**UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**MAURÍCIO SOPRANA COELHO**

# REENGENHARIA DE SOFTWARE APLICADA NA MANUTENÇÃO DE SISTEMAS POR MEIO DA METODOLOGIA ÁGIL NO MÉTODO SCRUM

**CRICIÚMA**
**2012**

**MAURÍCIO SOPRANA COELHO**

# REENGENHARIA DE SOFTWARE APLICADA NA MANUTENÇÃO DE SISTEMAS POR MEIO DA METODOLOGIA ÁGIL NO MÉTODO SCRUM

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Prof Msc Ana Claudia Garcia Barbosa

**CRICIÚMA**

**2012**

**MAURICIO SOPRANA COELHO**

# REENGENHARIA DE SOFTWARE APLICADA NA MANUTENÇÃO DE SISTEMAS POR MEIO DA METODOLOGIA ÁGIL NO MÉTODO SCRUM

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Criciúma, 25 de junho de 2012.

**BANCA EXAMINADORA**

Prof. Ana Claudia Garcia Barbosa - Mestre - (UNESC) - Orientador

Prof. Gustavo Bisoghin - Mestre - (UNESC)

Prof. Luciano Antunes - Mestre - (UNESC)

**À minha família e amigos.**

## AGRADECIMENTOS

Agradeço a Deus pela saúde e força espiritual concedida durante todos esses anos.

A todos os meus amigos pela amizade e presença em todos os momentos de minha vida.

A minha orientadora Ana Cláudia pelo apoio, ensino e disponibilidade em me auxiliar nesse estudo.

Por último, mas não menos importante, agradeço aos meus pais, por sempre me incentivarem e me apoiarem nesta caminhada.

Muito obrigado a todos!

**“Descobrir o verdadeiro sentido das coisas é querer saber demais”**

**O Teatro Mágico**

## RESUMO

Este trabalho visa utilizar a manutenção de softwares para melhoramento de projetos já existentes utilizando a metodologia ágil e a reengenharia de software como base de estudo. O método a ser utilizado para desenvolvimento desde trabalho é o método Scrum, que é considerado um método ágil e flexível. O Scrum tem o objetivo de facilitar tanto a criação quanto a manutenção de um software, sendo iterativo, incremental e voltado para os objetivos reais do usuário e da equipe de manutenção. A reengenharia de software utilizada através do método Scrum visa reestruturar um software legado, economizando em todos os sentidos o projeto de pequenas, médias e grandes empresas. As metodologias ágeis vêm crescendo de forma rápida e sólida, tornando todos os processos de softwares cada vez mais participativos, tanto para o usuário quanto para a equipe de manutenção.

**Palavras-chave**: Scrum. Agilidade. Manutenibilidade. Reengenharia.

## ABSTRACT

This work aims to use the software to improve maintenance of existing projects using the agile methodology and software reengineering based study. The method to be used for development of this work is the Scrum method, which is considered to be a nimble and flexible. Scrum aims to facilitate both the creation and maintenance of software, being iterative, incremental and focused on the real objectives of the user and maintenance staff. The re-engineering software used by the Scrum method aims to restructure legacy software, saving in every way the design of small, medium and large companies. The agile methodologies have been growing fast and solid, making all software processes more participatory for both the user and for the maintenance team.

Keywords: Scrum. Agility. Maintainability. Reengineering.

# LISTA DE ILUSTRAÇÕES

# LISTA DE QUADROS

**SUMÁRIO**

# 1 INTRODUÇÃO

De acordo com Sommerville (2003) a engenharia de software se ocupa em todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação.

É crescente o número de pesquisas em engenharia de software sobre o uso da metodologia ágil no método Scrum na manutenção de sistemas e algumas empresas tendem a analisar seus conceitos e suas motivações.

Segundo Pressman (2006) agilidade tornou-se atualmente um método bastante utilizado quando se descreve um processo moderno de software. Uma equipe ágil é uma equipe eficaz, capaz de responder adequadamente a modificações. Modificação é aquilo para o qual o desenvolvimento de software está principalmente focado.

Em projetos onde ocorrem diversas mudanças, refazer partes do código é uma atividade inevitável, geralmente com equipes pequenas e prazos de entrega curtos, o desenvolvimento de um planejamento para suprir essas necessidades se torna algo difícil (GAMBA, 2009).

Tanto desenvolvedores como usuários deparam-se com diversos problemas na elaboração, execução e manutenção de projetos, e acabam obtendo prejuízos no decorrer do desenvolvimento. O método Scrum da metodologia ágil visa facilitar a resolução de alguns problemas, como a manutenibilidade de software, utilizando a reengenharia e suas etapas.

A definição de metas, identificação e avaliação de processos existentes e a criação de melhorias visam aumentar a qualidade de um sistema de software já existente, condicionando-o a estar preparado para as modificações e demandas nas funções de negócios na TI (Tecnologia da Informação).

A reengenharia de software examina e altera um software para reconstruí-lo de uma nova forma e um de seus principais objetivos é propor ao usuário um sistema que atenda as suas necessidades, sobretudo a própria necessidade da empresa que não deseja perder o conhecimento sobre o projeto, disponibilidade de serviço e esforço investido.

De acordo com Pressman (2006) software não-manutenível não é um problema novo, a reengenharia de software tem sido motivada para resolução em

manutenção de software durante mais de 40 anos e a manutenção de software existente pode ser responsável por mais de 60% de todo o esforço realizado por uma organização de desenvolvimento.

Neste contexto é que esta pesquisa visa aprofundar o estudo e uso da metodologia ágil e da reengenharia de software em sistemas, não somente a fim de modernizar, mas sim adaptá-los as novas necessidades ao qual estiver inserido.

## 1.1 OBJETIVO GERAL

Contribuir para o estudo e uso da reengenharia de software na manutenção de sistemas por meio da metodologia ágil, utilizando o método Scrum.

## 1.2 OBJETIVOS ESPECÍFICOS

Os objetivos do projeto são baseados nos seguintes itens:

a) Pesquisar sobre os princípios da engenharia de software;
b) Estudar os métodos e as implantações da metodologia ágil em pequenas empresas;
c) Entender e analisar a forma de utilização do método Scrum;
d) Descrever os conceitos e processos da reengenharia de software;
e) Elaborar métodos de manutenção de softwares, utilizando a metodologia ágil e reengenharia de software.

## 1.3 JUSTIFICATIVA

Com o acelerado ritmo no crescimento de pesquisas e estudos em torno da engenharia de software, são visíveis as modificações existentes na manutenção dos sistemas. No estudo da reengenharia, considera-se que um sistema já existente deve ser analisado e modificado para que acompanhe esse ritmo. As metodologias ágeis de desenvolvimento permitem responder rapidamente as mudanças, reduzindo o impacto das mudanças no projeto.

De acordo com Davis (1994) a manutenção começa tão logo o sistema entre em uso. Um sistema pode apresentar defeitos que devem passar por testes e novos sistemas podem ser desenvolvidos exigindo interface com este.

Na manutenção de software, a metodologia ágil pode ser um caminho que levará a um ciclo de trabalho mais inteligente. Isso irá modificar a maneira da TI suportar e melhorar seu desempenho no desenvolvimento e manutenção de software.

A implantação e utilização do método Scrum visa buscar melhorias na manutenção do software focada no trabalho em equipe e na participação ativa do cliente. A manutenção contém novos requisitos e funções, realiza alterações para manter a adequação ao sistema e corrige erros ou mau desempenho do software.

Para Schwaber (2004) o método Scrum é utilizado em trabalhos onde não se pode prever com clareza o que irá ocorrer durante o desenvolvimento do projeto. Sendo assim, o Scrum oferece um conjunto de práticas que possibilita uma melhor visibilidade de todo o software, permitindo que os membros do Scrum saibam exatamente o que está acontecendo e quais os ajustes necessários para manter o projeto de acordo com as metas almejadas.

O método Scrum utiliza equipe de desenvolvimento pequena com revisões freqüentes e colaborações na manutenção do sistema, além disso, possui fases de planejamento e cronograma flexível.

Na percepção de Sommerville (2003) a reengenharia de software tem vantagens em relação a outras abordagens para a evolução do sistema, como a redução de riscos que se tem na criação de um novo software e nos custos reduzidos com o reuso do mesmo.

Nesse aspecto é que a reengenharia de software sendo aplicada por meio da metodologia ágil no método Scrum pode melhorar a qualidade de um sistema sem perder informações e investimentos. A reengenharia reutiliza os modelos e conhecimentos do software original e migra para uma plataforma moderna, além de melhorar a base e a estrutura do software dando sobrevida ao mesmo, reutilizando códigos.

# 2 ENGENHARIA DE SOFTWARE E METODOLOGIAS DE DESENVOLVIMENTO

Para Sommerville (2003) a engenharia de software é uma disciplina da engenharia que se ocupa de todos os aspectos da população de software, desde os estágios iniciais de especificação do sistema ate a manutenção desse sistema, depois que ele entrou em operação.

De acordo com Davis (1994) tanto usuários quanto programadores se concentram em um mesmo programa. Enquanto o programador vê um trabalho a ser realizado, o usuário vê um problema a ser resolvido.

Para Pressman (2006) com o passar do tempo o software não de desgasta, mas sim, se deteriora. Para isso, existem as metodologias de desenvolvimentos. São as metodologias que definem uma sequência de passos a seguir e o conjunto de modelos a utilizar para alcançar a implementação adequada do sistema desejado.

## 2.1 ESTUDO DA ENGENHARIA DE SOFTWARE

De acordo com Pressman (2006) a engenharia de software é uma tecnologia em camadas e qualquer abordagem de engenharia deve se apoiar num compromisso organizacional com qualidade.

As camadas da engenharia de software podem ser divididas de acordo com a figura 1:

Figura 1 - Camadas da Engenharia de Software


Fonte: Pressman (2006, p.17).

Ainda de acordo com Pressman (2006) o alicerce da engenharia de

software é a camada de processo. O processo de engenharia de software mantém unidas as camadas de tecnologia e permite o desenvolvimento de softwares de computador.

O processo de software pode ser definido como um grupo de padrões que definem um conjunto de atividades e ações necessárias para o desenvolvimento de softwares de computador.

