UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

RICARDO BRINA MONDARDO

UTILIZAÇÃO DO *FRAMEWORK* EPF *COMPOSER* PARA A CRIAÇÃO DE UM MODELO DE PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO NA INTEGRAÇÃO DE METODOLOGIAS ÁGEIS COM TRADICIONAIS

CRICIÚMA, NOVEMBRO DE 2010

RICARDO BRINA MONDARDO

UTILIZAÇÃO DO *FRAMEWOK* EPF *COMPOSER* PARA A CRIAÇÃO DE UM MODELO DE PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* BASEADO NA INTEGRAÇÃO DE METODOLOGIAS ÁGEIS COM TRADICIONAIS

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. MSc. Gustavo Bisognin

CRICIÚMA, NOVEMBRO DE 2010

RICARDO BRINA MONDARDO

Utilização do *Framework* Epf *Composer* para a Criação de um Modelo de Processo de Desenvolvimento de *Software* Baseado na Integração de Metodologias Ágeis com Tradicionais

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Profa. MSc. Ana Claudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

Prof. MSc. Gustavo Bisognin (UNESC)
Orientador

Prof. MSc. Paracelso de Oliveira Caldas (UNESC)

Esp. Murilo Burigo (Betha)

Dedico este trabalho a todos meus familiares e amigos pela compreensão e confiança ao longo de toda a minha formação.

AGRADECIMENTOS

Inicialmente agradeço a minha família pelo apoio concedido ao longo do desenvolvimento deste trabalho e por eu ter conseguido concluí-lo e realizar meu objetivo.

Agradeço também:

A Deus por ter me dado todos os requisitos necessários para concluir as minhas obrigações e futuros objetivos a alcançar.

A todos meus amigos que me apoiaram nessa jornada.

A meu orientador Gustavo Bisognin pelas orientações, pela sua compreensão e empenho na realização deste trabalho.

A Mateus Luiz Gambá por me ajudar em momentos de dificuldade na elaboração deste trabalho e na parceria durante o curso

Por fim, a todas as pessoas que me ajudaram direta e indiretamente.

RESUMO

Diante da constante busca pela melhoria do processo de desenvolvimento de *software* e a resolução de problemas como: a compreensão incorreta de requisitos do sistema, a pouca experiência de equipes, o custo elevado dos projetos de desenvolvimento entre outros, identifica-se um quadro de complexidade no desenvolvimento de *software*. Para minimizar esses problemas, a engenharia de *software* junto com as metodologias de desenvolvimento, procuram estabelecer métodos, técnicas e padrões para a melhoria do processo de desenvolvimento. Tendo em vista os fatos citados acima, este trabalho estudou princípios da engenharia de *software* como ciclos de vida de desenvolvimento de *software*, metodologias de desenvolvimento ágeis e tradicionais e ferramentas que auxiliem na construção de projetos de *software*. Cria-se assim um modelo de desenvolvimento de *software* baseado na integração de metodologias ágeis e tradicionais, que unem as melhores praticas dos dois tipos de abordagens desenvolvendo o processo com a utilização da ferramenta de gerenciamento de conteúdo EPF *Composer.*

Palavras-Chave: engenharia de *software*; metodologias de desenvolvimento tradicionais; metodologias de desenvolvimento ágeis.

ABSTRACT

In front of the constant search by get better the technique of the development of software and the resolution of problems such as: na wrong understanding of requirement of the system, a little experience of the groups, the high price of the projects of the development and others. It show a way of complexity in the development of software. To decrease these problem, the engineering of software together with the methodologies of development try to settle methods, techniques and patterns to get better the process of development according with these facts above, this work studied the origin of the engineering of software as cycles of life of development of software, methodologies of traditional and agile development and tools that help in the construction of projects of software. In this way begins a pattern of development of software based in the union of traditional and agile methodologies, that join the best practices from the two kinds of the themes developing the process with an use of the tool of management of content EPF *Composer*.

Key-words: engineering of software, methodologies of traditional development, methodologies of agile development.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CMMI	*Capability Maturity Model Integration*
DSDM	*Dynamic Systems Development Method*
FDD	*Feature Driven Development*
HTML	*Hyper Text Markup Language*
MPS.BR	Melhoria do Processo de Software Brasileiro
RUP	*Rational Unified Process*
SDLC	*Systems Development Life Cycle*
UML	*Unified Modeling Language*
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Uma metodologia de desenvolvimento de *software* é um conjunto de atividades e resultados associados que auxiliam na produção de artefatos e componentes de sistemas de *software*.

Na construção de um componente de *software*, nem sempre conseguimos compreender e desenvolver corretamente os requisitos do sistema, havendo impacto em prazo, custo e esforço. Além disso, o retrabalho causado pela inserção de *bugs* aumenta consideravelmente o custo da implementação do sistema.

Outro fator crítico de sucesso para o desenvolvimento de sistemas computacionais é a pouca experiência das equipes de desenvolvimento e a inexistência de recursos para o planejamento de projetos. Dentre as várias atividades existentes aplicadas na resolução deste tipo de problema, podendo-se citar: a análise, o desenvolvimento de requisitos, o planejamento de projetos e a codificação ágil.

Diante desse paradigma, identifica-se claramente a abordagem de metodologias ágeis como uma boa prática para obtenção de *software* com qualidade. As metodologias ágeis têm sido apontadas como uma alternativa às metodologias tradicionais de desenvolvimento, uma vez que essas possuem o foco na agilidade de desenvolvimento e na qualificação automática dos produtos de *software* enquanto as metodologias tradicionais possuem foco no processo e na forte documentação do sistema, consumindo um esforço consideravelmente grande para o seu desenvolvimento, que muitas vezes, é construído sem qualidade.

As metodologias tradicionais surgiram em um contexto de desenvolvimento de *software* muito diferente do atual. Nessa época, o custo de se

fazer alterações e correções era muito alto, uma vez que não existiam modernas ferramentas de apoio ao desenvolvimento do *software*, como depuradores e analisadores de código. Levando em consideração esses fatores sua documentação era toda feita antes de sua implementação. Uma das principais metodologias tradicionais atualmente utilizadas é o modelo desenvolvido pela IBM *Rational* conhecido como *Rational Unified Process* (RUP).

