode ser mais vantagem o uso de uma abordagem mais tradicional.

Observaram-se na equipe as seguintes melhorias após a aplicação da metodologia *Scrum*:

- Autogerenciamento dos membros da equipe em relação as atividades e suas responsabilidades;
- Comunicação mais ativa e eficaz;
- Aumento da colaboração entre envolvidos;
- Aumento da iniciativa e motivação da equipe;
- Diminuição dos prazos para entrega das atividades;

- Diminuição do risco do projeto falhar;
- As estimativas agora tem um senso mais realístico do que antes;
- Ações criadas durante as iterações para tratar problemas.

5 CONCLUSÃO

Pode-se afirmar que a contribuição principal da presente pesquisa é além de explicar de forma sucinta as ideias da metodologia ágil e do *Scrum* especificamente, mostrar o impacto da implantação do *Scrum* nos projetos de desenvolvimento de novos produtos de *software* de uma empresa de base de segurança digital.

Antes da aplicação da metodologia no processo de trabalho, os membros da equipe não tinham um informativo visual dos projetos que estavam em andamento e não eram eficientes em gerenciar suas próprias atividades ao longo da semana. Além disso, eram constantes as perguntas por parte da gerência sobre o status de determinado projeto. Não havia também uma maneira eficiente de avaliar se as metas estavam sendo cumpridas e se estavam dentro do prazo estabelecido.

Como já explicado no capítulo anterior, foi possível identificar uma melhoria satisfatória no processo de desenvolvimento de *software* da equipe. Uma melhoria significativa foi o aumento de produtividade e maturidade da equipe e o autogerenciamento. Também foi possível medir o atingimento da meta.

Apesar de não fazer parte dos objetivos da empresa ter a certificação CMMI [2] no momento, é um ótimo comparativo para mensurar o nível de maturidade da equipe.

A certificação CMMI é um modelo que traz muitos benefícios para a empresa, visto seu grande reconhecimento no mercado. O modelo atesta a maturidade das equipes de trabalho tanto na gestão quanto no desenvolvimento e manutenção de *software*.

Um comparativo rápido entre o CMMI e a metodologia *Scrum* é que o CMMI trata do que fazer, enquanto o *Scrum* trata de como fazer.

O modelo CMMI é baseado em melhores práticas para desenvolvimento e manutenção de produtos, sendo que suas atividades endereçam áreas de Engenharia de Sistemas, Engenharia de *Software* e Integração de Produtos.

Os benefícios do modelo são:

[2] Embora não seja parte integrante deste trabalho uma análise mais aprofundada sobre a certificação CMMI (*Capability Maturity Model Integration* ou Modelo Integrado de Maturidade), o modelo está dividido em cinco níveis de maturidade e 24 áreas de processos. Os níveis de maturidade atestam o grau de evolução que uma organização se encontra em determinado momento e esse nível sobe conforme a empresa cresce de maturidade. Os níveis de maturidade também funcionam como um indicativo de melhoria de processos da empresa, considerando atividades de gerenciamento de projetos, prazos e custos.

- Cria procedimentos e *checklists*;
- Fortalece o processo de auditoria
- Reduz o retrabalho;
- Replica melhores práticas.

Dentro do contexto dos cinco níveis de maturidade, a empresa avaliada possui certificação nível três nas filiais sediadas na França e Cingapura. Nestes países o uso de boas práticas, processos, desenvolvimento contínuo e estratégia são fortemente trabalhados. No Brasil nenhuma certificação foi obtida ainda, até porque a empresa é relativamente nova, após a fusão.

Avaliando a filial de Barueri, onde foi aplicado o estudo de caso, a empresa se encaixa no nível um até o momento da aplicação do *Scrum*. Os fatores para essa análise são:

- A falta de processos implicava no retrabalho a cada novo projeto, inviabilizando também a mensuração dos objetivos atingidos em relação às metas traçadas;
- Baixa qualidade apresentada no *software*;
- Atrasos na entrega de projetos;
- Falhas na comunicação.

Após o uso da metodologia *Scrum*, apesar de ainda praticar de forma imatura, a empresa sustenta-se para almejar o nível dois da certificação CMMI.

Para a melhoria contínua da Gemalto e o atingimento de um nível acima, os próximos passos são ações de melhoria em:

- Foco em produtividade;
- Qualidade na entrega do produto;
- Entrega dentro do prazo estabelecido;
- Criar métodos para medir as entregas e a satisfação do cliente;
- Criar medidores de resultados;
- Realizar testes mais eficientes antes de colocar um *software* em produção; e;

- Controlar as entregas para fácil rastreabilidade.

Estas melhorias são benéficas para a garantia e satisfação do cliente, melhores resultados e processos enxutos e eficazes. Além disso, com a padronização de alguns processos, é possível o reuso de partes do produto ou estratégia em futuros projetos.