Para Pressman (2006) um padrão pode descrever um processo completo de software. Ele também pode ser usado para descrever uma atividade de planejamento ou uma estimativa desse planejamento.

Ao estudar engenharia de software, deve-se entender o termo crise de software, utilizado para expressar as dificuldades encontradas para desenvolver softwares frente ao rápido crescimento da demanda e da complexidade dos problemas a serem resolvidos.

## 2.2 CAUSAS E EFEITOS DA CRISE DE SOFTWARE

O crescimento na construção de softwares é evidente em todo o mundo. Com esse crescimento, surgem as dificuldades que confundem programadores e usuários.

A habilidade em construir softwares deixa a desejar em relação ao potencial de hardwares já desenvolvidos, além disso, a construção de software não é rápida o suficiente para atender as necessidades do mercado.

Atualmente a sociedade depende cada vez mais de software confiável, pois quando ele falha podem ocorrer gastos enormes e desgaste de muitos profissionais para arrumá-lo.

De acordo com Bassi (2008) algumas empresas ainda partem do inicio do projeto para traçar as análises e estimativas do mesmo. Utiliza-se modelo de cascata, desconsiderando a possibilidade de rever as estimativas durante a implementação.

A engenharia de software progrediu muito desde que a crise de software foi identificada, utilizando novas metodologias que auxiliam tanto na qualidade, quanto na agilidade no planejamento e na execução de projetos.

## 2.3 METODOLOGIAS DE DESENVOLVIMENTO

Segundo Gamba (2009), para os grandes sistemas, existe a necessidade das atividades de desenvolvimento dos softwares serem mais eficientes, disciplinadas e previsíveis. Com isso surgiram as metodologias de desenvolvimento de software.

Dentre essas metodologias, a mais utilizada é a metodologia clássica, orientada a planejamento. As metodologias clássicas dominaram a forma de desenvolvimento de softwares até o início da década de 90. Entretanto, a utilização dessas metodologias é aplicada em situações em que os requisitos do sistema são estáveis e os requisitos futuros são previsíveis.

Outra metodologia que vem sendo muito utilizada em diversos setores da engenharia de software é a metodologia ágil, que difere em diversos fatores da metodologia clássica.

Os sistemas de softwares devem se aprimorar cada vez mais, gerando maiores facilidades, tanto para programadores quanto para usuários. A engenharia de software concede métodos que podem contribuir para tal afirmação.

De acordo com Sommerville (2003) um método de engenharia de software é uma abordagem estruturada para o desenvolvimento de software, cujo objetivo é facilitar a produção de software de alta qualidade, apresentando uma boa relação custo-benefício.

Conforme Borgonovo e Silva (2007) na engenharia de software existem vários processos definidos, entre eles, a metodologia ágil, que procura ter o mínimo possível de documentação e tende a contribuir para otimização dos esforços dos recursos humanos como financeiros de uma organização.

Deste modo é que a metodologia ágil procura agir na elaboração de softwares e na manutenção dos mesmos, diminuindo a documentação e contribuindo para a construção de softwares com mais qualidade.

# 3 METODOLOGIA ÁGIL

O desenvolvimento através da metodologia ágil, como o próprio nome já sugere, procura tornar o projeto do software cada vez mais produtivo, tanto no desenvolvimento quanto na manutenção do mesmo, diminuindo a documentação e criando uma iteração entre usuários e programadores.

Para Highsmith (*Agile Project Management*) Agilidade é a habilidade de criar e responder a mudanças com respeito ao resultado financeiro do projeto em um turbulento ambiente de negócios. Agilidade é a habilidade de balancear flexibilidade com estabilidade.

## 3.1 CONCEITOS

Dentre os conceitos da agilidade, pode-se destacar maior iteração do que processos e ferramentas entre indivíduos, software em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos e responder a mudanças mais que seguir um plano.

Segundo Abrahamsson (2002), uma metodologia pode ser dita ágil quando efetua o desenvolvimento de software de forma incremental (liberação de pequenas versões, em iterações de curta duração), colaborativa (cliente e desenvolvedores trabalhando juntos, em constante comunicação), direta (o método em si é simples de aprender e modificar) e adaptativa (capaz de responder às mudanças até o último instante).

De acordo com Pressman (2006), há algumas questões a serem abordadas sobre a agilidade, como métodos para alcançá-la e como construir softwares que satisfaçam as necessidades do cliente a curto e longo prazo.

Uma equipe ágil deve manter-se atualizada sobre modificações na elaboração do projeto desde a fase inicial até a fase de implantação. O método ágil propõe testes em todas as fases da elaboração para que a quantidade de erros seja mínima.

A idéia de Bassi (2008) o uso da metodologia ágil pode trazer ganhos em qualidade e produtividade em TI, porem não é tão simples chegar até eles. O uso da metodologia ágil exige uma mudança de paradigmas, portanto, tanto empresa como

a equipe devem estar dispostos a fazer.

## 3.2 PRINCÍPIOS DA METODOLOGIA ÁGIL

Em 2001 foi assinado pelos principais profissionais veteranos na área de desenvolvimento de softwares o manifesto ágil.

De seus objetivos, deve-se destacar a possibilidade de discutir uma nova forma para melhorar a velocidade no desenvolvimento de seus sistemas baseados em suas experiências de anos programando. Embora muitos desenvolvedores sejam contra esta metodologia, pois pode causar confusão dependendo do tamanho da aplicação, muitos já são adeptos dos principais conceitos do desenvolvimento ágil.

De acordo com os idealizadores do projeto, o manifesto ágil possui 12 princípios:

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

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

As idéias relativas ao movimento ágil têm sido rapidamente disseminadas pela comunidade de desenvolvimento. Todavia, mesmo que os desenvolvedores avaliem, de forma favorável, técnicas como o desenvolvimento incremental e a programação orientada a testes, sugerindo a adoção de uma metodologia ágil, essa decisão ainda tem que ser tomada pela organização na qual estão inseridos. Para isso, se faz necessária uma argumentação quantitativa.

Para Bassi (2008) o modelo ágil não trata apenas das questões técnicas, mas também do relacionamento entre as pessoas. Para desenvolver software com agilidade devem-se considerar também os fatores humanos, para que se chegue mais rapidamente nas soluções adequadas aos problemas do cliente.

Ainda de acordo com Bassi (2008) quando se fala em metodologia ágil, devemos considerar que o software é desenvolvido por pessoas e não por processos. E os modelos ágeis consideram que mudanças poderão acontecer e não tenta prever ou controlar as mesmas.

## 3.3 USABILIDADE E PLANEJAMENTO DA METODOLOGIA ÁGIL

A maior parte dos projetos de desenvolvimento de software pode ser descrita simplesmente como “programar e corrigir”, sendo desenvolvidos sem planejamento ou uma fase organizada de design do sistema. Disso usualmente decorre uma grande quantidade de erros, os quais precisam ser resolvidos, em uma longa etapa que sempre estende o prazo inicialmente proposto. O movimento original de melhoria no setor foi o que introduziu a noção de metodologia, ou seja, uma abordagem disciplinada para o desenvolvimento de software com o objetivo de tornar o processo mais previsível e eficiente (FOWLER, 2005).

A escolha da metodologia mais adequada para o desenvolvimento de software em uma organização não é uma tarefa trivial. As metodologias ágeis têm despertado o interesse do mercado, apresentando evidências de melhoria na

produtividade, mas, para que possam ser efetivamente usadas em larga escala, precisam provar alguns de seus pontos de vista.

A agilidade se torna importante na elaboração do projeto, pois tende a evitar erros e conflitos no sistema, colaborando para que a manutenção do software possa ser realizada de forma mais rápida e competente. Além disso, um sistema deve estar sempre pronto para incrementos de informações e reutilizações.

Conforme Pressman (2006), os membros de uma equipe de software devem ter competência nas suas habilidades especificas para elaborar um software. A equipe deve estar focada em uma mesma meta, mesmo cada membro com uma tarefa diferente, e a colaboração entre si é indispensável. Membros de equipe de software com capacidade de tomada de decisões e habilidade em resolver problemas vagos geram respeito e confiança mutua entre o grupo. A auto-organização também é um ponto chave na qualidade e na agilidade do desenvolvimento, pois serve para aperfeiçoar a colaboração e aumentar o moral da equipe.

Assim como outros métodos, a metodologia ágil, especialmente no método Scrum, deve seguir alguns princípios para que possa cumprir rigorosamente seu processo de agilidade, tanto no desenvolvimento quanto na manutenção de softwares.

Uma questão muito importante, a qual será sempre levantada por qualquer organização que estude a adoção de uma nova metodologia, é a quantidade de empresas que já a adotam com sucesso. Infelizmente, na comunidade de desenvolvimento de software, não existe praticamente nada determinado estatisticamente. É necessário valer-se de alguns poucos estudos, conduzidos informalmente, para verificar as tendências do mercado.

No uso da metodologia ágil, cada tarefa definida deve ser verificada quanto ao seu percentual de andamento e também quanto à sua qualidade através de testes e entregas que são feitas com maior freqüência, ocasionando assim a iteração entre usuário e programador.

# 4 MÉTODO SCRUM DA METODOLOGIA ÁGIL

O método Scrum da metodologia ágil é um método que gerencia projetos ou atividades complexas com base no processo iterativo e incremental (ZANATTA, 2004).

Segundo Schwaber (2004) o método Scrum é utilizado em grandes projetos, onde é praticamente impossível prever o que ira ocorrer durante o desenvolvimento do mesmo. Com isso, o método Scrum oferece um conjunto de práticas que mantém tudo visível, colaborando para que os membros do Scrum saibam exatamente o que está acontecendo e quais são os ajustem necessários para manter o projeto em direção às metas desejadas.

O método Scrum tem como idéia construir o todo aos poucos, refinando os requisitos, equalizando o entendimento sobre eles e corrigindo o rumo sempre que uma mudança provoque alterações significativas.

## 4.1 PRINCÍPIOS DO MÉTODO SCRUM

De acordo com Pressman (2006) na implantação de sistemas através da metodologia ágil no método Scrum, pequenas equipes de trabalho são organizadas gerando comunicação e compartilhamento de informações sobre o software, e o processo precisa ser adaptável a modificações.