Pode-se afirmar então, que a união das boas práticas das metodologias tradicionais com a agilidade fornecida pelas metodologias ágeis compõem um processo desburocratizado, controlado, gerenciado e ágil, podendo ser evoluído constantemente com os feedbacks gerados durante sua execução.

O planejamento de projetos de *software* utilizando metodologias ágeis tem um ciclo de vida em integração contínua, enfatizando entregas rápidas e funcionais do *software*.

Outra característica importante é a alta integração com o cliente onde o mesmo tem sempre participações no desenvolvimento (BASSI, 2008).

Atualmente existem vários métodos ágeis, dentre eles podem-se citar os seguintes: *Scrum*, *Extreme Programming* (XP), *Dynamic Systems Development Method* (DSDM), *Feature Driven Development* (FDD), entre outros. Alguns desses métodos já estão sendo usados por grandes corporações como Google, Nokia, HP, DELL, SAP, etc.

Segundo Sommerville (2003) o modelo de processo de *software* é apenas uma representação abstrata de como deve ser construído o modelo de *software*, com informações parciais de como será gerenciado.

As metodologias ágeis de desenvolvimento surgiram como medida para a resolução de alguns problemas críticos identificados pela engenharia de *software*

atual com a utilização dos modelos tradicionais. Tomando como base essa premissa, propõe-se como trabalho de conclusão de curso, a união das melhores práticas dos métodos ágeis com disciplinas da engenharia de *software* tradicional para a composição de um processo modelo aplicado no desenvolvimento de *software* com qualidade.

1.1 OBJETIVO GERAL

Desenvolver um processo modelo para o desenvolvimento de *software*, abordando a união das melhores práticas dos métodos ágeis com a engenharia de *software* tradicional.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do presente trabalho estão baseados nos seguinte itens:

a. compreender o funcionamento do framework EPF *Composer*;
b. estudo do modelo de processo OpenUP;
c. estudar as metodologias RUP, XP e *Scrum*;
d. analise de ferramentas para integração contínua;
e. construção de um processo de engenharia de *software*;

1.3 JUSTIFICATIVA

Em planejamento de projetos para o desenvolvimento de *software* nem

sempre é possível atingir bons resultados, como o cumprimento das atividades sem atrasos, o desenvolvimento dos requisitos do sistema de forma correta, a conclusão do projeto sem exceder orçamento e uma série de outros fatores que possam garantir a qualidade do produto de *software*.

Conforme pesquisas feitas em 1994, descobriu-se que dentre 365 empresas que colaboraram com 3682 projetos, 83.2% foram entregues com prazo ou orçamento excedidos ou cancelados. O custo desses projetos foi de 89% acima do previsto e o atraso médio foi de 122% do que se esperava no planejado. Os participantes dessa pesquisa apontaram seis fatores para esses resultados, que são: requisitos incompletos, falta de envolvimento dos usuários, mudanças de requisitos e especificações, falta de apoio da equipe de negócios, falta de recursos e expectativas não-realistas. Em pesquisas mais recentes apenas 29% dos projetos são finalizados com sucesso demonstrando que os projetos com prazos e custos excedidos se mantiveram com percentuais tão elevados quanto antes (BASSI, 2008).

Uma solução para esse problema é a utilização de métodos ágeis, conforme aponta Zanatta (2004) os métodos ágeis priorizam a entrega rápida e contínua do *software*, procurando simplicidade no planejamento e a alteração de requisitos de forma rápida sempre buscando um desenvolvimento sustentável.

O manifesto ágil ressalta que para os métodos ágeis as interações são mais interessantes que as ferramentas e documentações pesadas. Também afirma que a colaboração do cliente não fica apenas nos contratos se estendendo ao desenvolvimento do projeto e que a adaptação perante as mudanças, assumem maior importância que seguir o plano inicial.

Um método ágil para o desenvolvimento de *software* que está se

destacando é o *Scrum*, que trabalha muito bem com equipes pequenas, e com requisitos do sistema que podem mudar com mais frequência. Outra característica importante, conforme afirma Bassi (2008) é sua facilidade de iteração com outros métodos ágeis, como por exemplo, a integração *Scrum*/XP, onde o método *Scrum* tem suas atividades voltadas para práticas organizacionais, e o XP tem suas tarefas mais relacionadas com a programação em si. Segundo Beck (2000) o método ágil XP se baseia em técnicas simples, controlando o desenvolvimento do *software* com vários ajustes de pequeno porte e não poucos ajustes de grande proporção.

Porém, nem sempre é possível usar integralmente métodos ágeis, em casos de projetos que apresentem fatores como: equipes geograficamente dispersas, programadores que não tem o comprometimento ideal com o projeto, equipes com grande volume de pessoas envolvidas, projetos de grande porte e/ou envolvendo situações de risco. Deve-se recorrer à utilização de métodos tradicionais como uma alternativa mais adequada para o desenvolvimento.

Métodos tradicionais têm foco em organização e estrutura do projeto, podemos citar suas atividades genéricas da seguinte forma: comunicação, planejamento, modelagem, construção e implementação (PRESSMAN, 2006).

Conforme aborda Pressman (2006), a escolha de um modelo para o desenvolvimento *software* não é uma decisão fácil, as alternativas disponíveis para os engenheiros de *software* são os chamados métodos tradicionais como RUP, que tem na ordem e consistência do projeto, seus tópicos dominantes e por outro lado, os métodos ágeis que abordam a auto-organização, colaboração, comunicação e adaptabilidade como pontos fortes.

Baseando-se nos fatores acima apresentados e no grande anseio dos engenheiros de *software* pela utilização de métodos que remetam a organização e

colaboração em projetos de *software* de forma ágil e consistente, contextualiza-se a proposta para a criação de um processo modelo abordando as melhores práticas das metodologias ágeis e tradicionais, referida neste trabalho de conclusão de curso.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está descrito em oito capítulos definindo no primeiro, uma introdução sobre o referente trabalho, os objetivos gerais e específicos e finalizando com a justificativa do trabalho.

No capítulo dois tem-se uma breve história sobre a evolução da engenharia de *software*, seus ciclos de vida de desenvolvimento tradicionais e uma definição de como é o funcionamento de um ciclo de vida de desenvolvimento ágil.

