IRINEU AMBROZANO FILHO

APLICAÇÃO DE GESTÃO EM PROJETOS QUE UTILIZAM MÉTODOS ÁGEIS - EXTREME PROGRAMMING

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção da certificação no curso de MBA em Tecnologia da Informação.

Orientadora:

Profª. Gabriela Maria Cabel Barbarán

São Paulo

2008

FICHA CATALOGRÁFICA

Ambrozano Filho, Irineu

Aplicação de gestão em projetos que utilizam métodos ágeis – extreme programming / I. Ambrozano Filho. -- São Paulo, 2008.
55 p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Tecnologia da informação 2. Métodos ágeis 3. Desenvolvimento de software 4. Programação extrema I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

A meus pais, esposa e filha, motivação em minha vida,

dedico esta monografia.

AGRADECIMENTOS

Agradeço à Professora Gabriela Maria Cabel Barbarán que gentilmente forneceu seu conhecimento e experiência, os quais me ajudaram muito no desenvolvimento desta monografia.

Aos meus tios Sérgio e Sueli pelo apoio e estímulo constante em minha vida profissional.

Ao Departamento de Informática da Reitoria da USP pelo grande incentivo e colaboração.

E a todos os professores e amigos pelos conhecimentos compartilhados.

RESUMO

Este trabalho apresenta uma proposta das melhores práticas para o gerenciamento de projetos de software que utilizam XP como metodo ágil de desenvolvimento.

Para o desenvolvimento do trabalho foi realizado um estudo exploratório sobre contexto da metodologia de desenvolvimento ágil de software, Extreme Programming e os conceitos de gerenciamento de desenvolvimento de software, Scrum e CMMI.

Foi analisado, também, um estudo de caso de um projeto de desenvolvimento de software que utilizou os valores e práticas do Extreme Programming combinado com o método ágil Scrum para as atividades de Gestão.

Os resultados do estudo mostraram que as atividades de Gestão do Método Scrum aplicadas permitem uma adequada visualização do desenvolvimento do projeto, seja para a equipe assim como também para o cliente, que têm uma participação ativa no projeto.

O estudo de caso, permitiu verificar também que as práticas de gestão do Método Scrum são compatíveis com algumas atividades das PA´s do CMMI.

ABSTRACT

This paper presents a proposal of the best practices for the software projects management which use XP as an agile method of development.

For the development of this paper, an exploratory study on the context of the methodology of agile software development, Extreme Programming and the concepts of software development management, Scrum and CMMI was carried through.

It was also analyzed the study of the case of a software development project which used the Extreme Programming values and practices combined with the Scrum agile method for the management activities.

The study results had shown that the activities of the Scrum management method applied allowed an adequate visualization of the project development, either for the team or for the customer, which has an active participation in it.

The case study also allowed verifying that the management practices of the Scrum method are compatible with some activities of the CMMI PA's.

LISTA DE FIGURAS

LISTA DE TABELAS

ABREVIATURAS E SIGLAS

CAR – Análise Causal e Resolução

CM – Gerência de Configuração

CMM – Capability Maturity Model

CMMI – Capability Maturity Model Integration

DAR – Análise de Decisão e Resolução

DI – Departamento de Informática

IPM – Gerência Integrada de Projeto

PA – Áreas de Processo

MA – Medição e Análise

OID – Inovação Organizacional e Implantação

OPD – Definição do Processo Organizacional

OPF – Foco no Processo Organizacional

OPP – Performance (desempenho) do Processo Organizacional

OT – Treinamento Organizacional

PI – Integração de Produtos

PMC – Acompanhamento e Controle de Projeto

PP – Planejamento de Projeto

PPQA – Garantia da Qualidade de Processo e Produto

QPM – Gerência Quantitativa de Projeto

RD – Desenvolvimento dos Requisitos

REQM – Gerenciamento de Requisitos

RSKM – Gerência de Risco

SAM – Gerência de Acordos com Fornecedores

SEI – Software Engineering Institute

SG – Práticas Genéricas

SP – Práticas Específicas

TS – Solução Técnica

VAL - Validação

VER – Verificação

XP – Extreme Programming

SUMÁRIO

1. INTRODUÇÃO

Atualmente a produção de software com alta qualidade e produtividade está relacionada entre os fatores críticos para o sucesso de diversas áreas de negócios. O desenvolvimento de software é uma atividade que apresenta importância econômica crescente, observa-se que os produtos de software nem sempre atendem satisfatoriamente as necessidades de seus usuários, principalmente com relação aos aspectos ligados à qualidade, prazos de desenvolvimento e custos de produção e manutenção.

Podemos perceber que as empresas que têm uma estrutura formalmente organizada para o cumprimento dos objetivos de produção e um enfoque orientado para obtenção de resultados com qualidade, vêm se preocupando com o produto final entregue aos seus clientes, que cada vez mais estarão dependentes de softwares para resolverem seus problemas do dia-a-dia.

Os esforços das empresas que buscam diferenciar-se no mercado seguem no sentido de melhorias nos projetos e processos de desenvolvimentos de software, sobretudo na rapidez, convivendo com ambientes de negócios que requerem mudanças freqüentes em seus processos, as quais afetam os projetos de software. Os processos de desenvolvimento tradicionais são caracterizados por uma grande quantidade de atividades e artefatos que buscam proteger o software contra mudanças, o que faz pouco ou nenhum sentido, visto que os projetos deverão adaptar-se às tais mudanças ao invés de evitá-las.

Concentrando esforços da equipe de desenvolvimento em atividades que geram resultados rapidamente na forma de software testado e alinhado às necessidades dos usuários, o Extreme Programming (XP) simplifica e organiza o trabalho combinando técnicas eficazes e eliminando atividades redundantes.

Por fim, reduzem o risco dos projetos, desenvolvendo software de forma interativa e reavaliando permanentemente as prioridades dos

usuários, porém surge a necessidade da aplicação de gestão de projetos que utilizem métodos ágeis e possibilitem a criação de software de alta qualidade, de maneira ágil, econômica e flexível.

Neste trabalho apresentaremos um estudo de caso que representa o desenvolvimento de um projeto de software que utiliza os valores e práticas do Extreme Programming combinado com a metodologia de gerenciamento, Scrum.

1.1 Objetivos

Visando a melhoria no desempenho dos projetos de desenvolvimento de software, referente à qualidade, agilidade e custo, este trabalho busca identificar como as práticas do método de gerenciamento Scrum monitoram e controlam os projetos de softwares que utilizam XP.

Para o embasamento das práticas de gestão Scrum será usada uma análise comparativa entre a metodologia de maturidade, CMMI e o guia para gerência, Scrum. As práticas serão identificadas em seu uso através da aplicação em um estudo de caso.

1.2 Justificativa

A diversidade de processos de desenvolvimento de software, as deficiências em gestão decorrente não só da complexidade e variabilidade dos ambientes de desenvolvimento, mas também, da falta de instrumentos gerenciais capazes de controlar e apoiar as decisões ao longo do processo de produção faz buscar respostas às perguntas, tais como:

- Como gerenciar equipes que utilizam XP?
- Como conduzir indivíduos da equipe que não possuem conhecimento técnico adequado?

- Como agir quando alguns dos indivíduos da equipe abandonam o projeto durante o seu desenvolvimento?
- Como produzir software de qualidade com as diversas intercorrência durante o desenvolvimento do projeto?

A busca por estas respostas, neste trabalho, identificará quais as atividades de cada processo de gestão são mais adequadas ao desenvolvimento de projetos ágeis, podendo com isso concluir qual o melhor método a ser utilizado.

1.3 Metodologia de Elaboração

Para a elaboração deste trabalho foram considerados os seguintes critérios metodológicos:

1.3.1 Abordagem do Trabalho

O trabalho tem uma abordagem qualitativa, a qual permite um relacionamento direto do pesquisador com o ambiente em estudo. Neste caso o pesquisador conhece e participa do ambiente de desenvolvimento dos projetos de software.

1.3.2 Tipo de Pesquisa

O tipo de pesquisa deste trabalho é exploratória, a qual permite um aprofundamento do tema de estudo. Neste caso será feito um levantamento bibliográfico dos diferentes temas a serem abordados no trabalho, tais como: métodos ágeis: XP e Scrum e o modelo de Gestão CMMI. Tais temas serão complementados com dados coletados em um estudo de caso.

1.3.3 Método de Pesquisa

A forma de coleta de dados será feita através do método do estudo de caso. O pesquisador através da observação direta será capaz de descrever o ambiente de trabalho da empresa e identificar como são aplicadas as teorias estudadas.

1.3.4 Estrutura do Trabalho

O presente trabalho está divido em seis capítulos, o presente capítulo, Introdução, visa apresentar um breve histórico da área de TI, a situação da qualidade de software no mundo para compreender a necessidade dos esforços que vêm sendo empregados na sua melhoria. Também foram abordados os objetivos, justificativa focado em gestão de desenvolvimento de software.

O segundo capítulo, Metodologias Ágeis, apresenta conceitos e tem como objetivo aprofundar o entendimento da metodologia de desenvolvimento de software, Extreme Programming.

O capítulo seguinte (terceiro) tem como objetivo apresentar o modelo e o processo de gerenciamento no desenvolvimento de software. O CMMI e o Scrum serão os focos posteriormente expostos. Com isso no quarto capítulo faremos uma comparação entre o modelo CMMI e o processo de gerenciamento Scrum junto com a metodologia de desenvolvimento XP.

O quinto capítulo, Estudo de caso, apresenta o gerenciamento do desenvolvimento de software em um departamento de informática, onde é utilizada a combinação Scrum e XP.

Concluindo a dissertação, o sexto capítulo, conclusão, apresenta uma síntese dos temas estudados e das análises do estudo de caso. Espera-se que estas conclusões possam servir para melhorar a compreensão de como aplicar gestão em projetos que utilizam métodos ágeis. Neste capítulo serão indicados os trabalhos futuros.

Finalizando, será detalhada a bibliografia usada para o desenvolvimento do trabalho.

2. METODOLOGIAS ÁGEIS

Os objetivos das metodologias ágeis são de interesse de todos os que trabalham com a produção de software: desenvolvendo de forma mais rápida, barata e com menos burocracia, porém mantendo-se a qualidade exigida pelo cliente.

Em fevereiro de 2001, um grupo formado por dezessete desenvolvedores experientes, consultores e líderes da comunidade de software se reuniu em Utah (EUA) para discutir idéias e procurar uma alternativa aos processos burocráticos e às práticas adotadas nas abordagens tradicionais da Engenharia de Software e gerência de projetos. Dessa reunião surgiu o Manifesto do Desenvolvimento Ágil de Software [1], que destaca as diferenças com relação às abordagens tradicionais e define seus valores:

- Indivíduos e interações são mais importantes que processos e ferramentas;
- Software funcionando é mais importante que documentação completa e detalhada;
- Colaboração com o cliente é mais importante que negociação de contratos;
- Adaptação às mudanças é mais importante que seguir um plano.

O XP–Extreme Programming, FDD – Feature Driven Development, MSF–Microsoft Solutions Framework e SCRUM são algumas das metodologias ágeis.

A metodologia XP será descrita a seguir por ser uma das principais metodologias ágeis.

2.1 Extreme Programming – XP

Extreme Programming (XP) é uma metodologia ágil de desenvolvimento de software para pequenas e médias equipes. O XP adapta-se a mudanças, como prazos, equipe, projetos, ferramentas e listas de funcionalidades exigidas pelo cliente.

. É uma maneira eficiente, de baixo risco, científica e divertida de desenvolver software [2], baseada em valores de simplicidade, comunicação, *feedback*, e coragem [3].

Comunicação: XP foca em construir um entendimento pessoa-a-pessoa do problema, com o uso mínimo de documentação formal e com o uso máximo de interação "cara-a-cara" entre as pessoas envolvidas no projeto. As práticas de XP como programação em pares, testes e comunicação com o cliente têm o objetivo de estimular a comunicação entre gerentes, programadores e clientes.