Ainda na visão de Pressman (2006) o processo também possui diferentes incrementos de software e é dividido em pequenos pacotes, facilitando assim os testes e documentações que serão realizados à medida que o produto é construído.

Segundo Zapata (2004) ao invés de definir técnicas e ferramentas de desenvolvimento, o método Scrum define como as equipes irão se desenvolver trabalhando em ambientes com constantes alterações e surgimento de novos requisitos de software.

Para Schwaber (2004) o método Scrum incentiva cada membro a participar, mostrando suas idéias, críticas e sugestões para chegar ao seu objetivo de forma mais satisfatória.

Segundo Abrahamsson (2002) o método Scrum envolve freqüente gerência das atividades com o objetivo de identificar consistentemente algum

impedimento ou deficiência no método de desenvolvimento, bem como as práticas que estão sendo utilizadas.

## 4.2 CARACTERÍSTICAS E ATRIBUIÇÕES DO MÉTODO SCRUM

De acordo com Fonseca (2008) o Scrum é um *framework* de processo ágil utilizado para gerenciar e controlar o desenvolvimento de um produto de software através de práticas iterativas e incrementais. É composto por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos.

O Scrum mantém uma lista de funcionalidades que deverão ser implementadas, essa lista é chamada de *Product Backlog*. Para cada iteração ou *Sprint*, é feita uma reunião inicial de planejamento, chamada de *Sprint Planning Meeting*, onde itens desta lista são priorizados pelo cliente, chamado de *Product Owner.* A equipe define quais funcionalidades poderão ser atendidas dentro da iteração de acordo com a capacidade da mesma. Essa lista planejada para a iteração é chamada de *Sprint Backlog*.

Para Fonseca (2008) os principais papéis do Scrum são: *Product Owner, Scrum Master* e *Scrum Team* (equipe do projeto). Não há como fazer um mapeamento direto entre os papéis do Scrum e os papéis convencionais conhecidos. A figura única do Gerente de Projetos passa a não existir e suas responsabilidades estão diluídas entre os papéis citados.

A idéia de Fonseca (2008) cada membro da equipe conhece sua participação frente ao projeto e trabalha em conjunto para conseguir alcançar o objetivo (*goal*) definido.

### 4.2.1 *Product Backlog*

O *Product Backlog* é uma lista contendo todas as funcionalidades desejadas para um produto. Esta lista não precisa estar completa no início de um projeto. Pode-se começar com tudo aquilo que é mais importante em um primeiro momento. Com o tempo, o *Product Backlog* cresce e muda à medida que se aprende mais sobre o produto e seus usuários (PRESSMAN, 2006).

### 4.2.2 *Product Owner*

Para Gamba (2009) o *Product Owner* tem como objetivo definir o conteúdo do *Product Backlog* criado, e também, por rever e aceitar as entregas ao final de cada *Sprint*.

### 4.2.3 *Scrum Master*

O *Scrum master* tem atribuições essenciais para o sucesso de um projeto ágil. Além de Remover impedimentos e garantir o uso do Scrum, outra função é proteger o time de interferências externas.

### 4.2.4 *Scrum Team*

O *Scrum Team* é a equipe de desenvolvimento. Essa equipe deve trabalhar em grupo, sem divisão de papéis. Todos no projeto trabalham juntos para concluir o conjunto de trabalho com o qual se comprometeram conjuntamente para um *Sprint*.

Uma equipe de projetos Scrum é geralmente pequena, entre 6 e 10 membros.

## 4.3 ADAPTAÇÕES AO MÉTODO SCRUM

Como visto anteriormente, a metodologia ágil pode ser representada por vários métodos, e para utilizá-los o projeto de software tradicional deve adaptar-se ao método, de acordo com seus princípios e sua usabilidade.

Para Alves (2008) o método Scrum é composto por um conjunto de boas praticas de gestão que admite ajustes rápidos, acompanhamento constante e planos realísticos.

O processo de desenvolvimento de software é uma atividade trabalhosa em sua maior parte, onde as expressões “codificar e consertar” são constantemente aplicadas. Diversas vezes o software é desenvolvido com decisões em curto prazo e o projeto do sistema não possui planos definidos, causando defeitos que se tornam

constantes e de difícil detecção.

O método Scrum da metodologia ágil adaptado ao projeto do sistema impõe um processo disciplinado no desenvolvimento de software, com o objetivo de torná-lo mais previsível e mais eficiente.

Assim como outros métodos ágeis, o método Scrum é adaptativo ao invés de predeterminantes, os métodos tradicionais de metodologia tendem a planejar grande parte do projeto por um longo período de tempo, porém, esse planejamento pode não estar preparado para mudanças. O método ágil Scrum tenta se adaptar as mudanças de tal forma que as mesmas fortaleçam o desenvolvimento, fazendo com que o cronograma seja cumprido e a qualidade do software cresça.

O método Scrum é orientado a pessoas ao invés de ser orientado a processos. A metodologia tradicional tem como um de seus objetivos definir um processo que funcione bem, independente de quem ira utilizá-lo, já o método Scrum propõe que o processo de desenvolvimento de suporte a equipe de desenvolvimento e seu trabalho.

A tarefa de executar um projeto adaptativo não é tão simples. Exige uma equipe eficaz de desenvolvedores. A equipe deve ser efetiva na qualidade e na iteração entre seus desenvolvedores.

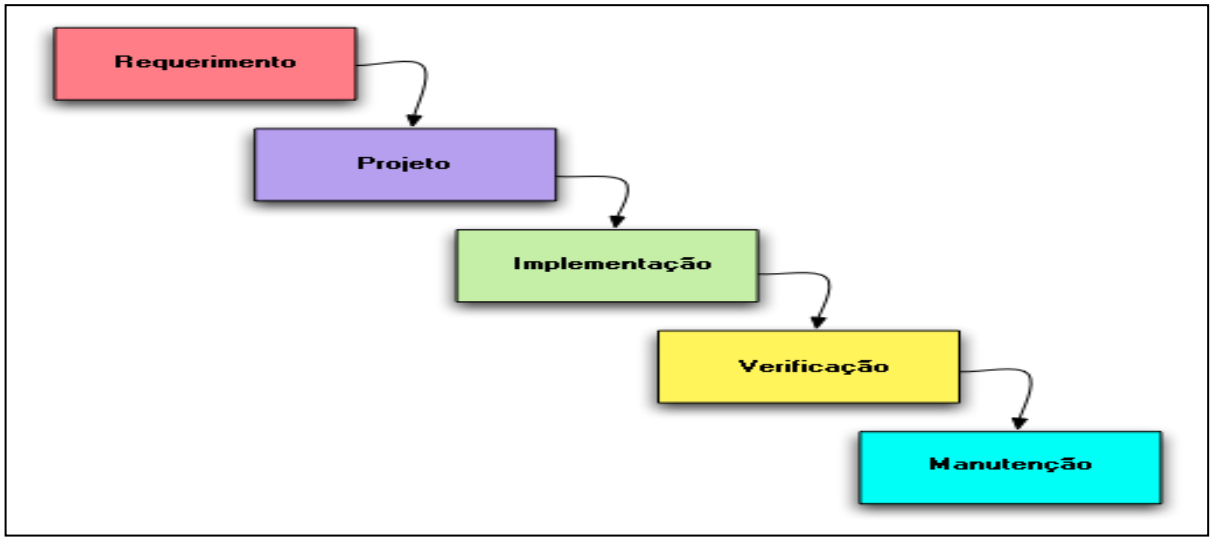
# 5 CICLO DE VIDA DE PROJETOS

O ciclo de vida de um projeto é a evolução que o mesmo passa desde a sua elaboração até sua manutenibilidade, passando por diversas fases, tendo-as que cumprirem rigorosamente, para de que o software apresente o menor número de erros possíveis.

## 5.1 CICLO DE VIDA CLÁSSICO

De acordo com Pressman (1995), o ciclo de vida clássico de um projeto de engenharia de software, também chamado de modelo cascata é representado em estágios, representado pela figura 2:

Figura 2 - Ciclo de Vida de Software no Modelo Cascata




Fonte: http://www.nerdinside.net/book/export/html/261

Pressman (1995) ressalta que, o ciclo de vida clássico é o mais utilizado na engenharia de software, entretanto, as criticas a esse paradigma fazem com que até mesmo seus defensores questionem sua aplicabilidade.

## 5.2 CICLO DE VIDA ÁGIL

A metodologia ágil propõe alterações no ciclo de vida do software,

visando à agilidade na documentação e aproximação entre programadores e clientes, procurando criar um melhor planejamento das ações na construção do software.

No ciclo de vida em projetos ágeis, deve-se considerar que as mudanças poderão ocorrer ao invés de tentar prever e controlar todas as variáveis, o que economiza tempo e energia tentando fazer um plano de projeto perfeito.

Bassi (2008) apresenta e descreve o papel e os objetivos de cada nível de planejamento, de acordo com a figura 3:

Figura 3 - Ciclo de Vida do Software – Modelo Ágil


Fonte: Revista Engenharia de Software, v. 8, p11.

Segundo Bassi (2008) o planejamento estratégico fica fora do escopo de desenvolvimento e o planejamento de produto define as atribuições da solução, avalia a viabilidade técnica e define a equipe. O planejamento do release foca na definição de uma versão que possa ser entregue ao usuário, já o planejamento de iteração cria um subgrupo de funcionalidades do release para ser implementado. Finalizando, no processo diário acontece uma reunião entre a equipe para observar seus objetivos alcançados e seus planos futuros.

Bassi (2008) afirma ainda que a separação em níveis e a flexibilidade são benéficas para o software a ser implementado, contudo, o planejamento em níveis requer experiência para obter resultados significativos.