O capitulo três refere-se às metodologias de desenvolvimento ágeis e tradicionais e algumas comparações entre os dois tipos; no capítulo quatro é apresentada a ferramenta EPF *Composer*, que é utilizada na criação e documentação do modelo proposto, em seguida são apresentados os trabalhos correlatos no capítulo cinco.

Por fim, tem-se uma descrição do modelo criado no capitulo seis, a metodologia utilizada para criação no capítulo sete, os resultados obtidos com a realização do trabalho no capítulo oito.

2 ENGENHARIA DE *SOFTWARE*

Engenharia de *software* é uma área do conhecimento que tem como objetivo a padronização e organização do desenvolvimento de *software* em projetos, buscando um aumento da produtividade e da qualidade dos produtos e processos envolvidos na concepção de sistemas computacionais.

Pressman (2006) afirma que a engenharia de *software* oferece ao engenheiro de *software* uma base de conhecimento para construção de um produto de qualidade. Sendo que para isso ocorrer, essa disciplina dispõe de quatro elementos fundamentais que são: métodos, ferramentas, procedimentos e pessoas.

Neste capítulo será realizado o estudo da evolução da engenharia de *software* bem como seus ciclos de vida de desenvolvimento, expondo as principais práticas envolvidas nesta área de conhecimento.

2.2 EVOLUÇÃO DA ENGENHARIA DE *SOFTWARE*

Em meados de 1970, o mercado de *software* passava por uma crise, e para solucioná-la surge a engenharia de *software* que propõe um tratamento mais sistemático e controlado para construção de *softwares* complexos, que envolvem um conjunto de componentes de *software*, como estruturas de dados, algoritmos, procedimentos, funções entre outros (PRESSMAN 2006).

Inicialmente os primeiros modelos usados para o desenvolvimento de projetos na área de *software*, eram baseados em técnicas utilizadas com sucesso na engenharia industrial em série.

Pode-se citar o modelo em cascata como o paradigma mais antigo e

amplamente utilizado na engenharia de *software*, segundo Pressman (2006) atualmente esse modelo esta sendo questionado até mesmo por seus ativos defensores por não se encaixar muito bem em situações que ocorrem em projetos reais.

Com a evolução da engenharia de *software*, originaram-se novos modelos de desenvolvimento de *softwares* com novos procedimentos que são mais adaptáveis a situações reais vividas nas empresas. Dentre esses modelos destacam-se os modelos ágeis, que proporcionam uma alteração do foco dos projetos dando ênfase a codificação e minimizando os esforços com a documentação.

2.3 CICLO DE VIDA DE DESENVOLVIMENTO DE *SOFTWARE*

Ciclo de vida de desenvolvimento de *software Systems Development Life Cycle* (SDLC), também chamado de ciclo de vida do *software*, trata-se genericamente dos estágios de concepção, projeto, criação e implementação, podendo variar conforme o modelo que se é utilizado.

Cada organização possui uma forma específica de desenvolver seus projetos, necessitando de recursos como pessoas, equipamento e *software*, nesse contexto uma empresa que tem um SDLC formalizado pode equilibrar de forma mais previsível seu processo de desenvolvimento de *software* até sua entrega final, alem disso, um SDLC bem definido auxilia na medição dos resultados apresentados e na qualidade final do produto. A seguir estão alguns modelos de SDLC.

2.3.1 Modelo Cascata

Modelo cascata ou *waterfall* como também pode ser chamado, segue uma linha de desenvolvimento linear e seqüencial, dessa forma ao iniciar uma etapa, a mesma tem que ser concluída para que só então se inicie uma próxima. Nesse tipo de abordagem não existe a possibilidade de retorno a uma etapa anterior. De acordo com a afirmação de Pressman (2006) esse modelo segue um ordem restrita e sem interatividade, o que está sendo muito criticado atualmente.

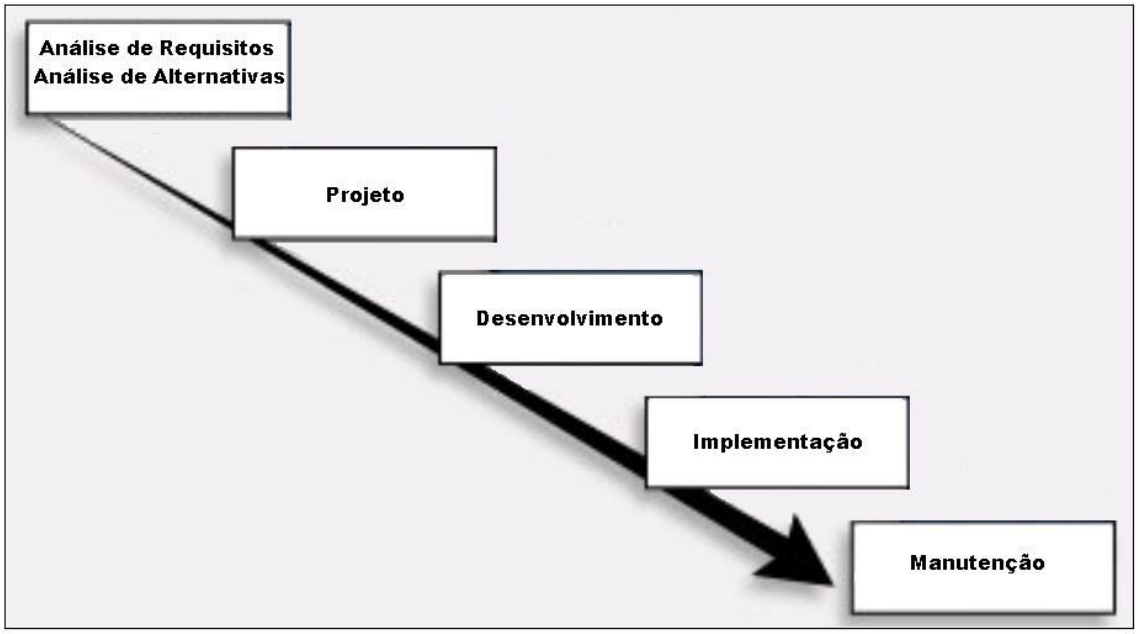


Figura 1. Modelo Cascata
Fonte: Adaptado de PRESSMAN, R. (2006, p.39)