Simplicidade: XP sugere que cada membro da equipe adote a solução mais fácil que possa funcionar. O objetivo é fazer aquilo que é mais simples hoje e criar um ambiente em que o custo de mudanças no futuro seja baixo. O objetivo dessa abordagem adotada por XP é evitar a construção antecipada de funcionalidades, como é feita em muitas metodologias tradicionais, que acabam muitas vezes não utilizadas.

Feedback: Os programadores obtêm feedback sobre a lógica dos programas escrevendo e executando casos de teste. Os clientes obtêm *feedback* através dos testes funcionais criados para todas as estórias (casos de uso simplificados). O *feedback* é importante, pois possibilita que as pessoas aprendam cada vez mais sobre o sistema e assim corrijam os erros e melhorem o sistema.

Coragem: Ela é necessária para que realmente se aplique XP como deve ser aplicado. Exemplos de atitude que exigem coragem são: alterar código já escrito e que está funcionando; descartar o código e reescrever; e permitir código compartilhado por todos. Estes exemplos

de atitudes podem ser necessários para trazer melhorias ao projeto e não devem ser evitadas simplesmente devido ao medo de tentá-las.

2. 1.1 Requisitos do XP

A Extreme Programming é recomendada para equipes pequenas, segundo Kent Beck [2], localizados num mesmo ambiente físico, um fator que realmente é um problema, pois em uma EQUIPE GRANDE a comunicação torna-se difícil e o XP não funciona sem este meio. O conselho dele é claro, não use XP para equipes com mais de dez pessoas, em projetos onde os requisitos são fracamente conhecidos e/ou sujeitos as mudanças contínuas (projetos exploratórios).

As iterações do XP costumam ser curtas, provendo constantes versões do produto (releases) para o cliente, que por sua vez provê comentários e opiniões que realimentam a próxima iteração. O objetivo do XP é tornar o projeto flexível, diminuindo o custo a possíveis mudanças.

2.1.2 Características do XP

Entre várias características de XP, podemos destacar três que são primordiais: o diálogo, o planejamento baseado em duas semanas e o controle de qualidade.

No diálogo, existem duas regras preliminares, a definição de cliente e programador. O cliente é a pessoa ou grupo de pessoas que irão representar os usuários do sistema. É responsável por identificar as características do sistema que os programadores deverão implementar, provendo detalhes dos testes de aceite para as todas as características definidas e a prioridade de implementação. Sendo assim os programadores devem implementar as características que o cliente quer, na prioridade definida por ele, e passar por todos os testes

especificados. Os programadores poderão não implementar características que não foram definidos explicitamente pelo cliente, mas são responsáveis por estimar quanto tempo gastará para implementar cada características do sistema.

Com a divisão de responsabilidades definidas, o planejamento do projeto é representado pelo diálogo entre cliente e programador. O cliente fala aos programadores quais características eles querem implementar na próxima interação, e os programadores dizem se é possível a implementação destas características, e quanto tempo gastará para implementá-las. Definindo isto, o cliente poderá remover ou trocar alguma característica, mas não pedir mais do que foi definido para próxima interação, e os programadores implementarão o que foi pedido, mas não poderão selecionar a ordem da implementação.

O planejamento em duas semanas é simples. Pode ser que não se confie em um programador para desenvolver um sistema que demorará um ano, mas provavelmente confiaria para um sistema que demoraria duas semanas. Em XP, este seria o tempo entre o cliente passar ao programador uma característica a ser implementada, e o programador executá-la. A análise detalhada que inclui planejamento do projeto, modelo de análise e o modelo de design são criados no início de cada interação e é referente somente a aquela interação. Melhor que fazer longos planejamentos e modelos complexos, é fazer que duas semanas de trabalho sejam bem valiosas, onde se pode avaliar o que foi feito e planejar as próximas duas semanas.

A rápida sucessão dos ciclos de planejamentos é como um prolongamento da conversação. Os participantes entram num ritmo em que fazem planejamentos curtos, executam e avaliam.

O desenvolvimento acaba ficando simples porque o rápido *feedback* remove os erros. A cada duas semanas, o cliente olha sistema já construído e avalia se as últimas duas semanas houve melhoria. Se não houve, o cliente direciona as próximas duas semanas de trabalho para melhorar o produto. Em poucas iterações o cliente consegue

perceber se a equipe de desenvolvimento, na velocidade em que as coisas estão acontecendo, está atendendo suas expectativas, e, caso não esteja poderá parar todo o projeto e afastar a equipe de desenvolvimento.

O controle de qualidade é sempre uma preocupação, pois é normal acontecer várias mudanças em cada interação. Por causa disso o XP procura muita simplicidade dos programadores, eles devem deixar o código o mais simples possível e aprovado em todos os testes de aceite. Quando o código é re-trabalhado de interação para interação, ele é continuamente reduzido para o estado mais simples possível que os programadores acharem.

2.2 Práticas – Regras

O XP baseia-se em doze práticas ou regras concisas e diretas [4], listadas abaixo:

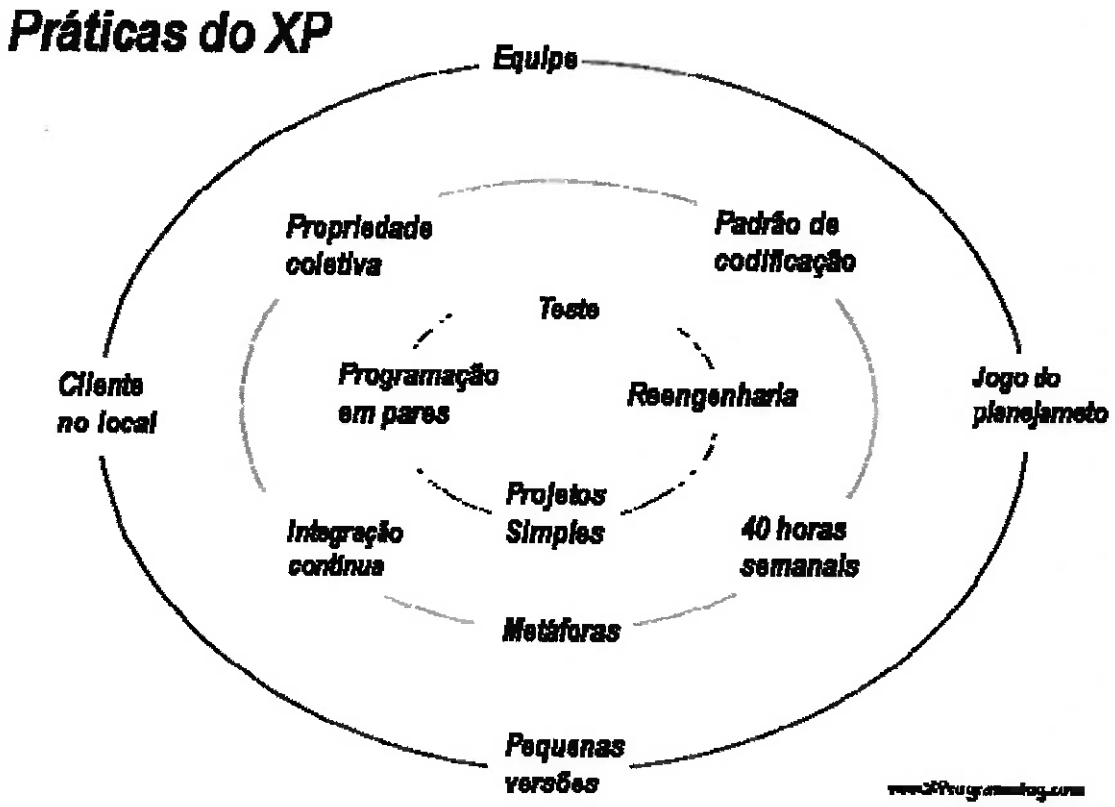

Figura 1 - Práticas do XP

(Fonte: http://www.xprogramming.com/images/circles.jpg)

1. Equipe: todos os contribuintes de um projeto XP sentam-se juntos. Nesta equipe deve-se incluir um representante do negócio, cliente, que prevê os requisitos e definem as prioridades, testadores, que ajudam os clientes a definir a aceitação testes, analistas ajudando o cliente a definir os requisitos, um treinador, que ajuda a manter a equipe na pista, e facilita o processo e um gestor, fornecendo recursos, a manipulação comunicação externa, coordenando atividades. As melhores equipes não têm especialistas e sim contribuintes com habilidades especiais;
2. Jogo do planejamento: no início de cada interação, clientes, gerentes e programadores se encontram para definir, estimar e priorizar os requerimentos. A idéia é elaborar um plano aproximado no início no projeto e fazer um refinamento à medida que as necessidades e requisitos se tornem mais conhecidos;
3. Programação em pares: dois programadores utilizando o mesmo equipamento escrevem o código;
4. Pequenas versões: as versões devem ser tão pequenas quanto possíveis e trazerem valor para o negócio. Uma versão inicial do software deve ser colocada em produção após um pequeno número de iterações e, em seguida, outras versões devem ser disponibilizadas tão logo faça sentido;
5. Metáforas: clientes, gerentes e programadores criam metáforas ou conjunto de metáforas para modelagem do sistema;
6. Projeto simples: os programadores são estimulados a desenvolver o código do software o mais simples possível;
7. Testes: os programadores devem criar os testes de unidade antes ou mesmo durante o desenvolvimento do código do sistema. Os clientes, por sua vez, escrevem os testes de aceitação. Ao final de cada iteração a bateria de testes deve ser conduzida;

8. Reengenharia de software: técnica que permite a melhoria de código sem a mudança de funcionalidade deve ser executada pela equipe do projeto, o tempo todo;
9. Integração contínua: os programadores devem integrar os novos códigos ao software rapidamente e com a maior freqüência possível;
10. Propriedade coletiva do código: o código do programa deve ser propriedade de toda a equipe e qualquer integrante pode fazer alterações sempre que for necessário;
11. Cliente no local: o cliente deve trabalhar com a equipe de projeto a todo o momento, respondendo perguntas, realizando testes de aceitação e assegurando que o desenvolvimento do software esteja sendo feito a contento;
12. Semana de 40 horas: como trabalhar por longos períodos reduz o desempenho, o conteúdo de cada iteração deve ser planejado de forma a não haver necessidade de realização de horas extras, fazendo com que os programadores estejam renovados e ansiosos a cada manhã e cansados e satisfeitos à noite;
13. Padrão de codificação: no início do projeto deve ser criado um padrão de codificação, simples e aceito por toda a equipe, que deverá ser seguido de forma a não permitir a identificação de quem desenvolveu determinada parte do código e a auxiliar a condução do trabalho.

2.3 Ciclo de Vida

O ciclo de vida ideal do XP consiste em seis fases [2];[4]:

A **fase de exploração** é anterior à construção do sistema. Nela, investigações de possíveis soluções são feitas e verifica-se a viabilidade de tais soluções. Os programadores elaboram possíveis arquiteturas e tentam visualizar como o sistema funcionará considerando o ambiente

tecnológico (hardware, rede, software, desempenho, tráfego) onde o sistema irá rodar. Com isso, os programadores e os clientes vão ganhando confiança, e quando eles possuírem estórias suficientes, já poderão começar a construir o primeiro release do sistema. Em [2] sugere-se que seja gasto no máximo duas semanas nessa fase.

A **fase de planejamento** inicial deve ser usada para que os clientes concordem em uma data para lançamento do primeiro release. O planejamento funciona da seguinte forma: Os programadores, juntamente com o cliente, definem as estórias (use case simplificados) a serem implementadas e as descrevem em cartões. Os programadores assinalam certa dificuldade para cada estória e, baseados na sua velocidade de implementação, dizem quantas estórias podem implementar em uma iteração. Após esta etapa, os clientes escolhem as estórias de maior valor para serem implementadas na iteração – a qual é chamada de planejamento de iteração. Este processo se repete até terminar as iterações do release, sendo que o tempo para cada iteração deve ser de uma a três semanas e para cada release de dois a quatro meses.