### 5.2.1 Planejamento Ágil e Qualidade do Software

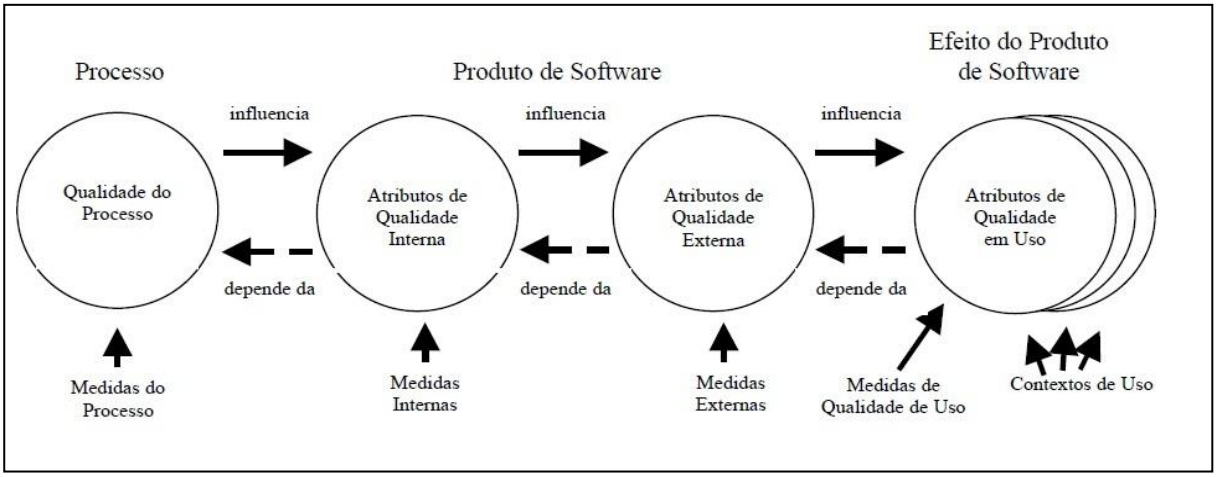
De acordo com Pressman (2006) a qualidade de um software deve ser aprimorada desde o inicio do projeto, apesar de que muitos dos programadores passam a se preocupar com essa questão somente depois que o código foi gerado.

Para Bassi (2008) uma alternativa para começar o projeto ágil é diminuir o tamanho do ciclo, pois ciclos completos de algumas semanas podem ser revistos com maior frequência.

Software de qualidade é fácil de usar, funciona corretamente, é de fácil manutenção e mantém a integridade dos dados em falhas do ambiente ou outras fora de controle.

A avaliação do produto de software é um dos processos no ciclo de vida de desenvolvimento de software. O objetivo é que o produto tenha o efeito desejado em um contexto particular de uso, conforme a figura 4:

Figura 4 - Qualidade de Software




Fonte: ROCHA (2001, pg. 13)

A idéia de Sommerville (2003) é de que especialmente em pequenos projetos, onde a equipe de desenvolvimento é pequena, a qualidade da própria equipe é mais importante que o processo de desempenho utilizado. Se a equipe possuir alto nível de conhecimento provavelmente a qualidade do processo também será alta.

A qualidade do processo de desenvolvimento vem a contribuir para a

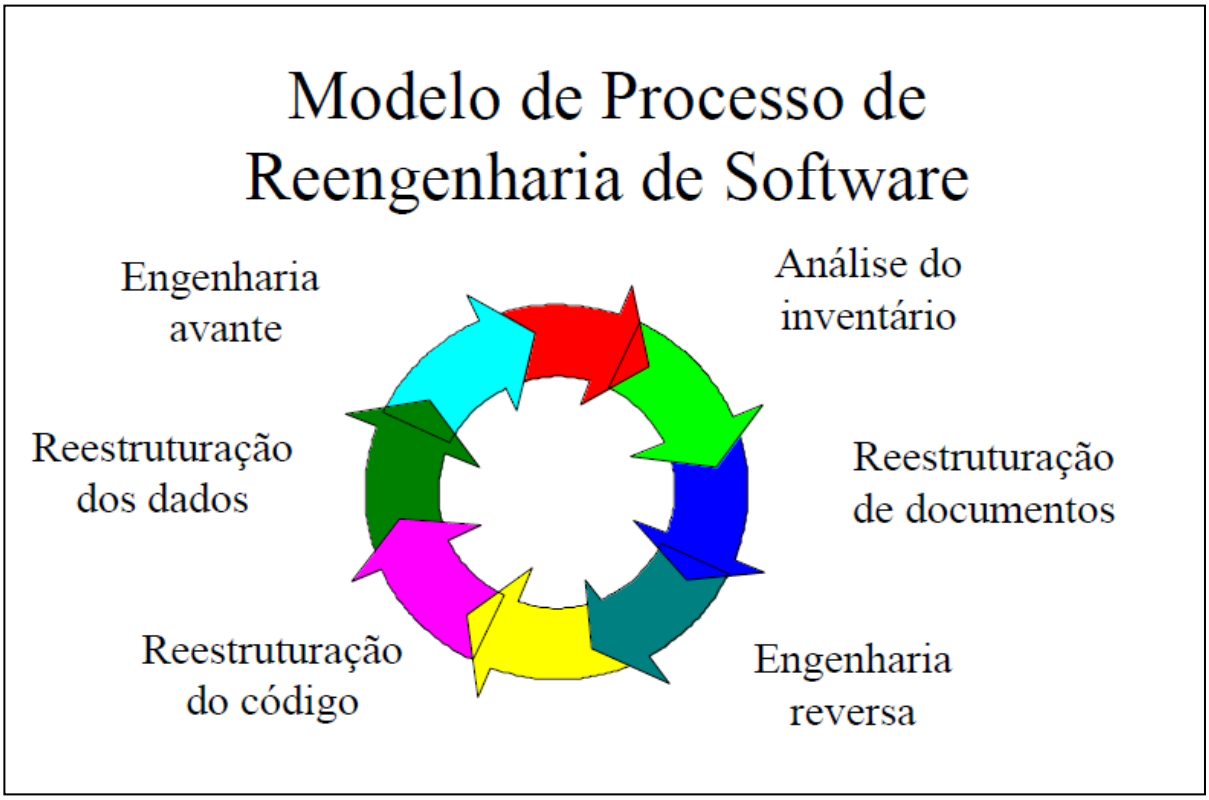
melhoria da qualidade de uso. A avaliação e melhoria de um desenvolvimento afetam diretamente na qualidade e é também um meio de melhorar a mesma.

## 6 REENGENHARIA DE SOFTWARE

Segundo Serra (2009) a reengenharia de software pode ser definida como uma aprimoração de software, ou seja, uma abordagem disciplinada para migrar softwares legados em softwares evolutivos. O processo de reengenharia de software aplica os princípios da engenharia de software em um software legado para atender requisitos existentes e novos requisitos.

O modelo de processo de reengenharia de software pode ser definido conforme a figura 5:

Figura 5 - Modelo de Processo de Reengenharia de Software.




Fonte: www.linhadecodigo.com.br

A reengenharia de software utiliza o termo “software legado” para iniciar seu funcionamento.

## 6.1 SOFTWARE LEGADO

Ao citar a reengenharia de software, é interessante ressaltar a existência do software legado, que é o software implantado que sofrerá modificações.

Para Serra (2009) a atribuição de software legado a um software antigo e ultrapassado nem sempre é verdadeira, pois o software legado é qualquer software que pode sofrer melhorias ou correções.

## 6.2 ATIVIDADES DA REENGENHARIA DE SOFTWARE

De acordo com Serra (2009) deve-se constatar a necessidade de alterações do software antes mesmo da reengenharia de software ser iniciada. Um desafio a ser enfrentado é determinar as atividades utilizadas em pontos do ciclo de vida. Com isso, podem-se destacar quatro atividades básicas para a reengenharia de software:

a) Reestruturação de documentos – análise na documentação do software legado;
b) Tradução do código fonte – pode-se traduzir o código de uma linguagem para outra, dependendo da utilizada no software legado.
c) Reengenharia reversa – deriva o projeto de software a partir de seu código fonte;
d) Reestruturação do código e dados – modifica os módulos de software, não alterando o comportamento externo do código, melhorando a estrutura interna.

As atividades da reengenharia de software garantem que a mesma seja aplicada da melhor maneira possível, garantindo a qualidade nas alterações do software legado. O uso da engenharia reversa destaca-se por ser semelhante e até mesmo confundido com a reengenharia de software.

### 6.2.1 Engenharia Reversa

A engenharia reversa consiste em um processo de exame e compreensão de um software existente, para recriar o projeto e decifrar os requisitos

implementados pelo sistema, apresentando-os em um nível ou grau mais alto de abstração.

Por meio da engenharia reversa um software pode ser visualizado em diferentes níveis de abstração. Cada visualização abstrai características próprias da fase do ciclo de vida correspondente à abstração.

A engenharia reversa também é considerada uma ferramenta eficaz para a garantia da segurança de sistemas de alta criticidade, pois por meio de suas técnicas pode explorar o código dos programas a fim de averiguar se há códigos maliciosos, averiguar se o código original de um software não sofreu alterações como injeção de rotinas ocultas ou se há códigos de desenvolvimento esquecidos pelos idealizadores.

As técnicas de engenharia reversa podem ainda ser de grande utilidade na recuperação de desastres com dados. Por exemplo, um sistema de arquivos criptografado pode ser compreendido e sua decodificação ser fundamental para a recuperação de dados perdidos.

Em alguns países a prática de engenharia reversa é considerada ilegal e alguns praticantes interessam-se em obter acesso não autorizado a recursos de software não oferecidos gratuitamente pelos detentores dos direitos comerciais do produto. Tal prática é denominada *Cracking.*

### 6.2.2 Engenharia Reversa X Manutenção

As atividades de manutenção de software podem motivar muitas ferramentas de engenharia reversa. Essas motivações são provenientes da elevada proporção entre tempo e custo no entendimento e exame do software a ser mantido.

Em geral, grande parte do tempo da manutenção do software é gasta na tentativa de entender o software.

Nas manutenções adaptativas, onde se ajusta o software a um novo ambiente, e também nas manutenções evolutivas, onde se adiciona novas funcionalidades ao software, as técnicas de engenharia reversa são usadas através do fornecimento de visões do software, para localizar componentes onde serão realizadas as mudanças e adições necessárias e para auxiliar no controle da estrutura do sistema modificado, através da produção de documentação.

### 6.2.3 Engenharia Reversa X Reuso

O reuso consiste em uma atividade capaz de identificar um software reutilizável, alem de capacitá-lo e reconfigurá-lo para uma nova aplicação de um sistema.