O modelo cascata pode ser aplicado de formas diferentes, na Figura 1 pode-se ver um tipo de utilização do mesmo que consiste nas etapas de análise de requisitos, análise de alternativas, projeto, desenvolvimento, implementação e manutenção. As etapas deste modelo são:

a. análise de requisitos e análise de alternativas: essas fases atuam na identificação das funções que o *software* deverá atender e avaliação de

sistemas alternativos que possam atender aos requisitos expostos pelo cliente;

b. projeto: consiste no planejamento do banco de dados, projeto de interfaces, características físicas do sistema, projeto de hardware, nessa etapa deve-se também identificar os procedimentos para o teste do sistema completo antes da instalação;
c. desenvolvimento: tem como composição a escrita do código ou aquisição do *software* proposto e também o teste do novo sistema;
d. Implementação: ocorre a transferência para o ambiente de produção e os testes de aceitação do cliente;
e. manutenção: corresponde a todas atividades relacionadas, como o sistema que são feitas, após sua implementação correspondente a fase de manutenção.

Esse tipo de abordagem seqüencial traz uma desvantagem, não permiti que se tenha uma revisão das etapas anteriores do desenvolvimento, dessa forma quando se detecta algum problema na fase de testes, se torna muito difícil o retorno para se possa fazer um correção (SOMMERVILLE, 2003).

Pressman (2006) aponta alguns problemas identificados em projetos que utilizam este modelo, que são:

a. projetos reais raramente seguem o fluxo sequencial que o modelo propõe;
b. muitas vezes é difícil compreender os requisitos do sistema na fase inicial;
c. versões de trabalho do programa estarão disponíveis em um ponto tardio.

2.3.2 Modelo Iterativo Incremental

Desenvolvimento seguindo a abordagem interativa incremental, segue um linha semelhante ao modelo cascata, tendo as fases de análise, projeto, implementação e testes sendo realizada uma única vez a cada ciclo, onde cada um desses ciclos contém um subconjunto de requisitos que passa por todas essas fases, como podemos ver na Figura 2.

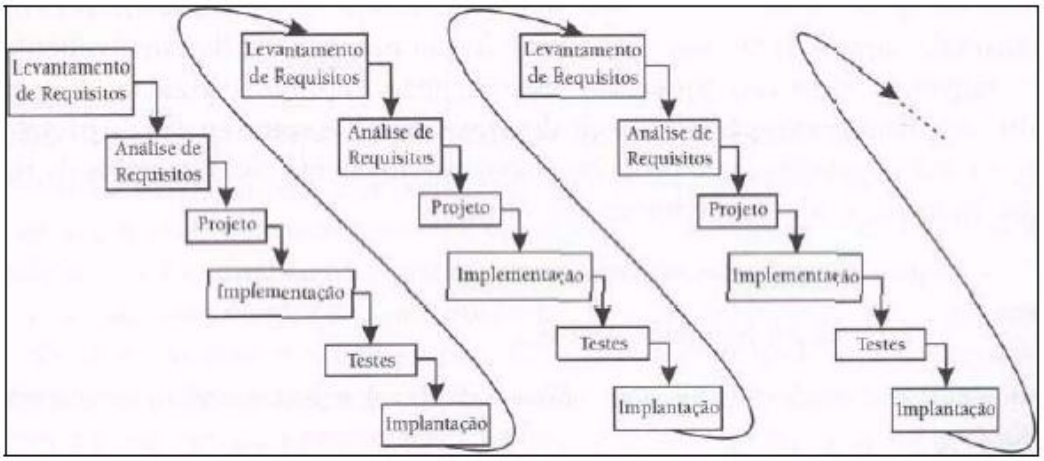


Figura 2. Processo Incremental e Iterativo
Fonte: BEZZERA, E. (2002, p. 8)

Conforme afirma Bezzera (2002), cada ciclo considera um conjunto de requisitos, que ao ser finalizado, dá início a outro onde esse terá um novo conjunto de requisitos a serem desenvolvidos. Ao final de cada ciclo é produzido um novo incremento ao sistema, que contém extensões e refinamentos sobre o incremento gerado anteriormente.

Dessa forma o desenvolvimento ocorre na forma de versões, deixando o usuário mais participativo, sendo que este está ao fim de cada ciclo, testando uma nova versão com mais funcionalidades prontas do sistema, diminuindo

consideravelmente a probabilidade da ocorrência de erros relacionados a especificação de requisitos do projeto de *software*.

2.2.3 Modelo de Prototipação

Segundo Pressman (2006) em projetos que o cliente não consegue definir os requisitos de entrada claramente, o desenvolvedor não tem certeza de qual solução técnica deve utilizar para a resolução do problema. Outro problema clássico de falta de engenharia é a ambigüidade quanto a especificação da iteração homem-máquina, gerando insatisfação do cliente quanto a usabilidade do sistema. Tendo em vista a resolução dos casos acima citados, o modelo de desenvolvimento utilizando o ciclo de vida prototipação pode ser destacado com sendo bastante adequado .

Um modelo seguindo essa abordagem pode seguir de três formas, podendo ser um modelo em papel que retrata a interação homem-máquina, um protótipo de trabalho que implementa algumas funções básicas do sistema ou um programa já existente que executa parte das funções desejadas pelo cliente.

Figura 3: Modelo Prototipação.
Fonte: PRESSMAN, R. (1995, p. 39)

Nesse tipo modelagem, como podemos notar na Figura 3, inicialmente é

feito a coleta e definição dos objetivos do *software*, logo após é elaborado um projeto rápido que vai contemplar aspectos que serão visíveis ao cliente como entradas e saídas, servindo para construção de um protótipo, onde será avaliado pelo cliente e refinado até que satisfaça as necessidades do mesmo, para que só então se inicie a engenharia do produto.

2.2.4 Modelo em Espiral

O desenvolvimento de *software* utilizando o modelo em espiral, segundo Pressman (2006) é realizado seguindo as melhores características dos modelos cascata e prototipação, porém, nesse modelo é acrescentada a análise de risco como mais um elemento de suma importância para o sucesso do projeto.