Na **fase das iterações** do release são escritos os casos de teste funcionais e de unidade. Os programadores vão seguindo mais ou menos o seguinte fluxo de atividades na seguinte ordem (Em cada iteração): escrita dos casos de testes; projeto e refatoramento; codificação; realização dos testes; e integração. À medida que esse fluxo vai seguindo, o sistema vai construindo segundo os princípios, valores e práticas apresentados nas seções anteriores. Depois de terminado o primeiro release, já terá uma idéia melhor das tecnologias e do domínio do problema de modo que as iterações poderão ser mais curtas nos releases subseqüentes podendo fazer estimativas mais confiáveis com o que se aprendeu das iterações passadas.

A **fase de produção** tem início após o final do primeiro release e depois de construído, é colocado para rodar em um ambiente que simula o ambiente de produção para ver seu comportamento em termos de desempenho. Podem-se fazer testes de aceitação adicionais para simular o funcionamento real do sistema no ambiente alvo.

A **fase de manutenção** pode ser considerada como uma característica inerente a um projeto XP. Em XP você está simultaneamente produzindo novas funcionalidades, mantendo o sistema existente rodando, incorporando novas pessoas na equipe e melhorando o código. Mecanismos como: refatoramento, introdução de novas tecnologias, e introdução de novas idéias de arquitetura podem ser utilizados em um projeto XP. É importante ressaltar que a manutenção dada em um sistema que já está em produção deve ser feita com muita cautela, pois uma alteração errada pode paralisar o funcionamento do sistema resultando em prejuízos para o cliente.

A **fase de morte** corresponde ao término de um projeto XP. Existem duas razões para se chegar ao final de um projeto, uma boa e a outra ruim. A boa razão é quando o cliente já está satisfeito com o sistema existente e não enxerga nenhuma funcionalidade que possa vir a ser implementada no futuro.

A má razão para a morte em XP seria a de o projeto ter se tornado economicamente inviável, devido a dificuldades de adicionar funcionalidades a um custo baixo e devido a uma alta taxa de erros. A documentação final do sistema é gerada e mais mudanças não são feitas na arquitetura. A morte do projeto acontece também quando o sistema não gera os benefícios esperados pelo cliente ou quando não há orçamento para mantê-lo.

3. GESTÃO DE PROJETOS

Gerenciar, administrar, coordenar ou gerir um projeto é a aplicação de técnicas, conhecimento e habilidades para garantir que um projeto tenha sucesso, desde iniciá-lo até finalizá-lo, passando pelas etapas de planejamento, execução e atividades de controle.

O gerenciamento de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender aos seus requisitos. O gerenciamento de projetos é realizado através da aplicação e da integração dos seguintes processos de gerenciamento de projetos: iniciação, planejamento, execução, monitoramento e controle, e encerramento [5].

A seguir descreveremos dois métodos para gerenciar um projeto, CMMI e Scrum.

3.1 Melhores Práticas – CMMI

Na década de 80, o SEI (Software Engineering Institute - Instituto de Engenharia de Software) dos Estados Unidos, para avaliar a qualidade dos softwares desenvolvidos pelas empresas, criou o CMM (Capability Maturity Model — Modelo de Maturidade da Capacitação).

O CMM é um modelo estruturado em estágios de maturidade, baseado nos princípios que norteiam a qualidade de software, a gerência de processos e o controle de qualidade estatístico, peças fundamentais dentro da Engenharia de Software que, de forma substancial, visam o controle quantitativo do processo de software [6].

O CMMI é uma metodologia de maturidade de processo mais abrangente que combina CMM for software [6] com disciplinas mais amplas nas áreas de engenharia de sistemas e desenvolvimento de produtos, inclui um questionário, o qual é útil no processo de melhoria da qualidade do software, promove a melhoria dos processos das

organizações e a habilidade de gerenciar, desenvolver e manter os seus produtos (software).

3.1.1 Níveis de Maturidade

O CMMI descreve os estágios de maturidade através dos quais organizações passam enquanto evolui o seu ciclo de desenvolvimento de software, através de avaliação contínua, identificação de problemas e ações corretivas dentro de uma estratégia de melhoria dos processos [6].

Este caminho de melhoria é definido por cinco níveis de maturidade. Cada Nível de Maturidade, exceto o Nível 1, possui um conjunto de PA preestabelecido:

1. Realização – Estágio inicial;
2. Gerenciado – Gerenciamento de Requisitos (REQM), Planejamento de Projeto (PP), Monitoramento e Controle de Projeto (PMC), Gerenciamento de Fornecedores (SAM), Medição e Análise (MA), Garantia da Qualidade do Processo e do Produto (PPQA) e Gerenciamento de Configuração (CM);
3. Definido – Desenvolvimento de Requisitos (RD), Solução Técnica (TS), Integração do Produto (PI), Verificação (VER), Validação (VAL), Foco no Processo Organizacional (OPF), Definição do Processo Organizacional (OPD), Treinamento Organizacional (OT), Gerenciamento de Riscos (RSKM), Gerenciamento Integrado do Projeto (IPM) e Análise da Decisão da Resolução (DAR);
4. Quantitativamente – Gerenciamento Quantitativo do Projeto (QPM) e Desempenho do Processo Organizacional (OPP);
5. Otimização – Análise Causal e Resolução (CAR), Inovação Organizacional e Implantação (OID).

3.1.2 Representação Contínua e por Estágio

O CMMI possui duas representações: "contínua" ou "por estágios". Estas representações permitem que a organização utilizem diferentes caminhos para a melhoria de acordo com seu interesse.

A representação contínua agrupa as áreas de processo por categorias de afinidade, com atribuição de níveis de capacidade para a melhoria de processos em cada área de processo. A representação em estágios, por sua vez, organiza as áreas de processo em cinco níveis de maturidade para suportar e guiar a melhoria dos processos. As áreas de processos podem ser agrupadas também em quatro categorias: Gerenciamento de Processos, Gerenciamento de Projetos, Engenharia, e Suporte [7].

Tabela 1 - Categorias de Processos do CMMI.

(Fonte: Autor)

Categorias do Processo CMMI		
Categoria	**Área de Processo**	**Nível de Maturidade**
Gerenciamento de Processos	OPD: Definição do Processo Organizacional	3
	OPF: Foco no Processo Organizacional	3
	OT: Treinamento Organizacional	3
	OPP: Desempenho do Processo Organizacional	4
	OID: Inovação Organizacional e Implantação	5
Gerenciamento de Projetos	PP: Planejamento de Projeto	2
	PMC: Acompanhamento e Controle de Projeto	2
	SAM: Gerência de Acordos com Fornecedores	2
	IPM: Gerência Integrada de Projeto	3
	RSKM: Gerência de Risco	3
	QPM: Gerência Quantitativa de Projeto	4
Engenharia	VAL: Validação	3
	VER: Verificação	3
	PI: Integração de Produtos	3
	TS: Solução Técnica	3
	RD: Desenvolvimento dos Requisitos	3
	REQM: Gerenciamento de Requisitos	2

Categorias do Processo CMMI		
Categoria	**Área de Processo**	**Nível de Maturidade**
Suporte	CM: Gerência de Configuração	2
	PPQA: Garantia da Qualidade de Processo e Produto	2
	MA: Medição e Análise	2
	DAR: Análise de Decisão e Resolução	2
	CAR: Análise Causal e Resolução	5

3.1.3 Áreas Chave de Processo

As Áreas de Processo, ou simplesmente PA como são conhecidas, representam áreas da organização de software, identificadas pelo modelo CMMI, que possuem importância fundamental no estabelecimento de um processo de software de qualidade.

O CMMI está agrupado em vinte e duas áreas de processo. Áreas de processo são um conjunto de práticas relacionadas, que, quando implementadas coletivamente, satisfazem os objetivos considerados importantes para obter melhoramentos naquela área [6]. A cada uma destas áreas de processo estão associadas metas genéricas e específicas, componentes considerados obrigatórios para analisar a satisfação na mesma por uma organização.

As metas genéricas têm este nome porque estão relacionadas a mais de uma área de processo. Elas contêm as características que são necessárias para considerar uma área de processo institucionalizada. Este documento não está relacionado a uma única organização, portanto, as metas genéricas e seus subcomponentes, não serão aqui considerados durante a análise de satisfação do CMMI. No caso de uma implementação real das soluções aqui propostas, cabe a cada organização avaliar os aspectos relacionados à institucionalização dos processos.

Já as metas específicas, por sua vez, descrevem as características únicas daquela área de processo e são necessárias para

considerá-la satisfeita. As metas específicas dividem-se em práticas específicas.

Práticas específicas é a descrição de uma atividade que é considerada importante para atingir determinada meta específica. As práticas específicas de uma meta, quando implementadas conjuntamente, resultam na satisfação da meta específica. Elas são consideradas componentes esperados durante uma avaliação e evidências de satisfação das práticas são coletadas. As práticas específicas dividem-se em subpráticas.

Subpráticas são componentes informativos que têm o objetivo de auxiliar o entendimento da prática a que estão associadas. Elas devem ser consideradas apenas para prover idéias que serão úteis na melhoria dos processos.

Conforme visto no nível 1 de Maturidade, o processo de software da organização é caótico e, portanto, não há como esperar que alguma área esteja bem estabelecida, motivo este que nenhuma PA é abordada, pelo modelo, neste nível de Maturidade.

Não é objetivo do modelo descrever exaustivamente todas as áreas envolvidas no desenvolvimento e manutenção de software. Ele somente trabalha com as áreas que foram consideradas 'chave' para a obtenção de cada um dos níveis de maturidade.

O CMMI apresenta níveis de maturidade da capacitação e além de definir cada um dos níveis, o modelo descreve como será avaliado se uma empresa já está pronta para passar ao nível seguinte. Em cada nível, o ambiente precisa demonstrar que tem plena capacitação em determinadas áreas-chave (Process Area, ou PA). Para passar para o nível 2, todas as PA's relativas a funções gerenciais precisam estar implantadas. Para atingir o nível 3, o nível definido, as PA's referentes a todos os procedimentos técnicos necessários para a execução de atividades técnicas, de treinamento e de revisão interna já devem estar plenamente capacitadas. As PA's correspondentes às formas de medir e avaliar cada procedimento técnico são exigidas para passar para o nível

gerenciado. Finalmente, no nível mais alto, PA's específicas apóiam o processo de melhoria contínua com correção de problemas e evolução de processos.

Observamos que o modelo CMMI descreve cada PA em termos de um propósito e de algumas metas a serem atingidas. Enquanto o propósito (Objetivo) explica qual a função daquela área chave, as metas são exatamente o que será avaliado pelo avaliador oficial. Resumindo, a função do avaliador é verificar o cumprimento das metas de cada PA.

No CMMI, os níveis de maturidade são acumulativos, ou seja, para uma organização ser avaliada nível 3, por exemplo, ela precisa primeiro cumprir todas as metas do nível 2, e assim por diante. Em tese, uma empresa que está sendo avaliada no nível 3, se não cumprir alguma meta do nível 2, automaticamente será avaliada oficialmente nível 1.

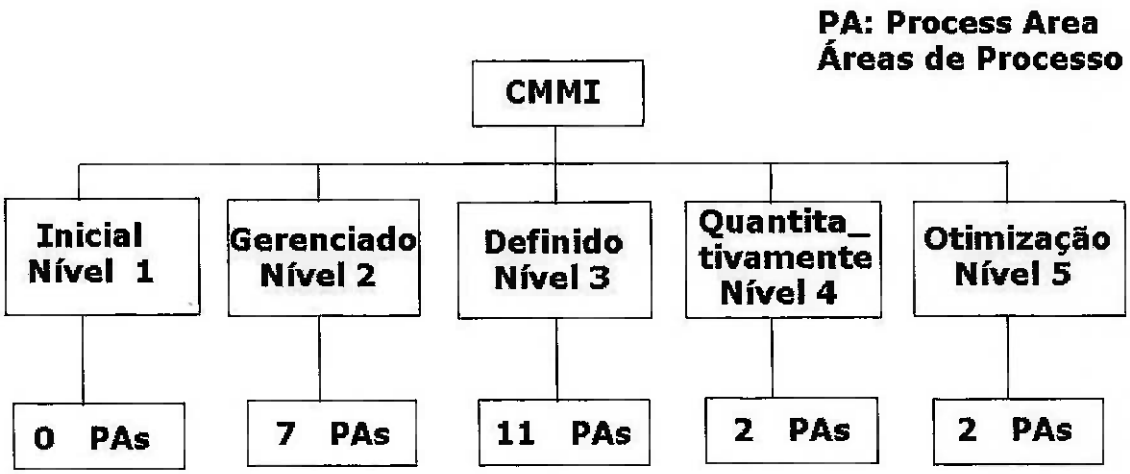

Figura 2 - Números de PA's

(Fonte: autor)

3.1.2.1 Práticas Específicas ou Práticas Genéricas do CMMI

Como já explicado no capítulo do CMMI, os componentes necessários para atender a uma avaliação são as metas, práticas e

subpráticas. Apesar das práticas não serem consideradas obrigatórias para o alcance dos níveis de maturidade e capacidade, as práticas de uma meta são, em geral, o que determinam ou não a satisfação da mesma. Devido à importância das práticas para o entendimento do CMMI, uma análise da satisfação em relação às práticas será realizada.