O processo de reuso é descrito por meio de atividades de reconhecimento, decomposição, classificação, seleção, adaptação e composição. As técnicas de engenharia reversa auxiliam essas atividades.

# 7 MANUTENÇÃO DE SOFTWARE

De acordo com Paduelli (2008) a atividade de manutenção de software é identificada como a modificação de um projeto de software já entregue ao cliente, para a correção de erros, melhora de desempenho ou ainda para adaptação do software a um ambiente modificado.

Ainda para Paduelli (2008) as atividades de manutenção de software são caracterizadas pela intervenção no produto de maneira a evitar sua deterioração. Um software não se desgasta como produtos de hardware, mas assim no sentido de seus objetivos e funcionalidades se adequarem cada vez menos ao ambiente externo.

Para Sommerville (2003) as mudanças que ocorrem em um software para deixá-lo livre de erros podem ser definidas como atividades de evolução de software.

A idéia de investir na evolução do software ou abandoná-lo para iniciar um novo projeto não é uma tarefa fácil. O custo envolvido e a confiabilidade do software após a manutenção são fatores importantes que devem ser considerados.

De acordo com Pressman (2006) um dos problemas da manutenção de software é a mobilidade do pessoal de software, pois a equipe que fez o trabalho original talvez não esteja por perto no momento da execução da manutenção. Modificação é inevitável quando sistemas baseados em computador são construídos, por conseqüência, precisamos desenvolver mecanismos para avaliar, controlar e realizar modificações.

Pressman (2006) define a manutenção de softwares em quatro atividades fundamentais: correção de erros, adaptação, aperfeiçoamento e reengenharia.

A manutenção de software e a inteligência dos programas são conceitos paralelos, pois quanto mais difícil é entender um programa, mais difícil é mantê-lo.

## 7.1 HISTÓRICO DOS PROBLEMAS DE MANUTENÇÃO DE SOFTWARE

Assim como em qualquer área, a manutenção é vista como o aprimoramento ou melhoramento do produto. No desenvolvimento de software não poderia ser diferente. Porem, para muitos, a manutenção de software é vista com um problema a ser enfrentado, a cada projeto desenvolvido, mais problemas de

manutenção de software podem surgir.

Yourdon (1992), considerado um dos nomes mais importantes da análise estruturada clássica, estimou, ainda no início dos anos 1990, que a manutenção de software viria a ser um dos problemas relevantes no futuro. Essa previsão foi feita com base em constatações da época, como a política de entregar em tempo ao usuário um sistema que funcionasse, não se preocupando com questões de manutenibilidade. Mais do que isso, Yourdon afirmou que a postura da alta direção das organizações que dependiam de software seria a de atacar o problema apenas quando ele emergisse de fato.

Antes, porém, do que afirmou Yourdon, ainda na década de 1980, estudos referentes a problemas com manutenção de software já ocorriam, e um pioneiro e relevante trabalho foi realizado por Lientz e Swanson (1980), publicado na forma de um livro. Esse trabalho descreve os resultados obtidos com estudos no intuito de averiguar a forma como as organizações tratavam a questão de manutenção de software na época. A pesquisa contou com duas fases. Na primeira, realizada com 69 organizações, foram levantadas as características da atividade de manutenção de software por elas realizada. Essa fase inicial obteve as seguintes conclusões:

A manutenção de software existente consome uma média de 48% das horas anuais do pessoal de sistemas e programação; Aproximadamente 60% dos esforços de manutenção são dedicados a manutenções do tipo perfectivas. Dentre esse percentual, dois terços se referem a esforços de melhoria de software;

Problemas de natureza gerencial em manutenção foram vistos como mais importantes do que aqueles de natureza técnica. Com base nessas verificações, uma segunda fase foi conduzida na pesquisa, nesse caso envolvendo 487 organizações, o que abrangia instituições governamentais, bancos, transportes, mineração, educação e indústrias de manufatura em geral, localizadas nos Estados Unidos e Canadá. O intuito dessa segunda fase foi o de validar os resultados obtidos na primeira, buscando uma compreensão mais profunda e abrangente.

## 7.2 O USO DA MANUTENÇÃO DE SOFTWARES

A manutenção de softwares possui alguns modelos já definidos, os quais diferem em seu uso: manutenção corretiva, manutenção adaptativa, manutenção

perfectiva, manutenção preventiva.

A manutenção corretiva, como o próprio nome propõe, tem por objetivo corrigir ou restaurar a produtividade de um software. A manutenção corretiva é a primeira ação a ser utilizada em um software, pois é utilizada logo após a falha do mesmo. Espera-se em uma manutenção corretiva agilidade e qualidade para a correção ser feita de forma eficiente.

O uso da manutenção adaptativa visa adaptar o software a um novo ambiente, normalmente imposto ou pelo usuário do software ou por motivos externos. A manutenção adaptativa não é usada para corrigir defeitos de um software, mas sim para adequá-lo a mudanças.

Com o surgimento de novas funcionalidades, o software pode adquirir novos problemas de manutenção, e o desenvolvedor pode não prever o que essas novas funcionalidades podem trazer de problemas para o software. A manutenção perfectiva pode auxiliar na detecção de falhas nas novas atividades adicionadas ao sistema.

A manutenção preventiva facilita a compreensão do sistema e aprimora sua estrutura interna, colaborando para que a manutenção de software possa ser realizada da melhor e mais rápida forma possível. O uso da prevenção pode diminuir defeitos de software e facilitar muito o processo de reengenharia do software.

Durante a execução de manutenção de um software qualquer, diferentes partes precisam interagir de forma que os objetivos sejam compreendidos e os resultados esperados alcançados. Essas partes são o cliente (organização), o mantenedor e finalmente o usuário, conforme mostra a figura 6:

Figura 6 - Partes da Manutenção de Software


Fonte: Revista Engenharia de Software pág. 38.

De acordo com a figura 6, essas partes devem ter um relacionamento organizado para que haja uma melhor comunicação entre os mesmo.

# 8 TRABALHOS CORRELATOS

No período em que foram feitos os estudos para a realização deste trabalho de pesquisa, foram analisados alguns trabalhos que possuíam idéias relacionadas a esta pesquisa, porem, com focos diferenciados. Alguns trabalhos envolvendo a metodologia ágil e o método Scrum pode ser visto a seguir.

## 8.1 IMPLANTANDO O SCRUM EM UM AMBIENTE DE DESENVOLVIMENTO DE PRODUTOS PARA INTERNET

O presente trabalho de pesquisa foi elaborado por Jacques Douglas Varaschim, na Universidade Católica do Rio de Janeiro em fevereiro de 2007.

Essa dissertação tem como objetivo auxiliar as empresas de internet a disponibilizar seus produtos aos usuários de forma rápida e eficiente utilizando a metodologia ágil como principal fonte. Para isso, o método Scrum requer mudança em gestão, arquitetura, testes e garantia de qualidade e esses são os desafios proposto nessa pesquisa.

## 8.2 UMA PROPOSTA DE MELHORIA NO PROCESSO DE ESTIMATIVA DE TAMANHO DE SOFTWARE PARA PROJETOS GERENCIADOS POR SCRUM

Monografia apresentada junto ao curso de Ciência da Computação da Universidade Federal de Pernambuco, na área de gerenciamento de projeto de software, elaborada por Flávio Almeida Araújo Sobrinho em dezembro de 2009.

Os métodos desenvolvidos para estimar tamanho são os mais diversos, indo desde simples analogias entre projetos, até estudos sistemáticos utilizando abordagens matemáticas. O objetivo deste trabalho é propor uma melhoria no processo de estimativa de tamanho de software para projetos que seguem a metodologia Scrum de gerenciamento de projetos.

## 8.3 METODOLOGIAS ÁGEIS: UM NOVO PARADIGMA DE DESENVOLVIMENTO DE SOFTWARE

Artigo elaborado por Renata Bastos Ferreira e Francisco de Paula Antunes Lima para apresentação no II *Workshop* Um Olhar Sociotécnico sobre a Engenharia de Software – *WOSES*.

O artigo trata da inovação e das melhorias atingidas com o uso de métodos ágeis na engenharia de software. Considera-se no presente artigo, o forte crescimento da metodologia ágil em empresas de software, onde a flexibilidade e a qualidade são palavras chaves para o sucesso.

# 9 MÉTODO SCRUM E REENGENHARIA APLICADOS NA MANUTENÇÃO DE SOFTWARES

De acordo com o que já foi visto anteriormente, o custo da manutenção de softwares vem crescendo de forma acelerada. O uso de Scrum na manutenção de software através da reengenharia visa suprir essas necessidades, contribuindo também para a diminuição de custos e uma maior agilidade na resolução dos problemas de manutenção.

Um dos principais problemas da manutenção de software tradicionalmente utilizada são o alto custo e a mobilidade do pessoal de software, muitas vezes os mesmos programadores que fizeram o projeto não estarão presentes na sua manutenção, dificultando e aumentando o custo da mesma.

O método Scrum utilizado na manutenção de softwares pode auxiliar com agilidade e menor custo na manutenção de sistemas já existentes, utilizando a reengenharia de software como ferramenta na modificação e melhoramento dos sistemas.

Pressman (2006) afirma que a manutenção de software tradicional possui quatro atividades fundamentais: correção de erros, adaptação, aperfeiçoamento e reengenharia.

Assim como no desenvolvimento de softwares, a identificação e o gerenciamento dos riscos no método Scrum são feitos em reuniões diárias durante toda a iteração, tendo assim o controle da qualidade dos trabalhos e sua avaliação. Esta tarefa pode ser executada através de testes, revisão por pares, inspeção contínua e acompanhamento pelo cliente, que seguem o mesmo processo.

Para Isotton Neto (2004) o foco das metodologias tradicionais são os grandes projetos, que necessitam de especificações detalhadas do mesmo, porém estes se tornam lentos para mudanças, já as metodologias ágeis têm seu foco nos pequenos projetos, totalmente adaptável a mudanças, entretanto fraco na parte de documentos.

Os principais papéis do Scrum devem ser aplicados na manutenção de softwares: *Product Backlog, Product Owner, Scrum Master* e *Scrum Team* (equipe do projeto), além de outros papéis que tornam o uso do Scrum cada vez mais

simplificado.