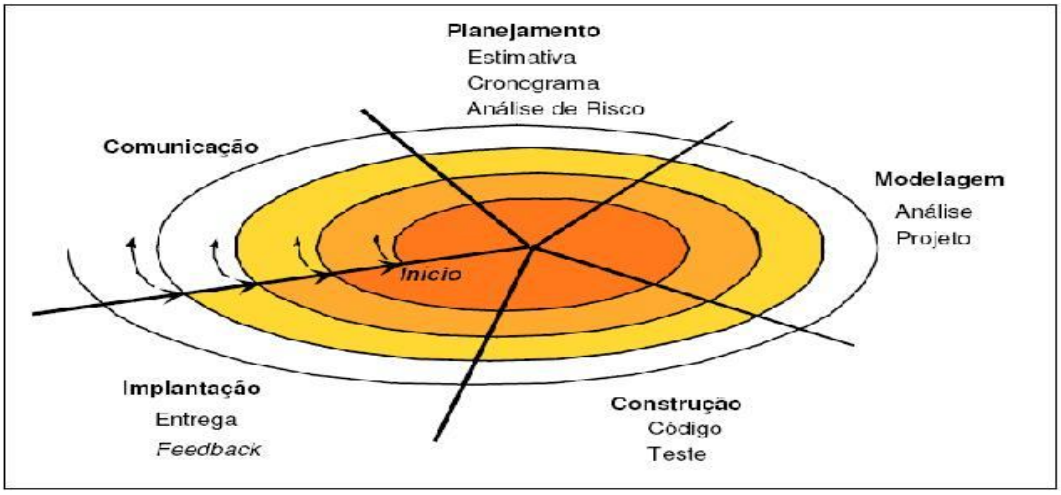


Figura 4. Modelo Espiral
Fonte: PRESSMAN, R. (2006, p.45)

Nesse tipo de abordagem como podemos notar na Figura 4, ao final de cada iteração ao redor da espiral tem se uma nova versão do sistema, que o cliente avalia e apresenta suas sugestões, que são utilizadas na fase de avaliação de risco, onde será decidido se o projeto continua ou é cancelado.

Segundo Pressman (2006), uma característica interessante desse modelo, é que o desenvolvedor pode utilizar abordagens de prototipação em qualquer etapa da evolução do sistema, e também pode valer-se de passos sistemáticos sugeridos pelo modelo de ciclo de vida cascata, dessa forma tornando o ciclo mais realista em reação ao desenvolvimento de sistemas.

2.3 CICLOS DE VIDA DE DESENVOLVIMENTO ÁGIL

Os modelos de desenvolvimento ágil, sugerem um tipo de abordagem que propõem algumas melhoras em relação a visão tradicional, tratando a construção do *software* de forma mais rápida e eficaz.

O desenvolvimento de projetos de *software* utilizando metodologia ágil ocorre através de interações, que são concluídas em pequenos intervalos de tempo, durando em média de uma à quatro semanas. Cada iteração feita inclui todas as tarefas necessárias para uma implantação do sistema, tais como: planejamento, análise de requisitos, projeto, codificação, testes e documentação da versão, podendo variar conforme a metodologia utilizada.

Alguns autores apontam que o desenvolvimento ágil, em contraposição aos modelos tradicionais, envolve a imediata codificação e teste do sistema na forma de iteração.

Modelos de desenvolvimento ágil são leves, dessa maneira ideais para pequenas e médias equipes, trabalhando muito bem como requisitos vagos e mudanças contínuas. Esses modelos também apresentam uma boa alternativa para a documentação envolvida no projeto, em relação ao modelos tradicionais, mantendo o foco no código, desvinculando o esforço aplicado na evolução de

artefatos de projeto que não agregam valor direto ao produto de *software* (BECK, 2000).

Assim como em outros modelos de desenvolvimento de *software*, o modelo ágil possui algumas características principais, das quais se destacam:

a. entrega do *software* em versões com novas funcionalidades incrementais;
b. comentários e participações do cliente são sempre bem vindos;
c. alterações nos requisitos são feitas sempre que necessário;
d. simplicidade e eficácia são essenciais;
e. documentação mais leve e apenas quando realmente ela é necessária;
f. comunicação em tempo real.

Zanatta (2004) considera que o modelo ágil difere dos modelos tradicionais basicamente em dois aspectos:

a. são mais adaptáveis que os modelos tradicionais;
b. são orientados a pessoas ao invés de processos.

Nesse capítulo foram expostos alguns dos mais conhecidos modelos tradicionais de desenvolvimento de *software*, e demonstrado de uma forma básica, as características de um modelo de desenvolvimento ágil. Pode-se considerar que os dois tipos de modelos, tradicional e ágil, têm suas qualidades e desvantagens. Percebe-se que os modelos tradicionais são mais adequados a grandes projetos, onde se tem a disposição equipes maiores e mais recurso; entretanto, modelos ágeis se encaixam melhor em projetos com equipes menores, pois trabalham melhor com a falta de recursos e mudanças constantes.

3 METODOLOGIAS DE DESENVOLVIMENTO

A construção de um *software* pode ser algo muito complexo, pois envolve várias tarefas e procedimentos. Em consequência disso, existem metodologias de desenvolvimento que são responsáveis por fazer com que a equipe envolvida no projeto, possa interagir com todos os procedimentos e tarefas necessários na construção do *software* de forma objetiva e sistemática.

Nessa seção, será apresentada uma visão sobre o surgimento, propósito e princípios das metodologias ágeis de desenvolvimento.

3.1 MANIFESTO ÁGIL

O manifesto ágil teve seu surgimento no início de 2001, quando dezessete pessoas se reuniram para discutir os rumos do desenvolvimento de *software*, obtendo como resultando o Manifesto do Desenvolvimento Ágil, também conhecido simplesmente por Manifesto Ágil. Tal documento descreve um conjunto de técnicas e práticas, criadas a partir de experiências que esses profissionais adquiriram com o desenvolvimento de *software*, ao longo de suas carreiras.

De acordo com Manifesto ágil (2001), os propósitos do desenvolvimento ágil são:

a. indivíduos e situações são mais importantes que processos e ferramentas;
b. *software* funcionando em vez de documentações abrangentes;
c. colaboração do cliente e não apenas contratos;
d. adequação a mudanças mesmo que se tenha um plano a seguir.