De acordo com o SEI:

Tabela 2 – Número de práticas por áreas de processo.
(Fonte: autor)

Áreas de Processos	Números de práticas
Planejamento de Projeto (PP)	14
Gerenciamento de Requisitos (REQM)	5
Monitoramento e Controle de Projeto (PMC)	10
Gerenciamento de Acordos de Fornecedores (SAM)	7
Medição e Análise (MA)	8
Garantia da Qualidade do Processo e do Produto (PPQA)	4
Gerenciamento de Configuração (CM)	7
Desenvolvimento de Requisitos (RD)	10
Solução Técnica (TS)	9
Integração do Produto (PI)	9
Verificação (VER)	8
Validação (VAL)	5
Foco no Processo Organizacional (OPF)	7
Definição do Processo Organizacional (OPD)	5
Treinamento Organizacional (OT)	7
Gerenciamento de Riscos (RSKM)	7
Gerenciamento Integrado do Projeto (IPM)	13
Análise da Decisão da Resolução (DAR)	6
Gerenciamento Quantitativo do Projeto (QPM)	8
Performance do Processo Organizacional (OPP)	5
Análise Causal e Resolução (CAR)	5
Inovação Organizacional e Implantação (OID)	7

Tabela 3 - Práticas das PA's da categoria de Gerenciamento de Projeto.
(Fonte: http://www.sei.cmu.edu/cmmi/adoption/pdf/stsc-mappings.pdf)

Planejamento do Projeto (PP)
SG 1 As estimativas de parâmetros do planejamento do projeto são estabelecidas e mantidas.
SP 1.1 Estabelecer uma estrutura alto-nível da avaria do trabalho (WBS) à estimativa do espaço do projeto.
SP 1.2 Estabelecer e manter as estimativas dos atributos dos produtos e das tarefas do trabalho.
SP 1.3 Definir as fases do ciclo de vida do projeto em cima de que ao espaço o esforço do planejamento.
SP 1.4 Estimar o esforço e o custo do projeto para os atributos dos produtos e das tarefas do trabalho baseados na lógica de estimação.
SG 2 Um plano de projeto é estabelecido e mantido como base para a gestão do projeto.
SP 2.1 Estabelecer e manter o orçamento do projetos e programe-o.
SP 2.2 Identificar os Riscos do Projeto
SP 2.3 Planejar o Gerenciamento de Dados
SP 2.4 Planejar recursos necessários para executar o projeto.
SP 2.5 Planejar os conhecimentos e as habilidades necessárias para executar o projeto.
SP 2.6 Planejar a participação das partes identificadas interessadas.
SP 2.7 Estabelecer e manter o plano do projeto.
SG 3 Compromissos do plano do projeto são criados e mantidos.
SP 3.1 Revisar os planos que afetam o projeto para atender todos os compromissos do projeto.
SP 3.2 Equilibrar Níveis de Trabalho e Recursos
SP 3.3 Obter compromisso das partes interessadas responsáveis para executar o plano.
Monitoração e controle do projeto (PMC)
SG 1 O atual desempenho do progresso do projeto são monitorados de encontro com o plano do projeto.
SP 1.1 Monitorar os valores reais dos parâmetros de planejamento do projeto de encontro com o plano do projeto.
SP 1.2 Monitorar os compromissos identificados de encontro com o plano do projeto.
SP 1.3 Monitorar os riscos identificados de encontro com o plano do projeto.
SP 1.4 Monitorar o gerenciamento dos dados do projeto de encontro com o plano do projeto.
SP 1.5 Monitorar a participação da parte interessada (STAKEHOLDERS) de encontro com o plano do projeto.
SP 1.6 Revisar periodicamente o progresso, desempenho, e mudanças do projeto.
SP 1.7 Revisar as realizações e os resultados do projeto em marcos selecionados do projeto.
SG 2 As ações corretivas estão controladas ao fechamento quando o desempenho ou os resultados do projeto desviam significativamente do plano.
SP 2.1 Coletar e analisar as mudanças e determinar as ações corretivas necessárias para dirigir-se às mudanças.
SP 2.2 Tomar ações corretivas nas mudanças identificadas.
SP 2.3 Controlar as ações corretivas ao fechamento.

Gerenciamento Quantitativo de Projeto (QPM)
SG 1 Gerenciar quantitativamente o projeto
SP 1.1 Estabelecer Objetivos do Projeto.
SP 1.2 Compor o Processo Definido.
SP 1.3 Escolher subprocessos que serão gerenciados estatisticamente.
SP 1.4 Gerenciar a desempenho do projeto.
SG 2 Gerenciar estatisticamente a performance dos subprocessos.
SP 2.1 Selecionar as medidas e técnicas analíticas.
SP 2.2 Aplicar métodos estatísticos para entender variação.
SP 2.3 Monitorar desempenho dos subprocessos selecionados.
SP 2.4 Gravar o gerenciamento estatístico dos dados.
Gerenciamento de Riscos (RSKM)
SG 1 Preparar o Gerenciamento dos Riscos.
SP 1.1 Determinar as fontes e categorias do risco.
SP 1.2 Determinar os parâmetros do risco.
SP 1.3 Estabelecer a estratégia de gerenciamento dos riscos.
SG 2 Identificar e Analisar Riscos.
SP 2.1 Identificar Riscos.
SP 2.2 Avaliar, categorizar e priorizar riscos.
SG 3 Mitigar Riscos.
SP 3.1 Desenvolver planos de mitigação de riscos.
SP 3.2 Implementar planos de mitigação de riscos.
Gerenciamento Integrado de Projeto (IPM)
SG 1 Utilizar o Processo Definido do Projeto
SP 1.1 Estabelecer o Processo Definido do Projeto.
SP 1.2 Utilizar os Ativos de Processos Organizacionais para o Planejamento das Atividades do Projeto.
SP 1.3 Estabelecer o Ambiente de Trabalho do Projeto.
SP 1.4 Integrar os Planos.
SP 1.5 Gerenciar o Projeto Utilizando os Planos Integrados.
SP 1.6 Contribuir para os Ativos de Processos Organizacionais.
SG 2 Coordenar e Colaborar com os Stakeholders Relevantes
SP 2.1 Gerenciar o Envolvimento dos Stakeholders.
SP 2.2 Gerenciar Dependências.
SP 2.3 Resolver Questões de Coordenação.
SG 3 Aplicar Princípios do Desenvolvimento Integrado do Produto e do Processo
SP 3.1 Estabelecer a Visão Compartilhada do Projeto.
SP 3.2 Estabelecer uma Estrutura Integrada para o Time.
SP 3.3 Alocar Requisitos para Times Integrados.
SP 3.4 Estabelecer Times Integrados.
SP 3.5 Garantir colaboração entre as interfaces dos times.

Este capítulo apresentou o CMMI, sua estrutura, níveis de maturidade, áreas chave de processo, práticas e características comuns.

3.2 Gerenciamentos Ágeis – Scrum

Scrum é um processo ágil para desenvolver software, é uma guia para a gerência que aceita que o desenvolvimento de software seja imprevisível, sendo aplicável a ambientes voláteis. É usado também para controlar o desenvolvimento complexo de software e de produtos através de práticas iterativas e incrementais, aumentando significativamente a produtividade e reduzindo o tempo [8].

O Scrum proporciona uma maneira de melhorar a comunicações e maximizar a cooperação. É ideal para projetos com mudanças rápidas ou exigências altamente emergentes.

O método baseia-se ainda, em princípios como: equipes pequenas; requisitos que são pouco estáveis ou desconhecidos; e iterações curtas, onde é necessário dividir o desenvolvimento em intervalos de tempos no máximo 30 dias, também chamados de Sprints.

Ele se destaca dos demais métodos ágeis pela maior ênfase dada ao gerenciamento do projeto. Reúne atividades de monitoramento e *feedback*, em geral, reuniões rápidas e diárias com toda a equipe, visando à identificação e correção de quaisquer deficiências e/ou impedimentos no processo de desenvolvimento.

O Scrum assume a premissa de que o desenvolvimento de software é muito complexo e imprevisível para ser planejado por inteiro inicialmente. Ao invés disso, deve-se usar controle do processo empírico para garantir a visibilidade, inspeção e adaptação [9].

3.2.1 Fases do Scrum

O ciclo de vida do Scrum é baseado em três fases principais, divididas em subfases [10],[19]:

- **Pré-planejamento *(Pre-game phase):*** através de reunião é definido e registrado em documento chamado backlog a lista de funcionalidades priorizadas, junto com o cliente, a serem desenvolvidas, onde estabelece a visão do projeto e expectativas garantindo recursos para a sua execução.
- **Desenvolvimento *(game phase):*** durante o desenvolvimento são controladas e monitoradas as realizações da lista de funcionalidades. Assim aumenta a flexibilidade para acompanhar as mudanças. Através dos sprints diários, semanais e mensais novas funcionalidades são adicionadas, onde são analisadas, implementadas e testadas.
- **Pós-planejamento *(post-game phase):*** nesta fase através de reuniões com os clientes é possível demonstrar e analisar o andamento do projeto e assim realizar os testes finais e a documentação.

3.2.1.2 Etapas de cada Fase

No Scrum, um projeto se inicia com uma visão do produto que será desenvolvido, a qual contém a lista das características do produto estabelecidas pelo cliente, além de algumas premissas e restrições. Em seguida, o *Product Backlog* é criado contendo a lista de todos os requisitos conhecidos, onde é priorizado e dividido em *releases*, tarefas, que são executas no prazo de horas ou dias. Após a execução dessas tarefas são listadas e priorizadas novas tarefas, passando por uma retrospectiva, analisando o que deu de errado e/ou certo para que as novas tarefas sejam executadas da maneira mais eficiente.

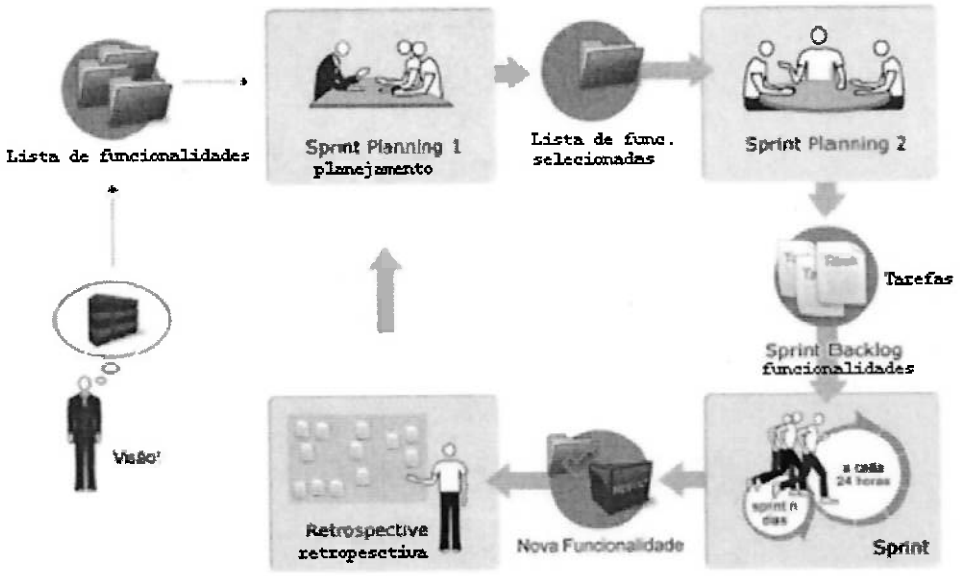

Figura 3 - Processo SCRUM (detalhado).