Não há como fazermos um mapeamento direto entre os papéis do Scrum e os papéis convencionais conhecidos. Não existe a figura única do Gerente de Projetos. Suas responsabilidades estão diluídas entre os papéis citados. Cada um conhece sua participação frente ao projeto e trabalha em conjunto para conseguir alcançar o objetivo definido.

Diante disso, propõe-se um estudo sobre o uso dessas práticas na manutenção do software, utilizando a reengenharia para manter um software em funcionamento da melhor forma possível. Alem disso, pretende-se utilizar o método Scrum da metodologia ágil e suas funções para o funcionamento da manutenção.

## 9.1 UTILIZANDO SCRUM E SEUS ARTEFATOS NA MANUTENÇÃO DE SOFTWARES

Como visto no capitulo quatro, o Método Scrum possui algumas funcionalidades que devem ser seguidas para que seu funcionamento seja o mais produtivo possível. Essas mesmas funcionalidades devem ser aplicadas na manutenção dos softwares, visando assim maior agilidade.

### 9.1.1 *Product Backlog*

Na manutenção de softwares o *Product Backlog* tem o mesmo papel no qual é utilizado no desenvolvimento, porém, visando à manutenção de um software já existente e utilizando a reengenharia de software para a resolução de diversos problemas envolvendo o software em questão.

O *Product Backlog* contem as funcionalidades desejadas no inicio do projeto e também as funcionalidades a serem melhoradas através da reengenharia do mesmo. Essa lista não necessita estar completa no inicio do processo de manutenção, podendo sofrer alterações durante as fases de trabalho.

O *Product Backlog* é a relação de funcionalidades que compõem o escopo do projeto. É comum ocorrer o incremento ou alterações no *Product Backlog* durante a execução, pois novas funcionalidades vão sendo agregadas e mudanças de rumo podem ocorrer à medida que se avança na execução dos trabalhos.

Para Cisneiros (2009), podemos exemplificar as funções do *Product Backlog* através dos modelos no quadro 1 (*Product Backlog*) e também no quadro 2 (*Product Backlog* detalhado):

Quadro 1 - *Product Backlog*

| Funcionalidade | Prioridade |
|---|---|
| Modelagem de dados | 1 |
| Cadastro e gerenciamento de usuários | 2 |
| Conversão de vídeo para visualização na Internet (Codec) | 3 |
| Cadastro e gerenciamento de vídeos pelos usuários | 4 |
| Layout (Aparência) da página | 5 |
| Comentários para os vídeos e usuários | 6 |
| Notas para vídeos (*ranking*) | 7 |
| Proteção contra SPAM | 8 |
| Canais (grupos) de vídeos e usuários | 9 |
| Sistema de legendagem de vídeos | 10 |
| Reconhecimento de sons dos vídeos | 11 |

Fonte: http://www.devin.com.br/modelo-scrum/

Ouadro 2 - *Product Backlog* detalhado

| Funcionalidade | Prioridade | Custo-horas |
|---|---|---|
| Modelagem de dados | 1 | 32 |
| Cadastro e gerenciamento de usuários | 2 | 26 |
| Conversão de vídeo para visualização na Internet (Codec) | 3 | 80 |
| Cadastro e gerenciamento de vídeos pelos usuários | 4 | 48 |
| Layout (Aparência) da página | 5 | 28 |
| Comentários para os vídeos e usuários | 6 | 20 |
| Notas para vídeos (ranking) | 7 | 16 |
| Proteção contra SPAM | 8 | 20 |
| Canais (grupos) de vídeos e usuários | 9 | 26 |
| Sistema de legendagem de vídeos | 10 | 64 |
| Reconhecimento de sons dos vídeos | 11 | 92 |

Fonte: http://www.devin.com.br/modelo-scrum/

### 9.1.2 *Product Owner*

O *Product Owner* pode ser considerado a única pessoa responsável pelo

gerenciamento do *Product Backlog* e por garantir o valor do trabalho realizado pela equipe. Essa pessoa mantém em funcionamento o *Product Backlog* e garante que ele esteja visível para todos. Todos os membros da equipe devem saber quais itens têm a maior prioridade para a manutenção e de que forma será o andamento do processo. Quem quiser mudar a prioridade de um item, terá que convencer o *Product Owner* a fazê-lo.

As empresas que adotam Scrum para a manutenção de seus softwares podem perceber que isso influencia seus métodos para definir prioridades e requisitos ao longo do tempo.

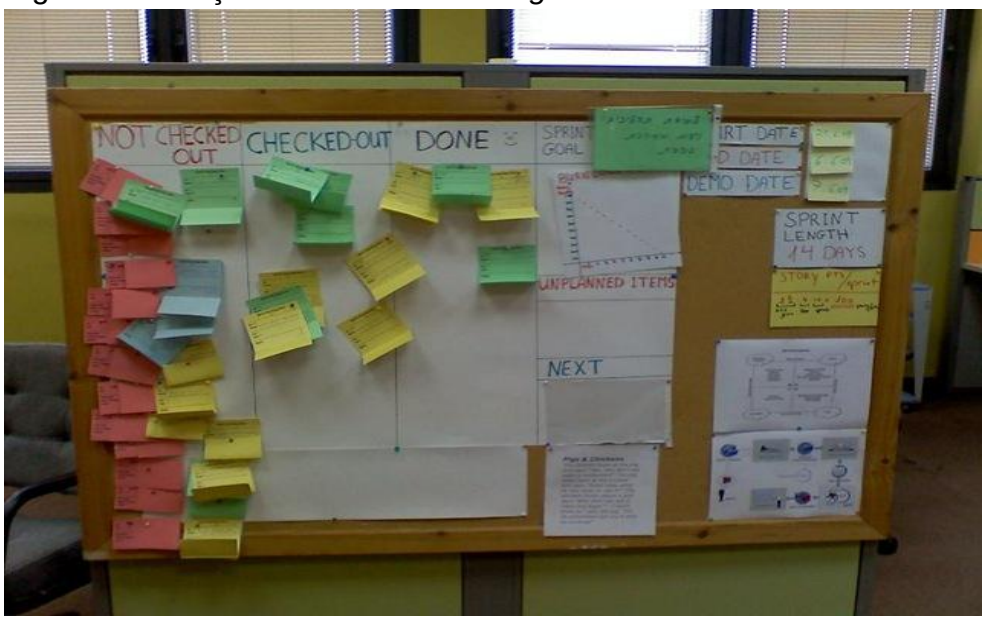
Para que o Product Owner obtenha sucesso, todos na organização precisam respeitar suas decisões. As decisões do *Product Owner* são visíveis no conteúdo e na priorização do *Product Backlog*. Essa visibilidade requer que o *Product Owner* faça seu melhor, o que faz o papel de *Product Owner* exigente e recompensador ao mesmo tempo.

Na criação do *Product Backlog*, o *Product Owner* deve listar as tarefas mais importantes para a manutenção correta do software. A partir da criação das funcionalidades do *Product Backlog*, surge os *sprints*, que são as fases do processo.

É importante manter-se alinhado com os objetivos específicos do projeto, portanto, toda alteração deve ser analisada e envolver a aprovação do cliente.

A criação do *Product Backlog* e seus *sprints* para a manutenção pode ser vista de acordo com a figura 7:

Figura 7 - Criação do *Product Backlog*



Fonte: http://uni4.com.br/blog/2011/scrum-conheca-o-sprint-backlog/

### 9.1.3 *Sprint Backlog*

O *Sprint Backlog* representa tudo o que deverá ser feito durante a próxima *Sprint* da manutenção. Ele surge a partir do que foi levantado e listado, pelo *Product Owner*, no *Product Backlog*.

Todos os itens que estão no *Product Backlog* devem estar preparados, estimados e priorizados para serem implementados, segundo a definição de preparado estabelecida no início do projeto.

A escolha dos itens que farão parte do *Sprint Backlog* deve estar de acordo com a meta de cada *sprint*. É essa meta que direciona o *Product Owner* e a Equipe Scrum para definir quais itens do *Product Backlog* farão parte *do Sprint Backlog* na manutenção do software.

O *Sprint Backlog* é criado durante a uma reunião de planejamento, que se pode chamar de *Sprint Planning Meeting*. Nesta reunião são definidos os papéis dos envolvidos na manutenção do sistema. Nesse momento, os itens de *Backlog* que serão considerados já devem estar preparados pelo *Product Owner*. Com os itens preparados, a Equipe Scrum estima o esforço de trabalho para cada um deles e, de acordo com a meta de cada *sprint*, esses itens são priorizados e uma determinada quantidade deles se torna o *Sprint Backlog*.

Durante o *Sprint Planning Meeting* o *Product Owner* prioriza os itens do *Product Backlog* e os descreve para a equipe. A equipe determina quais itens serão capazes de realizar durante a *sprint* que está por começar. Os itens são transferidos do *Product Backlog* para o *Sprint Backlog*. Ao fazer isso, a equipe divide cada item do *Product Backlog* em uma ou mais tarefas do *Sprint Backlog*. Isso faz com que o trabalho se divida entre os membros da equipe. Podem fazer parte do *Product Backlog* tarefas técnicas ou atividades diretamente relacionadas às funcionalidades solicitadas.

É aconselhável que nem todos os itens do *Sprint Backlog* estejam diretamente relacionados ao objetivo do projeto. Isso porque, caso não seja possível concluir um deles, o objetivo estará comprometido e, assim, a *Sprint* não terá sucesso. Incluir um novo item ao *Sprint Backlog* pode ocasionar na não conclusão de outro item. Por isso, itens novos só podem entrar no *Sprint Backlog* se estiverem relacionados ao objetivo e se forem cumpri-lo de uma forma que os itens já

presentes não cumprirão.

Diariamente, em uma *Sprint*, a equipe faz uma breve reunião de no máximo 15 minutos**,** com todos os participantes do projeto de manutenção presentes. O objetivo é cada integrante dizer o que fez no dia anterior, o que pretende fazer no dia que se inicia e se existe algum impedimento que está atrapalhando o seu trabalho.

Ao final de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma *Sprint Review Meeting* onde a equipe mostra o que foi alcançado neste *sprint*. Finalmente, faz-se uma *Sprint Retrospective* para identificar o que funcionou bem, o que pode ser melhorado e que ações serão tomadas para melhorar e a equipe parte para o planejamento do próximo *Sprint*.