O manifesto ágil trata o desenvolvimento do *software* de uma forma prática, sem tantas formalidades como em um modelo tradicional, definindo como prioridade a entrega de executáveis à seus clientes e priorizando o funcionamento do *software*, ao invés de sua documentação. É interessante ressaltar que o manifesto não é contra documentações e sim, indica que essas devem ser feitas apenas quando realmente se fazem necessário.

Segundo Pressman (2006 p. 60), O Manifesto Ágil 2001 define doze princípios para que se possa atingir a agilidade:

a. nossa maior prioridade é satisfazer ao cliente desde o início por meio de entrega contínua do *software* valioso;
b. modificações de requisitos são bem-vindas, mesmo que tardias no desenvolvimento. Os processos ágeis aproveitam as modificações como vantagens para competitividade do cliente;
c. entrega de *software* funcionando frequentemente, a cada duas semanas até dois meses, de preferência no menor espaço de tempo;
d. o pessoal de negócio e os desenvolvedores devem trabalhar juntos diariamente durante todo o projeto;
e. construção de projetos em torno de indivíduos motivados, fornecendo-lhes o ambiente e apoio à construção;
f. o método mais eficiente e efetivo de levar informação para dentro de uma equipe é a conversa face a face;
g. *software* funcionando é a principal medida do progresso;
h. processos ágeis apresentam desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante, indefinidamente;

i. atenção contínua à experiência técnica facilitando a agilidade;
j. simplicidade - a arte de maximizar a quantidade de trabalho não efetuado é essencial;
k. as melhores arquiteturas, requisitos e projetos surgem de equipes auto organizadas;
l. em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, então sintonizando e ajustando adequadamente seu comportamento.

Atualmente contamos com diversos métodos ágeis, onde se destacam: *Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development* e *Dynamic Systems Development Method.*

Cada um dos métodos acima citados possui particularidades e o sucesso de sua aplicação, depende diretamente do tipo de configuração do projeto abordado. Dentre os principais métodos apresentados nesse trabalho, podemos destacar o *Extreme Programming* e o *Scrum*, que, atualmente, estão sendo utilizados com grande empolgação pela comunidade desenvolvedora de *software*.

3.2. METODOLOGIAS ÁGEIS

Inicialmente essa seção aborda o Método *Scrum* e em seguida o Método Xp, apresentando suas origens, características, papeis principais e uma visão do desenvolvimento de projetos com sua utilização.

3.2.1 Método *Scrum*

Método de desenvolvimento ágil *Scrum* criado por Jeff Sutherland e Ken Schwaber no início da década de 1990, tem seu nome originário de um jogo de *Rugby*[1]. Segundo Schwaber (2002) o método trabalha o desenvolvimento de *software* de forma incremental e iterativa, focando nas pessoas e ambientes em situações onde os requisitos surgem ou mudam constantemente.

Conforme Schwaber (2002) o *Scrum* tem como características, requisitos instáveis ou até desconhecidos, desenvolvimento dividido em iterações que são chamadas de *Sprint*s e duram de duas a quatro semanas.

As equipes são auto-organizadas, a comunicação entre membros e com partes interessadas no projeto é verbal, e os integrantes de uma equipe *Scrum* sabem concluir suas tarefas de forma ágil e rápida.

O desenvolvimento no método *Scrum* se baseia nos papéis, *Product Owner*, *Scrum Master* e *Scrum* Team, seu processo é dividido pelas fases PreGame, Game e PostGame, onde são aplicadas as práticas gerenciais Sprint, Sprint Planning Meeting, Daily *Scrum*, Sprint Review Meeting, que são responsáveis pelos artefatos Product Backlog, *Sprint backlog*, *Burndown Chart e* Planning Poker, que é usado como métrica para mensuração das estimativas. A seguir pode ser visto por meio da Figura 5 o funcionamento do *Scrum*.

1 *Rugby*: Esporte originário da Inglaterra, onde a disputa de bola se chama *Scrum*

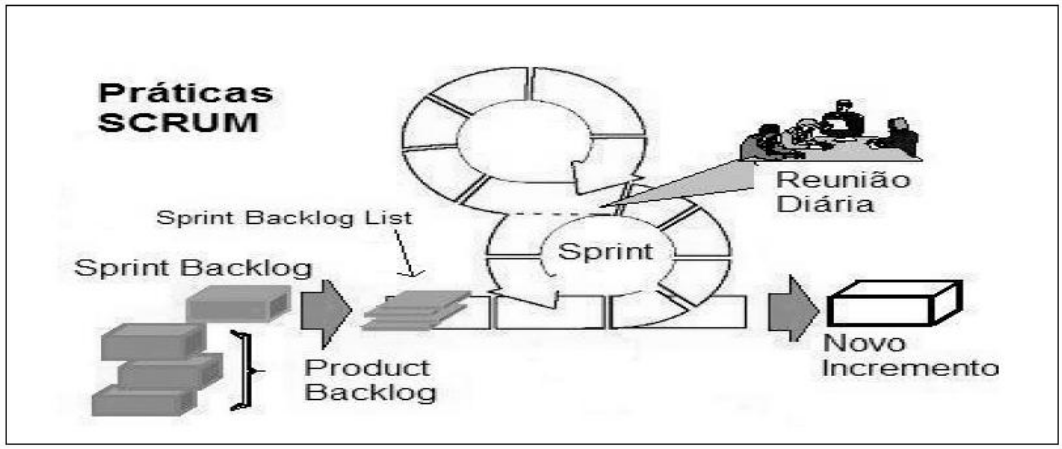


Figura 5. Funcionamento do *Scrum*
Fonte: ZANATTA (2004, p.48).

3.2.1.1. Práticas e Artefatos do *Scrum*

O Método *Scrum* já citado anteriormente, se baseia em práticas gerenciais e artefatos para compor o desenvolvimento do sistema. Abaixo são descritos com mais detalhes cada uma das práticas gerenciais e artefatos do método.

3.2.1.1.1 Sprint

Geralmente com duração de 2 a 4 semanas, é a etapa onde são implementados as funcionalidades descritas no Product Backlog. Segundo Abrahmsson (2002), o *Sprint* contém as tarefas tradicionais do desenvolvimento de *software* que são: análise, projeto e entrega.