(Fonte: http://www.glogerconsulting.de/downloads/gloger-6Step-HeartbeatRetrospectives-17.07.06.pdf)

Preparação: Onde se define o bu*siness-case*, *game concept* (conceito do jogo), *Product Backlog* (lista de tarefas) inicial e outras premissas ligadas aos projetos;

Sprints: São iterações realizadas, uma após a outra, para entregar gradativamente as estórias que compõem o jogo. Dentro de cada Sprint são realizadas algumas atividades:

- **Scrum Planning Meeting (reunião de planejamento)**: É o início de um Sprint, onde o *Product Owner (cliente)* tem a oportunidade de atualizar a priorização dos itens do *Product Backlog* e definir juntamente com a equipe um *Product Increment* (módulo/funcionalidade) a ser entregue ao cliente ao final do Sprint;
- **Daily Scrum Meeting (reuniões diárias)**: É um encontro diário realizado pela equipe onde os membros discutem aquilo que trabalharam, no que irão trabalhar e possíveis impedimentos que estejam atrapalhando o progresso do mesmo. Este encontro é uma maneira eficiente de manter os

membros cientes dos objetivos e impedir que o projeto "saia do rumo";

- **Criação do Product Increment**: A finalização das estórias definidas para um determinado Sprint marca a realização de um *Product Increment*. Similarmente, outros membros da equipe trabalham de maneira colaborativa de forma a realizar todas as estórias definidas para aquele Sprint. Além disso, diariamente, os membros da equipe devem atualizar a estimativa de esforço necessário para finalizar cada estória;
- **Sprint Review (revisão):** Neste momento, a equipe exibe o *Product Increment* construído ao *Product Owner*, que é responsável por validar e/ou solicitar ajustes para que o jogo se torne adequado aos anseios do cliente;
- **Sprint Retrospective (retrospectiva):** Tem com objetivo identificar os pontos positivos e negativos do Sprint que entregou o último *Product Increment* e procura corrigir os problemas encontrados.
- **Atualização do Product Backlog:** O *Product Owner* é responsável por repriorizar toda lista de itens do *Product Backlog* para que um próximo Sprint possa ser iniciado motivado pelos itens mais prioritários.

Encerramento: É a última etapa de um projeto utilizando Scrum, ocorre após a finalização de todos Sprints e é marcada pela entrega do jogo final que motivou a criação do projeto.

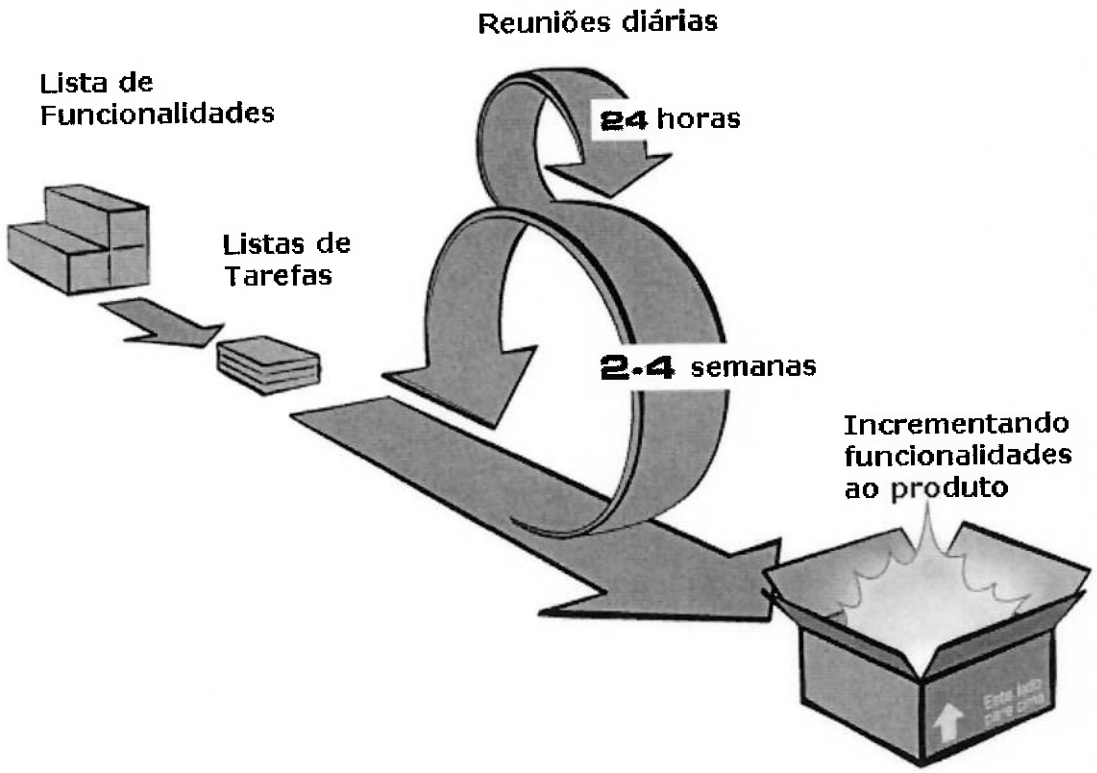

Figura 4 - Processo SCRUM.

(Fonte: http://www.improveit.com.br/images/br/scrum/ciclo_scrum.gif)

3.2.2 Processo Scrum

O Scrum implementa um esqueleto iterativo e incremental através de três papéis principais [11]:

- *Product Owner:* é a pessoa que define os itens que compõem o Product Backlog e os prioriza nas Sprint Planning Meetings.
- *ScrumMaster* : é responsável por assegurar que a equipe respeite e siga os valores, as regras e as práticas do Scrum e também é responsável por remover os impedimentos do projeto
- *Time:* é responsável por escolher as funcionalidades a serem desenvolvidas a cada interação e desenvolvê-las. Todos os membros do time são coletivamente responsáveis pelo sucesso de cada iteração, se auto-gerencia, se auto-organiza.

Os controles na metodologia de SCRUM são [10];[12]:

- Reserva: requisitos que não são tratados adequadamente pela atual liberação do produto, erros, defeitos, melhorias solicitada pelo cliente, funcionalidade, vantagem competitiva funcionalidade, tecnologia e upgrades.
- Liberação/entregas: representam uma viável libertação com base nas variáveis de requisitos de tempo, qualidade e concorrência.
- Pacotes: componentes ou objetos do produto que devem ser mudados para executar um artigo da reserva em uma liberação nova.
- Mudanças: mudança que deve ocorrer em um pacote para executar um item reservado.
- Problemas: problemas técnicos que ocorram e devam ser resolvidos para executar uma mudança.
- Riscos: os riscos que afetam o sucesso do projeto continuamente são avaliados e as respostas são planejadas. Outros controles são afetados em conseqüência da avaliação do risco.
- Soluções: soluções aos problemas e aos riscos, freqüentemente resultando em mudanças.
- Edições: edições do projeto que não são definidas nos termos dos pacotes, das mudanças e dos problemas.

É primordial que se estabeleça claramente onde a organização deseja chegar em termos de melhoria, quais suas reais necessidades estratégicas e adequar da melhor forma possível um modelo de gestão que traga benefícios ao seu negócio. O CMMI e/ou Scrum deve ser adaptado à realidade de cada organização, reduzindo-se assim custos e esforços de implementação.

4. ESTUDO COMPARATIVO ENTRE OS MÉTODOS DE GERENCIAMENTO

Devido à importância das práticas para entendimento do CMMI e alcance das metas, um diagnóstico de satisfação será realizado contra algumas práticas do nível 2 e nível 3, que compõem a categoria de Gerenciamento de Projetos, pois se as práticas forem satisfeitas, conseqüentemente as metas também serão. Este comparativo tem como referência o estudo sobre o Mapeamento das práticas de SCRUM nas áreas de processo do CMMI e uma análise de sua aderência [13].

Este diagnóstico irá focar apenas nas áreas-chave de processo: Gerenciamento quantitativo do projeto (QPM), Gerenciamento de riscos (RSKM), Gerenciamento Integrado do projeto (IPM), Monitoramento e controle de projeto (PMC), Planejamento de Projeto (PP).

4.1 Metodologia para realização do diagnóstico

Para realizar o diagnóstico a seguinte abordagem foi utilizada: cada prática de implementação do CMMI na categoria de Gerenciamento de Projetos foi analisada e uma pontuação que lhe foi atribuída, a qual representa a satisfação da prática pela metodologia Scrum e XP. Os valores desta pontuação foram realizados para atender ao objetivo deste diagnóstico, cujos significados são:

- Satisfeita: se a prática foi completamente satisfeita pelo Scrum.
- Não satisfeita: se há pontos fracos significativos associados à prática comprometendo a sua implementação.
- Parcialmente satisfeito: Não há evidências da satisfação da prática por Scrum.
- Não aplicável: se o componente não se aplica ao escopo deste trabalho.

Tabela 4 - Mapeamento das práticas de SCRUM nas áreas de processo do CMMI.
(Fonte: adaptado da monografia: Mapeamento das práticas de Scrum nas áreas de processo do CMMI e uma proposta para sua aderência [13])

Áreas de Processo	*Diagnóstico*			
	Satisfeito	*Não Satisfeito*	*Parcialmente satisfeito*	*Não aplicável*
Planejamento de Projeto (PP)				
Determinar os recursos necessários; Estimativas de esforço e custo; Negociar compromissos; Identificar e analisar os riscos; Produzir um cronograma e Orçamento.			X	
Monitoramento e controle do projeto (PMC)				
Monitorar atividades, comunicar status e tomar as ações corretivas, monitorar os riscos			X	
Gerenciamento de Acordos com Fornecedores (SAM)				
Aceitação do produto adquirido, identificação das necessidades de aquisição				X
Gerenciamento Quantitativo de Projeto (QPM)				
Gerenciar quantitativamente o processo		X		
Gerenciamento de Riscos (RSKM)				
Identificação, análise dos riscos, priorização e classificação, mitigação e contingência, monitoramento e ações			X	
Gerenciamento Integrado de Projeto (IPM)				
Estabelecer e gerenciar o projeto e o envolvimento dos stakeholders	X			

- O objetivo do Planejamento do Projeto (PP) é estabelecer e manter planos que definam as atividades do projeto, mas o Scrum não prevê a produção de orçamento conseguindo atender apenas parcialmente está prática.
- O objetivo do Monitoramento e Controle do Projeto (PMC) é oferecer um entendimento do progresso do projeto, de maneira que as ações corretivas apropriadas possam ser tomadas quando o desempenho do projeto desviar significativamente do plano, pois o Scrum atende parcialmente onde apenas identifica os riscos e não os monitora.

- O objetivo do Gerenciamento de Acordos com Fornecedores (SAM) é a aceitação do produto adquirido e identificação das necessidades de aquisição, os quais não são aplicados pelo Scrum.
- O objetivo do Gerenciamento Quantitativo de Projeto (QPM) é gerenciar quantitativamente o processo, porém não existe nenhuma prática do Scrum que satisfaça esta PA e que consiga atingir os objetivos de qualidade e desempenho.
- O objetivo do Gerenciamento de Riscos (RSKM) é a identificação, análise dos riscos, priorização / classificação, contingência e monitoramento / ações. O Scrum não possui a prática de monitoramento, somente a identificação do risco.
- O objetivo do Gerenciamento Integrado de Projeto (IPM) é estabelecer e gerenciar o projeto e o envolvimento dos *stakeholders*. Nesta área de processo existem práticas no Scrum que satisfazem completamente esta PA.

5. ESTUDO DE CASO

O estudo de caso representa o resultado de um projeto de desenvolvimento de software que utilizou os valores e práticas do Extreme Programming e Scrum durante um período de seis meses.