Pode-se observar um exemplo de *Sprint Backlog* através do quadro 3, observada no modelo de Cisneiros (2009):

Quadro 3 - *Sprint Backlog*

| Funcionalidade | Prioridade | Custo-horas |
|---|---|---|
| **Modelagem de dados** | **1** | **32** |
| Definição de dados | 1.1 | 8 |
| Organização de tabelas | 1.2 | 12 |
| Relacionamento | 1.3 | 8 |
| Implementação em SGBD | 1.4 | 4 |
| **Cadastro e gerenciamento de usuário** | **2** | **26** |
| Formulários | 2.1 | 6 |
| Interação com cadastro na base de dados | 2.2 | 6 |
| Visualização de perfil | 2.3 | 6 |
| Mudança de dados | 2.4 | 4 |
| Relacionamento entre usuários | 2.5 | 4 |
| **Conversão de vídeo para visualização na Internet (Codec)** | **3** | **80** |
| Definição e Implementação de Codec | 3.1 | 56 |
| Integração de Codec com sistema | 3.2 | 24 |
| **Cadastro e gerenciamento de vídeos pelos usuários** | **4** | **48** |
| Upload de vídeos | 4.1 | 16 |
| Remoção de vídeos | 4.2 | 14 |
| Perfis de vídeos | 4.3 | 18 |

Fonte: http://www.devin.com.br/modelo-scrum/

Na *Sprint Planning Meeting* lista-se as tarefas ainda não cumpridas pela equipe, as tarefas que já estão em andamento e também as tarefas concluídas, agilizando de uma forma eficaz o andamento da manutenção do software.

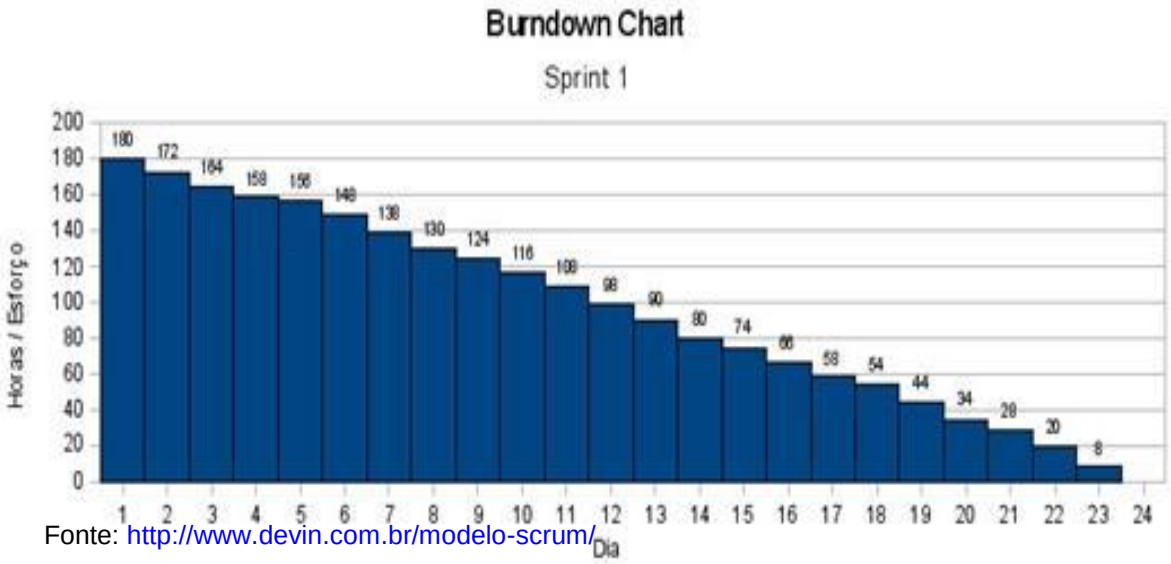
### 9.1.4 *Scrum Master*

Com o *Sprint Backlog* definido o *Sprint* pode ser iniciado. O *Scrum Master* deve acompanhar diariamente com a equipe o andamento das tarefas do *Sprint Backlog*, visando à meta do *Sprint* que é alcançada quando todos os itens do *Sprint Backlog* estão concluídos.

O *Scrum Master* deve manter algum indicador que apresente o cenário atual do andamento dos trabalhos, permitindo acompanhar as atividades concluídas e a expectativa de conclusão do *Sprint*. Através da aplicação *Sprint Burndown Chart* é possível acompanhar graficamente o desenvolvimento do *Sprint*, com uma visão diária da estimativa de tempo para o final do ciclo.

De acordo com o exemplo de Cisneiros (2009), a figura 8 nos mostra o *Burndown Chart* aplicado no modelo anterior:

Figura - 8 *Burndown Chart*.




Fonte: http://www.devin.com.br/modelo-scrum/

Para Sanchez (2007), o *Scrum Master* além de ser um líder do projeto deve estar sempre em contato com o *Product Owner* e também ser um mediador das tarefas do processo de manutenção.

O *Scrum Master* possui algumas responsabilidades que são de extrema importância para que a manutenção do software ocorra da forma mais correta possível. É ele quem assegura que a equipe funcione de forma produtiva, coopera para que as funções de cada membro da equipe sejam respeitadas e também é responsável de assegurar que a metodologia esteja sendo seguida.

Além disso, o *Scrum Master* deve resolver problemas pessoais que possam existir entre os membros da equipe. De acordo com Aguiar (2008) estudos mostram que 50% dos problemas de manutenção e desenvolvimento de software ocorrem por razões pessoais.

### 9.1.5 *Scrum Team*

No método Scrum da metodologia ágil, a equipe que ira realizar a manutenção do software, assim como no desenvolvimento, é chamada de *Scrum Team.*

A equipe deve possuir algumas características especificas para que o sucesso da manutenção seja obtido. Deve ser multifuncional, ter capacidade e conhecimento técnico sobre o processo de manutenção do software, organização e também serem capazes de definir as metas a serem atingidas em casa *sprint.*

### 9.1.6 Organização das Equipes

Para trabalhar de uma forma eficaz e bem distribuída, as empresas de softwares devem se adequar de modo que sua equipe possa render de forma mais produtiva. A organização das equipes de forma errada pode acarretar inúmeros problemas na manutenção de softwares. A iteração e colaboração entre os membros da equipe são fundamentais para que o Scrum seja aplicado de forma organizada e correta.

A estruturação de um projeto de manutenção de software precisa de alguns passos para obter o sucesso desejado. A alocação dos membros em cada equipe é um deles. As equipes devem ser criadas para suas devidas funções, mas cada uma dela deve ter especialistas em diversos assuntos, para que cada um interaja com o outro.

Para Morais (2012) após definir a alocação dos membros da equipe,

pode-se ter duas formas de organização. Em uma delas, cada localidade devera possuir todos os perfis de especialistas necessários e assim cada localidade forma uma equipe.

Com isso, é mais interessante montarmos equipes que possuam todas as especialidades e fazer com que eles interajam bastante. Após definirmos como alocar os membros, iremos apresentar duas possíveis formas de organização das equipes.

Na primeira formação, cada localidade possui todos os perfis necessários, e nesse caso cada localidade forma uma equipe. Na segunda formação, pode-se formar uma equipe com membros de diversas localidades. Porem, nas duas formações de equipes de manutenção, deve-se ter uma localidade central que manterá contato com o cliente, facilitando assim o processo.

## 9.2 O USO DA REENGENHARIA NA RESOLUÇÃO DE PROBLEMAS DE MANUTENÇÃO

Para resolver os inúmeros problemas relacionados à manutenção de software e seus custos, as empresas que pretendem aprimorar e melhorar seus softwares ficam restritas a algumas alternativas. Podem manter seus softwares desorganizados e com custos cada vez maiores, reconstruir os mesmos ou realizar a reengenharia para aumentar sua manutenibilidade.

A reengenharia é uma forma que as empresas estão buscando para manter ou refazer seus softwares de forma que elimine manutenções difíceis e as alterações de sua estrutura.

Outro fator que envolve a reengenharia é a qualidade em relação aos processos de desenvolvimento e manutenção de softwares. A garantia de qualidade na execução das tarefas somada com um bom gerenciamento de configuração garante que a reengenharia seja a melhor solução para os problemas na manutenção de software.

Dessa maneira, a aplicação de modelos de avanço na qualidade que originalmente eram utilizados somente no desenvolvimento de software apoiados também na reengenharia tornam o processo mais confiável e ágil.

### 9.2.1 Preparação e Planejamento do Processo de Reengenharia

A preparação do processo de reengenharia de software deve ter inicio com a definição dos limites do projeto. Deve-se ter o entendimento de seu escopo, os trabalhos a serem elaborados e as atividades a serem realizados, todos dentro dos padrões necessários para a obtenção dos produtos desejados.

Logo após a execução do sistema e do conhecimento de seu contexto pelo *Scrum Master*, podem ser realizadas entrevistas com os usuários de forma a aprimorar as informações disponíveis. A realização de entrevistas com os usuários do sistema possibilita conhecer realmente a necessidade do cliente, visando menor tempo de trabalho e menor custo para a empresa.

### 9.2.2 Resultados Obtidos

Com os estudos acompanhados nesse trabalho, a importância da metodologia ágil pode ser vista de uma forma simples e de fácil entendimento.

O estudo dos métodos ágeis surgiu como uma nova maneira de projetar sistemas. Essas mesmas formas podem ser utilizadas na manutenção de softwares, adaptando-as e utilizando algumas técnicas de manutenção, neste caso a reengenharia de software.

A reengenharia de software aplicada juntamente com o método scrum apresenta um menor custo de manutenção nos softwares e possibilita empresas a se aproximarem cada vez mais de seus clientes e parceiros.

Neste trabalho pode-se acompanhar a diferença entre os métodos tradicionais e dos métodos ágeis, em especial o método Scrum. Também foram estudados alguns artefatos do método Scrum que contribuem para a busca da agilidade.

A reestruturação de softwares não é uma tarefa fácil, a resolução destes problemas pode ser obtida através da reengenharia do mesmo, utilizando a engenharia reversa.