Segundo Zanatta (2004) o *Sprint* é a principal fase do *Scrum*, onde são executados os itens definidos no *Product Backlog*, obtendo como resultado em um novo incremento. O autor ainda descreve que não existem processos pré-definidos dentro dessa fase, ao invés disso, durante o *Sprint* ocorrem reuniões periódicas

chamadas *Daily Scrum,* que são responsáveis por coordernar as atividades existentes.

No início de cada *Sprint*, são definidos quais itens serão desenvolvidos no decorrer através da etapa *Sprint Planning Meeting*, em seguida, é descrito como serão implementados tais itens resultando em um *Sprint backlog*.

Ao fim do Sprint é inciado a etapa *Sprint Review Meeting,* onde é feito a revisão dos itens incompletos e finalizados. Esses itens serão re-organizados gerando alterações e inclusões de incrementos ao *Product Backlog*,

3.2.1.1.2 Sprint Planning Meeting

Cada *Sprint* inicia com uma etapa chamada *Sprint Planning Meeting,* que tem como objetivo verificar e definir os itens do *Product Backlog* a serem realizados durante o *Sprint*, esse procedimento se divide em dois momentos:

a. inicialmente o *Product Owner* seleciona e apresenta à equipe detalhadamente os itens do *Procuct Backlog*, objetivando repassar o conhecimento necessário à equipe, para que a mesma possa desenvolver e definir quais itens serão capazes de finalizar no *Sprint*;
b. em um segundo momento, são definidos como serão desenvolvidos esses itens que foram escolhidos com base em estimativas para o desenvolvimento no Sprint. Esse procedimento é chamado de *Sprint Goal.*

3.2.1.1.3 Daiy Scrum

De acordo com Linda e Norman (2003) são reuniões feitas diariamente ou em dias alternados, onde a equipe se reúne de pé, com tempo limite de 30 minutos sendo ideal de 15 minutos.

Schwaber (2002, p.43) destaca três questões levantadas pelo *Scrum Master* rotineiramente nessa reunião que são:

a. o que foi finalizado desde a última reunião do grupo? O *Scrum Master* registra quais tarefas foram completadas e quais ainda estão pendentes;
b. quais foram as dificuldades encontradas durante o trabalho? O *Scrum Master* registra todas as dificuldades encontradas, para posteriormente encontrar uma maneira de resolvê-las;
c. quais são as atividades específicas que a pessoa planeja finalizar para o próximo encontro? O *Scrum Master* ajuda os integrantes da equipe a escolher as tarefas mais importantes, devido ao curto espaço de tempo, em média 24 horas.

A reunião é focada no *Product Backlog* fazendo com que os itens finalizados durante o *Sprint* sejam retirados, e a equipe fique atualizada com o progresso e dificuldades encontradas no projeto.

3.2.1.1.4 Sprint Review Meeting

No último dia do *Sprint* é apresentado ao cliente e a equipe os resultados do mesmo em uma reunião informal, em seguida esses resultados são discutidos

para que se possa definir quais novas atividades vão fazer parte do *Product Backlog*.

Segundo Schwaber (2004, nossa tradução) pode-se destacar como características da etapa *Sprint Review Meeting:*

a. reuniões com duração máxima de 4 horas;
b. a preparação da equipe para um reunião não pode passar de 1 hora;
c. a equipe tem o dever de apresentar as tarefas finalizadas para o *Product Owner* e aos *Stakeholders*;
d. não se deve apresentar tarefas incompletas;
e. as tarefas devem ser apresentadas em um ambiente semelhante ao de homologação;
f. a reunião tem início com a apresentação do *Sprint*, itens do *Product Backlog* comprometidos e os que foram finalizados;
g. a reunião tem como prioridade apresentar à equipe, as tarefas propostas, respondendo as perguntas e tomando nota das mudanças solicitadas;
h. no final da apresentação das funcionalidades, é questionado aos *stakeholders* suas impressões sobre as funcionalidades mudanças necessárias e alterações de prioridades que os mesmos julgam necessário;
i. o *Product Owner* pode identificar funcionalidades que não foram entregues conforme o esperado, e inclui-las no próximo *Sprint* com prioridade alta;
j. o *Scrum Master* avalia e define quantas pessoas irão participar da reunião, para organizar o ambiente de forma que fique confortável;
k. ao final da reunião, o *Scrum Master* agenda uma próxima reunião junto

com o *Product Owner* e os *Stakeholders*.

3.2.1.1.5 Product Backlog

Nessa atividade é feita a coleta das necessidades do cliente, para que se possa desenvolver a lista de requisitos, ou seja, serão definidas as funcionalidades, características, padrões, prioridades e estratégias.

Segundo Schwaber (2002) essa prática contém a análise das necessidades do negócio e requisitos que serão desenvolvidos ao longo do processo do projeto, ocorrendo por meio de reuniões com o *Product Onwer*, *Scrum Master* e os *Stakeholders*.

Durante a análise é elaborado o Product Backlog, que é a lista de requisitos discutidos e que serão desenvolvidos ao longo dos *Sprint*s. Um exemplo de *Product Backlog* pode ser visto na Tabela 1.

Tabela 1. *Product Backlog*

***PRODUCT BACKLOG* (exemplo)**				
IDNome	**Imp.**	**Est.**	**Como Demonstrar**	**Notas**
1Depósito	30	5	Logar-se, abrir a página de depósito, depositar R$ 10,00, ir para a página do meu saldo e verificar que este aumentou em R$ 10,00.	Precisa de um diagrama UML de sequência. Não é necessário se preocupar com criptografia por enquanto
2Verificar seu próprio histórico de transações	10	8	Logar-se, clicar em “transações”. Fazer um depósito. Voltar para transações. Verificar se o novo depósito é listado.	Usar paginação para evitar consultas muito grandes ao banco de dados. Projetar de forma similar à página de visualização de usuários.

Fonte: KINIBERG, H (2007, p.10)

Kiniberg (2007) descreve os itens do *Product Backlog* da seguinte forma:

a. ID: identificação única, usada para que se tenha controle das estórias;
b. nome: preferencialmente curto e que deixe claro o objetivo da estória;
c. importância: recebe uma pontuação, sendo que quando maior o número, mais importante (esse valor será definido pelo Produc owner);
d. estimativa inicial: geralmente é utilizado uma relação “Homem/Dia” , (esse valor será definido pela equipe por meio de métricas).
e. escopo: descrição de alto nível da estória, demonstrando o funcionamento da estória.
f. nota: qualquer esclarecimento ou detalhe adicional em relação ao Escopo.