5.1 Unidade de Análise

O Departamento de Informática (DI) da Universidade de São Paulo (USP) é um órgão ligado à Coordenadoria de Administração Geral (CODAGE), criado em 1993 para promover uma importante modernização da informática administrativa na USP. O DI é responsável pela criação e manutenção de softwares para agilizar o dia-a-dia dos alunos, funcionários e professores, tais como os sistemas Júpiter, que cuida da vida acadêmica dos alunos da graduação, o Fênix, para a pós-graduação, o Marte, que controla a folha de pagamento, o Mercúrio, para a requisição de compras e pedidos no almoxarifado, e o Proteos, que acompanha o andamento dos protocolos. Todos esses sistemas são integrados.

A Universidade de São Paulo possui atualmente - 2007 - uma frota de aproximadamente 800 veículos. Esses veículos estão distribuídos entre as faculdades e instituições da USP, de acordo com o seu tamanho e necessidades.

Existem diversas funcionalidades e práticas que se operam sobre estes patrimônios da Universidade, como: abastecimento, manutenção e reparo, solicitação de ônibus para viagens, carros para transporte de passageiros, controle de condutores e controle de tráfego.

O controle dessas funcionalidades era realizado através de papéis e de softwares desagregados, que não compartilhavam informações. As práticas, além de terem alta carga de trabalho, duplicação de tarefas, se mostravam ineficazes e ineficientes.

Por essas razões, entre diversas outras, a USP decidiu investir na aquisição de um software de Gestão de Frotas, que atendesse às suas demandas, mas como os softwares apresentados pelo mercado não foram completamente aceitos, resolveu-se que o desenvolvimento do software seria da própria Universidade, sendo a responsabilidade do desenvolvimento repassada ao Departamento de Informática da USP.

5.2 Necessidade do Desenvolvimento Ágil

O DI inicia o planejamento do projeto entendendo o que deve ser feito após o cliente informar a necessidade de um novo aplicativo. São registrados todos os requisitos funcionais ou não funcionais que serão necessários para a construção deste aplicativo ou parte do aplicativo (módulo).

O departamento efetua uma análise sobre o que deve ser feito e faz a estimativa de tempo para cada item apresentado no documento.

Após a conclusão de cada módulo ou funcionalidade inicia-se a fase de teste pelo próprio desenvolvedor, que efetua os testes unitários e os de comportamento do sistema e só após a validação desta área é que o módulo ou funcionalidade é liberado em forma beta para o usuário testar. Isso depende muito da necessidade do cliente, liberar antes do termino de todo o aplicativo ou liberando progressivamente. Após a validação do cliente, o aplicativo é colocado em produção.

O sistema de Gestão de Frota começou a ser desenvolvido em fevereiro de 2006, após um grande período de documentação, um ano. E devido à necessidade dos gestores da universidade em possuírem informações sobre uma grande fatia do orçamento da USP.

Seguindo a metodologia atual o departamento poderia demorar mais para apresentar um módulo ou uma versão do sistema, o qual prejudicaria a imagem do departamento e também a necessidade dos

gestores em visualizar os gastos com a frota de veículos da USP para futuros investimentos.

Neste sentido, o departamento decidiu investir nas práticas do Extreme Programming juntamente com o gerenciamento Scrum, pois precisava apresentar resultados com rapidez e o mesmo estava quase todo envolvido em outras tarefas de grande importância a universidade.

O sistema está sendo implementado por uma equipe interna do DI, composta de quatro pessoas, mais o coordenador do projeto. As práticas originais do Extreme Programming e gerenciamento Scrum, foram sendo implantadas no primeiro e início do segundo mês de desenvolvimento, à medida que as necessidades iam surgindo.

5.3 Metodologia de Desenvolvimento

Papéis em uma equipe XP não são fixos e rígidos. O objetivo é fazer com que cada um contribua com o melhor que tem a oferecer para que a equipe tenha sucesso.

Todos os contribuintes do projeto sentavam-se juntos como membros de um time. Através de reuniões com usuários e clientes, a equipe definia os requisitos, fixava as prioridades e guiava o projeto. O time possuía quatro programadores, os quais também absorviam a tarefa de ajudar o cliente a definir os testes de aceitação e os requisitos. Além destes, existia um gerente que provia recursos, mantinha a comunicação externa e coordenava as atividades. Nenhum destes papéis foram necessariamente de propriedade exclusiva de um só indivíduo, todos os membros do time contribuíram de todas as formas que puderam, conforme suas capacidades.

O ambiente de trabalho de uma equipe XP deve ser um reflexo do projeto. Alguém que entre na sala da equipe deve conseguir obter, em poucos segundos, uma noção clara de como está o andamento do projeto.

Um dos primeiros passos na implantação do XP e talvez a mais visível de todas as práticas foi aquisição de um quadro de avisos. Desde sua implantação, já chamou a atenção das outras equipes do departamento. Neste quadro são colocados os cartões em papel com as tarefas, de modo que possam ser acompanhados facilmente por todos os participantes do projeto, incluindo desenvolvedores, gerentes, clientes, entre outros. Existem softwares que procuram ajudar no planejamento de projetos XP, entretanto eles não são tão eficazes quanto os cartões em papel com as tarefas.

Esses cartões são criados após programadores e clientes priorizar os requerimentos e dividi-los em tarefas, assim cada tarefa se transforma em um cartão. Mesmo após definido, os programadores poderão "quebrar" o cartão de tarefa em mais cartões, comunicando ao cliente. Este caso ocorre quando uma tarefa não foi estimada corretamente e o tempo excede o limite máximo estabelecido para cada cartão.

Com o intuito de coordenar as atividades, estabelecer uma harmonização das tarefas e tornar evidentes os objetivos do período, o quadro foi dividido em quatro partes:

- Disponíveis: tarefas a serem escolhidas e realizadas por alguém disponível;
- Atribuídas: tarefas escolhidas por um membro e que estão em andamento;
- Concluídas: tarefas completadas;
- Pendentes: tarefas que necessitam de outras pessoas para serem iniciadas ou que ainda deverão ser discutidas.

Posteriormente, devido à necessidade de membros da equipe que realizavam tarefas alheias ao projeto, surgiu mais uma divisão no quadro: Externas.

Foram preenchidos pequenos cartões para serem anexados nessas áreas. Cada cartão continha o nome da tarefa, uma descrição e

a quantidade de tempo que se presumia levar para o término da mesma. A escolha de quais seriam as tarefas e o tempo geralmente eram definidos semanalmente após reuniões.

Partindo-se dos princípios de que nenhuma tarefa leva menos de duas horas para ser completada, já que podem surgir dificuldades no caminho, e de que uma tarefa que leve mais de dois dias para ser concluída deve ser dividida em outras menores, a equipe adotou as seguintes metáforas:

- uma esfera vazia equivale à duas horas;
- uma esfera meia-lua equivale à quatro horas;
- uma esfera cheia equivale à um dia;
- duas esferas cheias equivalem à dois dias.

A esfera é a representação das horas gastas com a execução de cada tarefa. A escolha desta representação foi devido à perda de tempo com a marcação exata de horas e minutos, pois cada programador necessitaria cronometrar o tempo, e assim, sairíamos de nosso foco que é o desenvolvimento. A marcação com esfera se tornou prática no segundo dia de desenvolvimento.

Os cartões com as tarefas eram anexados como disponíveis, de acordo com a sua ordem de prioridade. O membro da equipe escolhe uma tarefa e atribuí ao seu nome, dessa forma é possível visualizar rapidamente quem está ocupado e com o que se está trabalhando no momento. Isso também evita que mais de uma pessoa esteja trabalhando sobre uma mesma tarefa. O desenvolvedor escolhe a tarefa com a qual possui maior familiaridade e que mais lhe agrade, conseqüentemente há uma melhora significativa na produtividade. Conseguindo realizar tarefas que levariam aproximadamente quatro semanas, passaram a serem realizadas em três semanas.

Ao término da tarefa, é atribuída ao cartão uma data, o nome de quem a realizou e a quantidade de esferas (tempo) que ficou com o mesmo, o qual é movido para a seção dos concluídos.

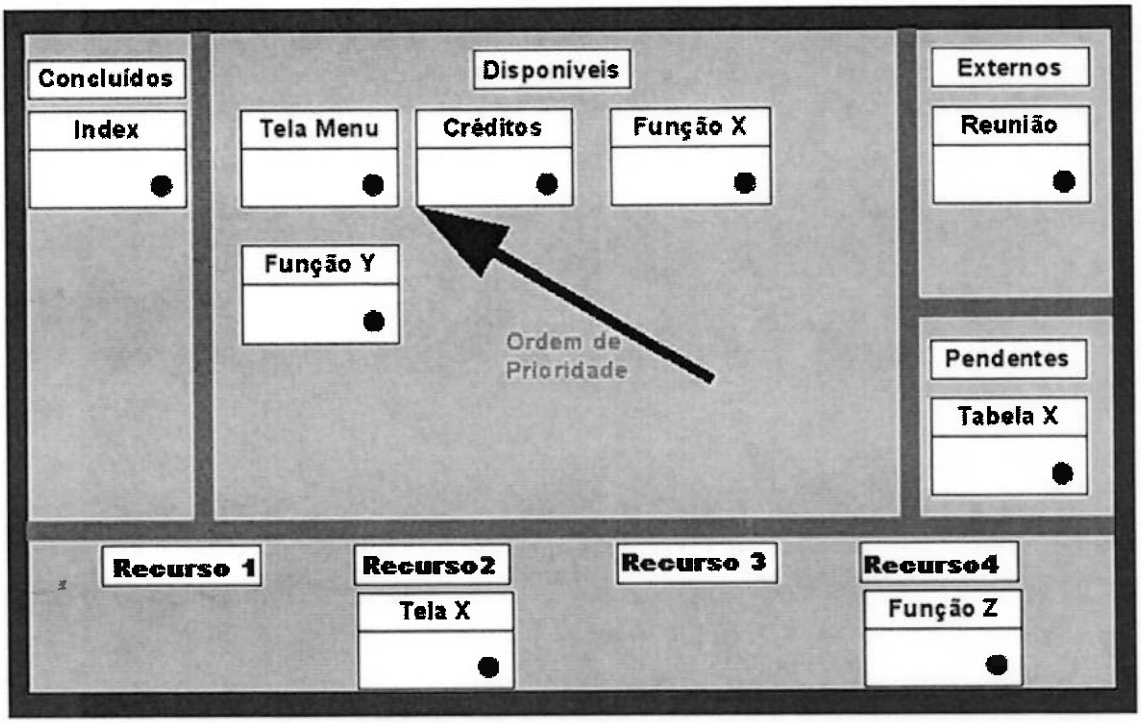

Figura 5 - Quadro de Atividades Semanal
(Fonte: Departamento de Informática da Reitoria da USP)

Nas equipes que não utilizam XP no departamento fica difícil à identificação do integrante que não está contribuindo, podendo até atrapalhar o desenvolvimento.

Já as práticas do XP e o método de gerenciamento SCRUM, como a programação em pares, os cartões concluídos com o nome do responsável pela tarefa e o gráfico semanal de desempenho, faz com que o integrante que se diz com falta de conhecimento na tecnologia, que não cumpre o horário estabelecido de forma indireta seja destacado dos outros integrantes da equipe, fazendo que o mesmo que não contribuiu melhore seu desempenho.

5.3.1 Programação em Pares

Essa prática do Extreme Programming, no início do projeto, foi utilizada somente em algumas tarefas ou em situações especiais, devido ao fato da equipe ter aprendido a trabalhar com as diversas ferramentas.

Somente com o decorrer do processo, tornou-se necessário que aqueles mais experientes nos métodos se sentassem com aquele que necessitassem de auxílio. Dessa forma, além da transferência de conhecimento para futuras tarefas, houve um aumento da produtividade da equipe como um todo e diminuição do retrabalho. Verificamos que com a programação em pares em cada dez tarefas, o retrabalho foi reduzido de cinco para um.

5.3.2 Cliente Presente

O cliente esteve presente durante toda a fase de implementação e implantação do sistema. Isso se deu através de sucessivas reuniões, às vezes semanais, conforme surgiam necessidades de correções, dúvidas ou para definir as novas funcionalidades a serem produzidas.