O objetivo do trabalho proposto foi mostrar que assim como o método Scrum vem sendo muito usado na elaboração de softwares, o mesmo pode também ser usado na manutenção desses mesmos softwares, utilizando suas técnicas e

seus artefatos com o auxilio da reengenharia.

Pode-se também estudar seus artefatos detalhadamente e quais os papéis de cada membro da equipe no projeto de manutenção.

Não foi necessário um estudo de caso neste trabalho, pois o mesmo visou apenas comparar os métodos ágeis e tradicionais e propor a utilização de um método já existente na resolução de problemas em softwares legados, porém, pode ser realizado em trabalhos posteriores, podendo obter pesquisas através de empresas de pequeno porte.

## 10 CONCLUSÃO

Os métodos de desenvolvimento e manutenção de software vêm sofrendo mudanças ao longo do tempo. Os métodos tradicionais estão perdendo seu espaço para novos métodos, chamados de métodos ágeis.

A metodologia ágil surge como uma nova alternativa tanto no projeto quanto na manutenção de softwares.

O método Scrum da metodologia ágil busca colaborar de forma a diminuir os custos de manutenção utilizando técnicas muito bem elaboras na manutenção de sistemas legados. O seu uso é organizado, com pouca documentação, focado principalmente no real problema do usuário.

Usando a reengenharia de software como caminho para essa agilidade, podemos obter resultados satisfatórios na analise e manutenção de softwares, tendo em vista todas as dificuldades de manutenção através dos métodos tradicionais.

Pode-se então concluir que a reengenharia de software aplicada juntamente com o método Scrum da metodologia ágil contribui para a reestruturação de softwares, possibilitando a equipe de software e aos usuários um maior poder de utilização de todas as ferramentas do sistema.

# REFERÊNCIAS


AGILE MANIFESTO. **Manifesto for Agile Software Development**. Disponível em: <http://www.agilemanifesto.org>. Acesso em: 20 abr 2012.

ALVES, Fernanda. **Revista engenharia de software.** v.7, ano 1, 2008.

BASSI, Dairton. **Revista Engenharia de Software**. v. 8, ano 1, 2008.

BORGONOVO, Ana Marta; SILVA, Roberta Santos da. **Aplicação de metodologia ágil em grandes empresas com pequenos grupos de TI**. 2007. 82 f. - Universidade Federal de Santa Catarina, Florianópolis - Sc, 2007.

DAVIS, Willian S. **Análise e projeto de Sistemas.** Rio de Janeiro: Addison-Wesley, 1994. 378 p.

FONSECA, Isabella. **Revista engenharia de software**. v. 4, ano1, 2008.

GAMBA, Mateus Luiz. **Aplicação de Métricas de Software no método Scrum da Metodologia Ágil.** 2009. 102 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade do Extremo Sul Catarinense, Criciúma.

ISOTTON NETO, Erasmo. **Scrumming**: Ferramenta Educacional para Ensino de Práticas do SCRUM. 2004. 80 f. - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2004. 83

MORAIS, Lenildo. **Revista engenharia de software.** v.48, ano 4, 2012.

PADUELLI, Mateus Maida. **Revista engenharia de software.** v. 11, ano 1, 2008.

PRESSMAN, Roger S. **Engenharia de Software**. 6. ed. São Paulo: McGraw-Hill, 2006. 720 p.

_____. **Engenharia de Software**. São Paulo: Makron Books, 1995. 1056 p.

SCHWABER, Ken. **Agile Project Management with Scrum**. Microsoft Press, 2004.

SERRA, Ana Paula Gonçalves. **Revista engenharia de software.** v. 11, ano 1, 2009.

ZANATTA, Alexandre Lazaretti; VILAIN, Patrícia. **Uma análise do método ágil Scrum conforme abordagem nas áreas de processo. Disponível em:** < http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf>. Acesso em: 17 mai. 2012.

ZANATTA, Alexandre Lazaretti. **XScrum**: uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI. 2004. 180 f. - Universidade Federal de Santa Catarina, Florianópolis, 2004.

# APÊNDICES

APÊNDICE A – Artigo Científico

## REENGENHARIA DE SOFTWARE APLICADA NA MANUTENÇÃO DE SISTEMAS POR MEIO DA METODOLOGIA ÁGIL NO MÉTODO SCRUM

**Mauricio Soprana Coelho**
UNESC – Universidade do Extremo Sul Catarinense

***Abstract.*** *This work aims to use the software to improve maintenance of existing projects using the agile methodology and software reengineering based study. The method to be used for development of this work is the Scrum method, which is considered to be a nimble and flexible. The agile methodologies have been growing fast and solid, making all software processes more participatory for both the user and for the maintenance team.*

*Keywords: Scrum. Agility. Maintainability. Reengineering.*

***Resumo.*** *Este trabalho visa utilizar a manutenção de softwares para melhoramento de projetos já existentes utilizando a metodologia ágil e a reengenharia de software como base de estudo. O método a ser utilizado para desenvolvimento desde trabalho é o método Scrum, que é considerado um método ágil e flexível. As metodologias ágeis vêm crescendo de forma rápida e sólida, tornando todos os processos de softwares cada vez mais participativos, tanto para o usuário quanto para a equipe de manutenção.*

*Palavras-chave: Scrum. Agilidade. Manutenibilidade. Reengenharia.*

### 1. Engenharia de Software

De acordo com Sommerville (2003) a engenharia de software se ocupa em todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação.

Neste contexto é que esta pesquisa visa aprofundar o estudo e uso da metodologia ágil e da reengenharia de software em sistemas, não somente a fim de modernizar, mas sim adaptá-los as novas necessidades ao qual estiver inserido.

## 2. Justificativa

Nesse aspecto é que a reengenharia de software sendo aplicada por meio da metodologia ágil no método Scrum pode melhorar a qualidade de um sistema sem perder informações e investimentos. A reengenharia reutiliza os modelos e conhecimentos do software original e migra para uma plataforma moderna, além de melhorar a base e a estrutura do software dando sobrevida ao mesmo, reutilizando códigos.

## 3. Metodologia Ágil

O desenvolvimento através da metodologia ágil, como o próprio nome já sugere, procura tornar o projeto do software cada vez mais produtivo, tanto no desenvolvimento quanto na manutenção do mesmo, diminuindo a documentação e criando uma iteração entre usuários e programadores.

No uso da metodologia ágil, cada tarefa definida deve ser verificada quanto ao seu percentual de andamento e também quanto à sua qualidade através de testes e entregas que são feitas com maior freqüência, ocasionando assim a iteração entre usuário e programador.

## 4. Método Scrum

O método Scrum da metodologia ágil é um método que gerencia projetos ou atividades complexas com base no processo iterativo e incremental (ZANATTA, 2004).

O método Scrum tem como idéia construir o todo aos poucos, refinando os requisitos, equalizando o entendimento sobre eles e corrigindo o rumo sempre que uma mudança provoque alterações significativas.

## 5. Reengenharia de Software

Segundo Serra (2009) a reengenharia de software pode ser definida como uma aprimoração de software, ou seja, uma abordagem disciplinada para migrar softwares legados em softwares evolutivos. O processo de reengenharia de software aplica os princípios da engenharia de software em um software legado para atender

requisitos existentes e novos requisitos.
As atividades da reengenharia de software garantem que a mesma seja aplicada da melhor maneira possível, garantindo a qualidade nas alterações do software legado. O uso da engenharia reversa destaca-se por ser semelhante e até mesmo confundido com a reengenharia de software.

## 6. Manutenção de Softwares

De acordo com Paduelli (2008) a atividade de manutenção de software é identificada como a modificação de um projeto de software já entregue ao cliente, para a correção de erros, melhora de desempenho ou ainda para adaptação do software a um ambiente modificado.
A idéia de investir na evolução do software ou abandoná-lo para iniciar um novo projeto não é uma tarefa fácil. O custo envolvido e a confiabilidade do software após a manutenção são fatores importantes que devem ser considerados.

## 7. Método Scrum e Reengenharia aplicados na manutenção de softwares

De acordo com o que já foi visto anteriormente, o custo da manutenção de softwares vem crescendo de forma acelerada. O uso de Scrum na manutenção de software através da reengenharia visa suprir essas necessidades, contribuindo também para a diminuição de custos e uma maior agilidade na resolução dos problemas de manutenção.
Os principais papéis do Scrum devem ser aplicados na manutenção de softwares: *Product Backlog, Product Owner, Scrum Master* e *Scrum Team* (equipe do projeto), além de outros papéis que tornam o uso do Scrum cada vez mais simplificado.
Diante disso, propõe-se um estudo sobre o uso dessas práticas na manutenção do software, utilizando a reengenharia para manter um software em funcionamento da melhor forma possível. Alem disso, pretende-se utilizar o método Scrum da metodologia ágil e suas funções para o funcionamento da manutenção.

### 7.1 Organização das equipes

Para trabalhar de uma forma eficaz e bem distribuída, as empresas de softwares devem se adequar de modo que sua equipe possa render de forma mais produtiva. A organização das equipes de forma errada pode acarretar inúmeros problemas na manutenção de softwares. A iteração e colaboração entre os membros da equipe

são fundamentais para que o Scrum seja aplicado de forma organizada e correta.

A estruturação de um projeto de manutenção de software precisa de alguns passos para obter o sucesso desejado. A alocação dos membros em cada equipe é um deles. As equipes devem ser criadas para suas devidas funções, mas cada uma dela deve ter especialistas em diversos assuntos, para que cada um interaja com o outro.

## 8. Referências

GAMBA, Mateus Luiz. Aplicação de Métricas de Software no método Scrum da Metodologia Ágil. 2009. 102 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade do Extremo Sul Catarinense, Criciúma.

MORAIS, Lenildo. Revista engenharia de software. v.48, ano 4, 2012.

PADUELLI, Mateus Maida. Revista engenharia de software. v. 11, ano 1, 2008.

ZANATTA, Alexandre Lazaretti; VILAIN, Patrícia. Uma análise do método ágil Scrum conforme abordagem nas áreas de processo. Disponível em: < http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf>. Acesso em: 17 mai. 2012.

ZANATTA, Alexandre Lazaretti. XScrum: uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI. 2004. 180 f. - Universidade Federal de Santa Catarina, Florianópolis, 2004.