A adoção de uma metodologia de forma gradativa e em constante aperfeiçoamento de melhoria de processos pode permitir também que a empresa, ao necessitar certificar-se no modelo CMMI, esteja preparada e consiga de forma menos traumática e com um custo menor para as adaptações.

Após a aplicação da metodologia *Scrum*, apesar dos benefícios e resultados obtidos, não foi possível identificar se houve aumento da qualidade do produto entregue e da satisfação do cliente. Uma proposta para futuras pesquisas seria a identificação destes fatores. Além disso, outra proposta para um futuro trabalho seria a mensuração quantitativa de certos itens como os custos de produção, tempo gasto no projeto e o índice de reclamações.

Concluiu-se que o método *Scrum* foi condizente com a realidade da equipe de *software*, pois se mostrou focado em resultados, na comunicação da equipe e interação com os clientes, sendo um método adequado para gestão de projetos de *software*.

Percebeu-se que os pontos críticos da implantação foram a falta de conhecimento em metodologias ágeis pela equipe e a dificuldade em seguir o processo, ponto que foi aprimorado após algumas iterações. Um dos fatores de sucesso na implantação foi o comprometimento da equipe. Os membros do time concordaram que o grande empenho dos envolvidos foi fator fundamental para alcançar o objetivo.

Após a implantação, a equipe continuará fazendo uso da metodologia *Scrum* para futuros projetos.

REFERÊNCIAS

BECK, K., et al.: **Manifesto for Agile Software Development** < http://www.agilealliance.org/the-alliance/the-agile-manifesto/>. Acesso em Outubro de 2013.

KALERMO, J., RISSANEN, J.: **Agile Software Development in Theory and Practice**, Dissertação de Mestrado. University of Jyväskylä. Finland. 2002.

RISING, L., JANOFF,N.S.: **The Scrum Software Development Process for Small Teams.** IEEE Software, Vol. 17, No. 4, July-August 2000.

LEFFINGWELL, D., MUIRHEAD, D., **Tactical Management of Agile Development: Achieving Competitive Advantage**. Boulder, Colorado. 2004.

MANN, C.; MAURER, F.: **A Case Study on The Impact of Scrum on Overtime and Customer Satisfaction.** IEEE Computer Society, 2005. p. 70-79.

CARVALHO, B. V; MELLO, C. H. P.: **Aplicação do Método Ágil Scrum no Desenvolvimento de Produtos de *Software* Em Uma Pequena Empresa de Base Tecnológica.** Gest. Prod., São Carlos, v. 19, n. 3, p. 557-573, 2012.

SCHWABER, K.; BEEDLE, M.: **Agile Software Development With Scrum**. 1ª Edição. Upper Saddle River: Prentice-Hall. 2001.

FRANCO, E. F.: **Um Modelo de Gerenciamento de Projetos Baseado nas Metodologias Ágeis de Desenvolvimento de Software e nos Princípios da Produção Enxuta.** Dissertação de Mestrado. Escola Politécnica da Universidade de São Paulo. 2007.

MARTINS, José Carlos Cordeiro. **Técnicas Para Gerenciamento de Projetos de *Software*.** Rio de Janeiro, Editora Brasport, 1ª Edição, 2007.

PAULA, R. S. A.: **Scrum na Melhoria do Gerenciamento de Projetos de *Software*.** Revista Engenharia de *Software*. Edição 23, 2012.

LIBARDI, P. L. O; BARBOSA, V.: **Métodos Ágeis.** Trabalho de Conclusão de curso. Faculdade de Tecnologia – Limeira, São Paulo, 2010.

RICO, D. F., SAYANI, H. H., AND SONE, S.: **The Business Value of Agile Software Methods: Maximizing Roi With Just-in-time Processes and Documentation.** J. Ross Publishing, 2009.

DAMKE, Daniele Erica; MORAES, Patrícia Freitas de. **Engenharia de Software: Aplicação da Abordagem GQM para a Definição de um Processo de Engenharia de Requisitos de *Software* Embarcado**. Monografia, Brasília: Universidade Católica de Brasília, 2008.

SOMMERVILLE, Ian; **Engenharia de Software**. 6. ed. São Paulo: Addison Wesley, 2003.

PRESSMAN, Roger S. **Software Engineering: a practitioner's approach**. 5. ed. McGrawHill, 2003.

LESSA, Rafael Orivaldo; JUNIOR, Edson Orivaldo. **Modelos de Processos de Engenharia de Software**. Artigo. Palhoça/SC: UNISUL, 2012.

SILVA, Meire Elen Alves da. **Metodologia Ágil de Gerenciamento de Projetos - Scrum**. Monografia. Taquaritinga/SP: Faculdade de Tecnologia, 2010.