Esse contato permanente com cliente do sistema foi de grande valia, pois possibilitou um melhor encaminhamento do projeto, prevenindo erros de requisitos e adaptando-se a interface conforme os critérios definidos por aqueles que antes trabalhavam com métodos não informatizados.

À medida que o sistema era desenvolvido, testado e implantado, o cliente expunha suas novas carências, sugeria alterações e relatava os possíveis erros do sistema.

Apesar do cliente não conseguir expressar a totalidade de suas reais necessidades, verificamos que com sua presença em cada dez tarefas, o retrabalho foi reduzido de quatro para um.

5.3.3 Desenvolvimento Orientado a Testes

Esta prática não foi realizada conforme o XP descreve e sim parcialmente, pois os testes eram realizados após o desenvolvimento das tarefas. À medida que as telas do sistema eram concluídas com as

suas devidas funcionalidades, o desenvolvedor testava todas as suas funções. Seu trabalho era repassado ao *coach,* uns dos integrantes da equipe que além de líder da equipe, também era uns dos programadores, que então colocava a função no ambiente de testes.

O desenvolvedor testava novamente os métodos e só depois de tudo verificado a nova página com sua funcionalidade era disponibilizada para o cliente. Isso preveniu erros e diminuiu a carga de trabalho gerada quando o cliente detectou erros no sistema.

Testes é a prática com grande mudança no desenvolvimento de software e por ser um projeto pioneiro no departamento e a falta de experiência nesta metodologia fez com esta prática não fosse implantada da maneira como foi descrita.

5.3.4 Código Coletivo

A prática de código coletivo foi usada durante todo o projeto, mas de forma diferente, pois não possuíamos as práticas do "Testes" antes da codificação. Porém tornar o código coletivo permitiu que a equipe avançasse com agilidade, pois não era necessário esperar por outros desenvolvedores sempre que havia a necessidade de editar um arquivo do sistema. Além disso, o revezamento entre as diversas funcionalidades permitiu que os desenvolvedores obtivessem vivência em todos os aspectos do projeto, porém houve uma centralização da responsabilidade de implantar o código. Após a implementação, os membros da equipe transmitiam seus códigos para o *coach,* que os agregava com o existente e disponibilizava-os ao sistema.

5.3.5 Código Padronizado

Nos primeiros dias do desenvolvimento a equipe estabeleceu um padrão para o código do sistema. Isso foi muito útil para as práticas de código coletivo e da programação em pares, ambas ajudaram a assegurar que os desenvolvedores aderissem aos padrões.

Uma parte da padronização foi baseada em um documento do Departamento de Informática que regula o uso de siglas em sistemas, como a criação de banco de dados. Esse documento contém uma lista de palavras e suas respectivas siglas, com três letras. Como muitas palavras, eram específicas e não estavam presentes na documentação, a equipe também passou a criar siglas padronizadas para essas palavras. Entre outras utilizações dessas siglas, podem-se citar o caso de nomeação de campos de tabelas e a nomenclatura das diversas variáveis presentes no sistema. Os nomes das páginas web também foram padronizados.

Códigos semelhantes, como consultas ao banco de dados e scripts, foram agregados em arquivos próprios para o grupo desses.

5.3.6 Design Incremental ou Design Simples

A cada renovação, a equipe de desenvolvimento implementava novas características na arquitetura do sistema que fossem suficientes para comportar apenas as funcionalidades da criação. Ou seja, mesmo que cartões de iterações futuras fossem conhecidos, a equipe não criava mecanismos para sustentar a construção futura dos mesmos.

A equipe se focava basicamente nas tarefas que estavam sendo implementadas, não se preocupando com futuras funcionalidades. Isso se mostrou vital a partir do momento que novas necessidades do cliente iam surgindo enquanto outras previstas se mostraram não mais necessárias. Outro ponto é que o desvio de atenção ou uma

implementação mais complexa poderia gerar tarefas e funções desnecessárias ao cliente.

5.3.7 Metáforas

Além das esferas que representam as horas que cada tarefa, também um dos principais usos desse instrumento se deu na criação de figuras para os ícones do projeto. Foram utilizadas figuras como peças, pneus e ferramentas nas funcionalidades de ordem de serviços para reparos em oficinas, outro ponto foi a transposição do processo de preenchimento do papel do controle de viagem para o formato digital.

5.3.8 Ritmo Sustentável e Trabalho Energizado

A metodologia XP prega que o mais importante não é trabalhar em excesso e sim trabalhar de forma inteligente. A equipe de desenvolvimento trabalhou das 8h00 às 19h00, devido à diferença entre turnos dos membros, contudo o que se mostrou primordial foi à força, inteligência e sustentabilidade que os membros desempenharam no decorrer do processo. Métodos como o do quadro de avisos ajudou a focar o trabalho no necessário. Com um ritmo de produção constante o projeto foi se encaminhando, porém houve ocasiões em que foi necessário estender a jornada de trabalho de alguns membros, devido às datas de entregas curtas de partes do sistema.

5.3.9 Contrato de Escopo Negociável

Esse princípio se mostrou de grande valia no decorrer do projeto. Tradicionalmente é feito um documento levantando os requisitos e cases do software a ser desenvolvido, sendo que com o XP esse escopo não é fixo. Assim o cliente pode fazer ajustes, os desenvolvedores têm a

liberdade para questionar funcionalidades e propor o que é mais prioritário.

O escopo negociável teve facilidade de ser implantado devido às condições do projeto e por não se tratar de um software pago, ou destinado à venda, mas sim uma solicitação de um departamento para outro em uma instituição pública, sem um custo e escopo previsível.

Na tabela a seguir apresentamos um resumo das práticas do XP adotadas e não adotadas no DI.

Tabela 5 - Práticas do XP- implantadas e Não implantadas.
(Fonte: autor)

Extreme Programming		
Práticas	**Implantadas**	**Não Implantadas ou parcial**
1. Equipe	X	
2. Jogo do planejamento	X	
3. Programação em pares	X	
4. Pequenas versões	X	
5. Metáforas		X
6. Projeto simples	X	
7. Testes		X
8. Reengenharia de software		X
9. Integração contínua	X	
10. Propriedade coletiva do código		X
11. Cliente no local	X	
12. Semana de 40 horas	X	
13. Padrão de codificação	X	

5.4 Gerenciamento Ágil

O software, Gestão de Frota, até a produção desse estudo de caso, não estava completamente concluído e continua a passar por manutenção e acréscimo de funções, conforme o surgimento de novos requisitos. Porém o método ágil utilizado propiciou a conclusão de cada etapa no prazo estipulado cumprindo o cronograma.

5.4.1 Preparação

As funcionalidades e módulos do sistema foram definidos para serem entregues até a próxima iteração.

5.4.2 Sprints

As iterações combinadas na preparação inicial foram semanais.

Houve uma reunião inicial com a priorização das funcionalidades pelo cliente para ser entregue em cada iteração.

5.4.2.1 Scrum Planning Meeting

Dentro das funcionalidades priorizadas pelo cliente a equipe se reuniu e atualizou a prioridade conforme conhecimento de cada integrante, pois ocorria às vezes que o integrante estava engajado em terminar certa tarefa e não podia fazer par com outro integrante e desta forma com o acordo de todos podia-se "pular" a ordem estabelecida e "pegar" o cartão que o integrante conseguisse resolver sozinho, sem comprometer o resultado final.

5.4.2.2 Daily Scrum Meeting

Reuniões diárias de apenas quinze minutos aconteciam para que o trabalho não saísse do rumo. Nestas reuniões era feita uma retrospectiva do acontecido até o momento como: dificuldades, problemas ou soluções que possam ajudar nas futuras tarefas.

Muitos problemas foram resolvidos nessas reuniões e também foram importantes para que a equipe pudesse ter uma melhor noção do todo, através do relato de seus colegas.

5.4.2.3 Criação do Product Increment

Cada tarefa concluída era registrada no quadro (figura 5) para que pudesse em outra reunião diária estimar novamente com mais precisão os outros cartões. Também a tarefa concluída aguardava a validação do cliente.

5.4.2.4 Sprint Review

Esta fase era usada de forma diferente, a cada cartão concluído o *coach* verificava e validava a tarefa como finalizada. E se após a conclusão de algumas tarefas o *coach* tinha conseguido construir um módulo ou uma funcionalidade do sistema, daí sim era requisitado ao cliente à validação ou ajustes ao produto.

5.4.2.5 Sprint Retrospective

Após a validação do cartão (tarefa) pelo cliente, a equipe se reunia para comentar sobre os problemas encontrados e quais as soluções mais viáveis para cada caso especificamente.

5.4.2.6 Atualização do Product Backlog

A cada fim de iterações eram re-priorizado as tarefas com o cliente.

5.4.3 Encerramento

A cada término do sprint, o cliente era solicitado para validar o escopo semanal e propor diretrizes para o próximo.

Com os dados da soma semanal de esferas que estavam disponíveis, as quais demonstravam a produção prevista e a realizada, gerou-se um gráfico, o qual facilitará a equipe a estimar com maior precisão os próximos sprints semanais.

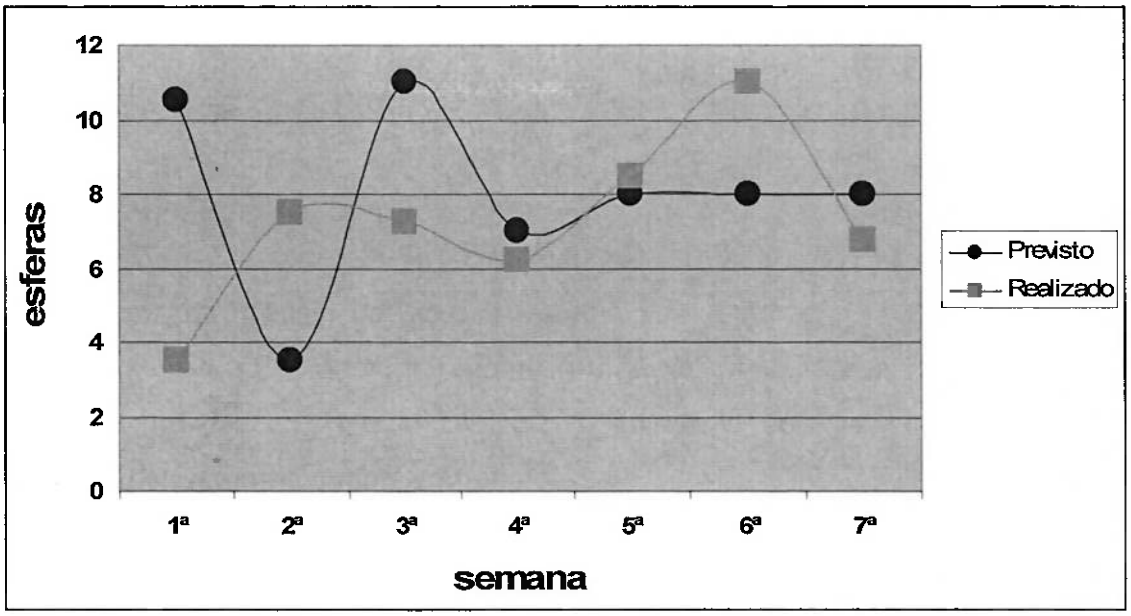

Figura 6 – Gráfico de Desempenho da Equipe.
(Fonte: Departamento de Informática da Reitoria da USP)

O gráfico acima representa a quantidade de esferas concluídas por semana. Neste gráfico foram medidas sete semanas de trabalho, onde as esferas previstas são representadas por uma bola cheia e as esferas realizadas são representadas por um quadro cheio.

Passado algum tempo, a equipe consegue realizar a quantidade de esferas (tempo total) para a realização das tarefas programadas, para a semana, mais próxima do prazo previsto.

O gráfico proporciona uma melhor visualização do desenvolvimento do projeto, possibilitando ao *coach* estabelecer com exatidão o planejamento das tarefas, gerando previsibilidade e controle sobre a produtividade da equipe, geralmente exigido pelos gerentes e clientes, definindo-se o prazo de entrega, sendo este um dos principais problemas do desenvolvimento de software.

5.5 Comparação entre Scrum e CMMI

Apesar de não utilizar a CMMI, algumas áreas de processo da categoria Gerenciamento de Projetos do mesmo foram atendidas parcial ou totalmente, pela equipe no dia-a-dia.

5.5.1 Planejamento de Projeto (PP) versus Scrum

O escopo definido para cada sprint é planejado, preparado e desenvolvido para ser entregue no final do mesmo.

No desenvolvimento do software estimávamos o tempo e o cronograma, após isso alocávamos a equipe e recursos de infra-estrutura.

O conhecimento da equipe em uma determinada ferramenta e o envolvimento do cliente foi decisivo para a continuidade do projeto.

Durante os sprints semanais o projeto era revisado, havendo a necessidade de remoção dos integrantes, formando novos pares com conhecimento específicos necessários, conforme a prioridade de cada tarefa. Isso gerou um equilíbrio e um aumento no compromisso da equipe com o projeto, melhorando o desempenho da mesma.

Através das atividades de planejamento do Scrum é possível atender parcialmente as práticas genéricas da PA Planejamento de Projeto do CMMI.

Tabela 6 - Comparação PP x Scrum.
(Fonte: autor)

SG 1 As estimativas de parâmetros do planejamento do projeto são estabelecidas e mantidas.	Parcialmente satisfeito, pois não estima atributos dos produtos e nem custos do produto.
SG 2 Um plano de projeto é estabelecido e mantido como base para a gestão do projeto.	Parcialmente satisfeito, pois não contempla o orçamento, não monitora os riscos e não gerencia a coleta de dados.
SG 3 Compromissos do plano do projeto são criados e mantidos.	Satisfeito por completo.

5.5.2 Monitoração e Controle do Projeto (PMC) versus Scrum

O compromisso era sustentado através das reuniões diárias (em pé) onde os riscos, problemas e dificuldades encontrados eram ressaltados, para que os outros integrantes pudessem ajudar na solução ou registrá-los nos cartões de pendência. No caso do surgimento de impedimento (pendência) o *coach* era responsável em identificar o problema e solucioná-lo.

Todos os envolvidos tinham uma visão geral do andamento do projeto, possibilitando um melhor monitoramento do mesmo, conseguindo com isso dar rumo ao projeto.

Os sprints semanais foram à forma de monitorar o cumprimento dos compromissos para cada iteração.

A cada iteração o cliente ajudava na priorização e no escopo de cada sprint e o envolvimento quando necessário formando pares.

Todos os cartões concluídos eram registrados em uma planilha, gerando-se um gráfico de desempenho da equipe e garantindo o progresso na entrega do escopo.

A PA, Monitoração e Controle do Projeto do CMMI foram atendidos parcialmente pelo Scrum.

Tabela 7 - Comparação PMC x Scrum
(Fonte: autor)

SG 1 O atual desempenho do progresso do projeto são monitorados de encontro com o plano do projeto.	Parcialmente satisfeito, pois os riscos não são monitorados.
SG 2 As ações corretivas estão controladas ao fechamento quando o desempenho ou os resultados do projeto desviam significativamente do plano.	Parcialmente satisfeito, pois não há planejamento e monitoramento dos impedimentos.

5.5.3 Gerenciamento de Riscos (RSKM) versus Scrum

A equipe apenas identificava os riscos nas reuniões diárias em pé ou nos sprints semanais.

Nesta PA, o Scrum atende apenas uma prática.

Tabela 8 - Comparação RSKM x Scrum
(Fonte: autor)

SG 1 Preparar o Gerenciamento dos Riscos.	Não é satisfeito pelo Scrum.
SG 2 Identificar e Analisar Riscos.	Parcialmente, pois somente identifica o risco.
SG 3 Mitigar Riscos.	Não é satisfeito pelo Scrum.

5.5.4 Gerenciamento Integrado de Projeto (IPM) versus Scrum

O coordenador da equipe gerenciava o envolvimento dos clientes, monitora as pendências através dos cartões, os quais eram afixados no quadro e resolvidos pelo mesmo.

O acesso à documentação do sistema, a infra-estrutura básica garantida pelo DI, mais a ajuda da prática de programação pareada do XP permitiram que a coordenação e o *coach* conseguissem a coesão da equipe.

Somente a prática genérica, Utilizar o Processo Definido do Projeto, do Gerenciamento Integrado de Projeto, não é atendido pelo Scrum.

Tabela 9 - Comparação IPM x Scrum
(Fonte: autor)

SG 1 Utilizar o Processo Definido do Projeto	Não é satisfeito pelo Scrum.
SG 2 Coordenar e Colaborar com os Stakeholders Relevantes	Parcialmente satisfeito pelo Scrum, pois não gerência as dependências.
SG 3 Aplicar Princípios do Desenvolvimento Integrado do Produto e do Processo	Satisfeito por completo.

O uso do método de gerenciamento ágil Scrum, proporcionou maior rapidez no processo de desenvolvimento. O mesmo não cobre diretamente todas as práticas específicas de gerenciamento de projeto do CMMI, mas pode conviver numa mesma organização.

6. CONCLUSÃO

6.1 Considerações Finais

O estudo de caso permitiu verificar que quando há a possibilidade da participação ativa do cliente no desenvolvimento de projetos de software a aplicação dos métodos ágeis torna-se viável no desenvolvimento e no gerenciamento dos projetos. A aplicação do XP e Scrum no DI trouxe os seguintes ganhos:

- Agilidade no desenvolvimento do projeto, pois a equipe conseguiu focar-se em atividades que estavam expostas no quadro (figura 5) evitando a interferência de pessoas estranhas às tarefas priorizadas, reduzindo aproximadamente em 25% o tempo para a conclusão do projeto.
- Diminuição do retrabalho em média de 77,5% devido á interação constante com o cliente e a programação em pares, porém às vezes o cliente não consegue expressar suas reais necessidades e na programação em pares existe a possibilidade de não conseguirem visualizar um possível erro.
- Aumento de produtividade, onde a estrutura do desenvolvimento criado pelo método ágil procura ajudar o projeto a explorar o que os integrantes têm de melhor e solucionar as possíveis falhas com rapidez e segurança. Também procura agir continuamente com priorização para evitar que trabalhos desnecessários sejam executados, isso ajuda a poupar tempo, recursos e permite gerar maior valor para os clientes.
- Garantia da qualidade, onde com as práticas do XP e Scrum foram possíveis dar um rumo a qualidade do software através das reuniões (sprints), das identificações dos problemas e soluções, da reengenharia, do padrão de codificação e principalmente da programação em pares junto com o cliente, além de possibilitar a troca de experiência e conhecimento mútuo.

No estudo de caso observou-se também, que projetos que usam no seu desenvolvimento métodos ágeis também podem ser controlados e monitorados. Através dos gráficos semanais o *coach* conseguia aferir a produtividade com uma previsão aproximando à exatidão, podendo cumprir a data de entrega estipulada de cada escopo.

Através do Scrum e XP foram aplicadas técnicas de planejamento, acompanhamento, gestão de riscos, gestão de integração do projeto as que atendem parcialmente o nível 2 de maturidade do CMMI e que por outro lado, para empresas que buscam níveis de maturidade maiores, o Scrum pode ser uma excelente "largada".

O quadro (figura 5) e o gráfico (figura 6) proporcionaram uma previsibilidade, transparência. As práticas do Scrum e XP trouxeram uma melhor integração da equipe, maiores facilidade na solução de problemas e redução do tempo gasto no desenvolvimento do software, principalmente através das reuniões e programação em pares.

Os resultados apontam que o desenvolvimento e gerenciamento de projetos de software através de métodos ágeis podem se tornar uma tendência nas outras equipes de desenvolvimento dentro do Departamento de Informática da USP, isto devido principalmente à forte participação do cliente, que no caso são clientes internos que pertencem às outras áreas da Universidade.

Trabalhar com desenvolvimento ágil equivale a encarar o processo de uma forma diferente daquela a que estamos habituados. Mais importante que trabalhar muito e produzir muito, é desenvolver o software, de maneira consistente, segura e rápida ao longo de todo o andamento do projeto, assegurando ao cliente um retorno do investimento e também o máximo aproveitamento do tempo de trabalho da equipe de desenvolvimento.

6.2 Trabalhos Futuros

Neste trabalho identificou-se a necessidade de uma ferramenta que auxilie na captação dos dados e permita uma visão gerencial sobre

os projetos de desenvolvimento de software, mostrando transparência aos mesmos que estão em andamento e previsibilidade para outros projetos.

Neste sentido sugerimos como trabalho futuro o estudo e desenvolvimento de um repositório onde ficarão registrados os dados do planejamento e as tarefas realizadas. Com isso será fácil declarar a transparência e fazer com que a equipe tente sempre focar em trabalhos planejados.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. **Manifesto for agile software development.** Disponível em: www.agilemanifesto.org, 2001. Acessado em: 26/01/2008.

[2] Beck, Kent. **Programação Extrema Explicada – Acolha as Mudanças,** Editora BOOKMAN COMPANHIA ED, 2004.

[3] Jeffries, Ron - **What is XP? -** Disponível em http://www.xprogramming.com, visitado em 17/10/2007.

[4] WELLS, D., disponível em http://www.extremeprogramming.org, visitado em 07/10/2007.

[5] PMI, Guia PMBOK, 3ª edição – **Um Guia do Conjunto de Conhecimentos em gerenciamento de projetos.** Editora Global Standard, 2004.

[6] CMMI (Capability Maturity Model for Software In) – Disponível em <http://www.sei.cmu.edu/cmmi>. Acessado em 15 de outubro de 2007.

[7] ftp://ftp.sei.cmu.edu/pub/documents/07.reports/07tr017.doc <http://www.sei.cmu.edu/cmm/cmm.html >. Acessado em 15 de outubro de 2007.

[8] **SCRUM - It's About Common Sens -** Disponível em http://www.controlchaos.com/, Visitado em 08/10/2007.

[9] Schwaber, Ken, **What Is Scrum? -** Disponível em http://www.scrumalliance.org/system/resource/file/275/whatIsScrum.pdf, visitado em Outubro de 2007.

[10] James Bach. October 1995. "American Programmer", **SCRUM Software Development Process**, disponível em http://www.controlchaos.com/old-site/scrumwp.htm#Phases, visitado em Outubro/2007.

[11] Marçal, A., Freitas, B., Soares, F., Belchior, A., **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI**, disponível em http://www.cesar.org.br/files/file/SCRUMxCMMI-CLEI-2007.pdf, visitado em 10/10/2007.

[12] Advanced Development Methods – ADM, **Controlled chaos: living on the edge**, http://www.controlchaos.com/old-site/ap.htm, (Outubro, 2007).

[13] Henrique Borges Alencar Siqueira – **Mapeamento das práticas de Scrum nas áreas de processo do CMMI e uma proposta para sua aderência**. Disponível em http://www.cin.ufpe.br/~tg/2007-1/hbas.pdf, visitado em Dezembro/2007.

[14] BECK, K. - **Embracing change with Extreme Programming. IEEE Computer.** V. 32, n. 10, p. 70-77, 1999.

[15] Fábio Camara-**Um cardápio de metodologias ágeis**–disponível em http://imasters.uol.com.br/artigo/7396/gerencia/um_cardapio_de_metodologias_ageis/ (Janeiro, 2008).

[16] GRENNING, J. - **Launching extreme programming at a process-intensive company**. IEEE Software, v.18, n.6, p. 27-33, 2001.

[17] LIPPERT, M. et. al. - **Developing complex projects using XP with extensions**. IEEE Computer. V. 36, n. 6, p. 67-73, 2003.

[18] MARTIN, R.C. - **eXtreme Programming development through dialog.** IEEE Software, p. 12-13, 2000.

[19] Michel dos Santos Soares – **Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software.** Disponível em www.facecla.com.br/revistas/resi/edicoes/ed4artigo06.pdf, visitado em Outubro/2007.

[20] PRESSMAN, R. S., - **Engenharia de Software**, 6ª edição, Editora: McGraw-Hill, 2006.

[21] POOLE, C.; HUISMAN, J. W. - **Using extreme programming in a maintenance environment.** IEEE Software. v. 18, n. 6, p. 42-50, 2001.