Ainda segundo Kiniberg (2007) é possível adicionar outros campos ao *Product Backlog,* citando como exemplo de informação extra o campo: “responsável pela atividade” que irá conter o nome da pessoa responsável para desenvolver a determinada tarefa.

Normalmente esse documento é feito no Excel armazenado e compartilhado com todos os membros da equipe.

3.2.1.1.6 Sprint backlog

Sprint backlog segundo Zanatta (2004) é o ponto inicial de cada *Sprint,* sendo por meio dele que a equipe vai trabalhar. O *Sprint backlog* é composto por itens selecionados do *Product backlog* divididos em requisitos menores que

geralmente são concluídos em um dia.

Sprint backlog pode ser apresentado seguindo uma técnica denominada *Kambam*, onde são inseridas as informação em um painel alocado em uma parede visível a todos membros da equipe. Pode ser visto na Figura 6, um exemplo de *Sprint backlog* apresentado com a utilização de um *Kambam*.

Figura 6: *Sprint backlog*
Fonte: KINIBERG, H (2007, p.48).

Segundo Kiniberg (2007) é possível incluir novas colunas no *Sprint backlog* como por exemplo: “Cancelados” ou “Esperando pelo teste de iteração”. Entretanto ainda segundo o autor, colunas demais podem ser prejudiciais ao projeto desviando a atenção da equipe do que é realmente importante.

3.2.1.1.7 Burndown Chart

Uma ferramenta muito interessante utilizada no *Scrum* é o *Burndown Chart*, que tem por objetivo medir o *status* do projeto. É composto pelo eixo *X* representando os dias do *Sprint* e o eixo *Y* representando o trabalho restante, como pode ser visto na Figura 7.

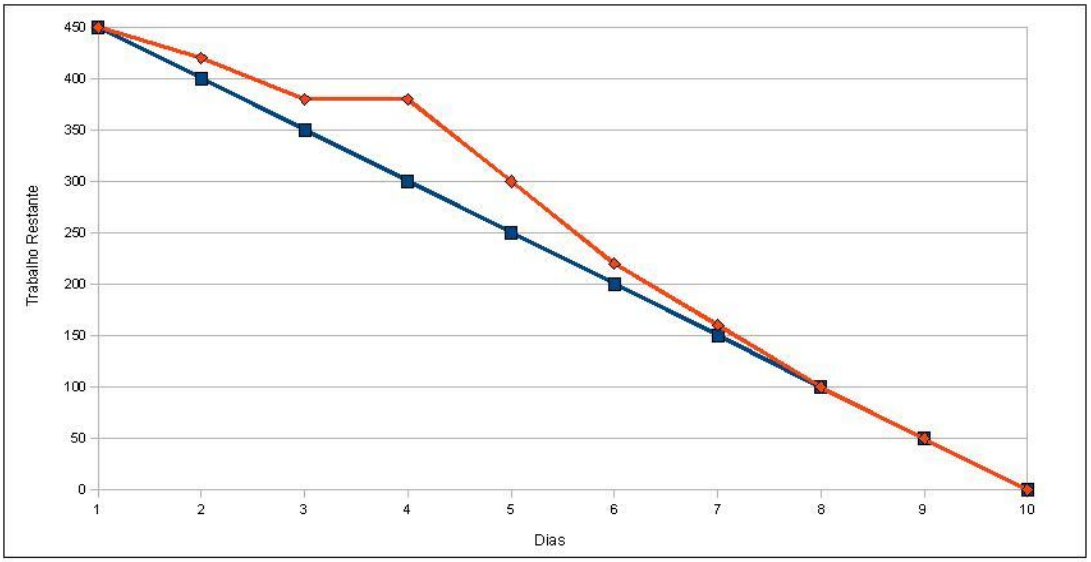


Figura 7. *Burndown Chart*

Esse tipo de gráfico é ajustado conforme a necessidade da equipe, podendo representar o trabalho restante que é visto no eixo *Y* em forma de horas, pontos por estórias, dias ou como a equipe julgar mais adequado (CASTRO 2009).

Segundo Castro (2009) o gráfico de *Burndown Chart* mostra de forma rápida e eficaz o rendimento da equipe atuando como um agente motivador na mesma.

3.2.1.1.8 Planning Poker

No *Scrum* definir estimativas é um processo importante que envolve toda equipe para sua realização. A métrica *Planning Poker* tem como objetivo ajudar a equipe a definir estimativas de forma simples e eficaz.

O *Plannig poker* funciona de forma que cada integrante da equipe recebe uma sequência de cartas com os valores 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100 e uma carta com um ponto de interrogação,

Em seguida, é definido um requisito descrito no *Product backlog* e cada membro da equipe sugere uma carta com o valor correspondente ao tempo de conclusão do mesmo e apresentam a carta com a face virada para baixo, em seguida todas as cartas são viradas simultaneamente. Após as cartas viradas, é definido um valor em comum para ser a estimativa do requisito (CRISP, 2010).

Nos casos em que as cartas apresentarem valores muito dispersos, como por exemplo, um membro dar valor 3 à estória e outro membro valor 20, é necessário que a equipe discuta rapidamente e faça uma nova rodada para estimar novamente a estória.

Entre as cartas, existem os valores especiais, sendo o valor *0* e o símbolo *?*, o valor *0* tem o objetivo de demonstrar que a estória já está ou leva poucos minutos para ser finalizada, e o símbolo *?* representa que o membro da equipe não tem nenhuma idéia sobre a estimativa da estória.

Segundo Crisp (2010) se o símbolo *?* aparecer muitas vezes durante as jogadas, a equipe precisa discutir mais sobre as estórias disseminando melhor o conhecimento.

3.2.1.2. Papéis do *Scrum*

No *Scrum* como em outros métodos, existem papéis e responsabilidades que são desempenhados durante o processo de construção do *software*, pode-se citar como papéis do *Scrum*: *Scrum Master*, *Product Owner*, Equipe *Scrum*, Cliente, que serão descritos a seguir:

3.2.1.2.1 Scrum Master

O *Scrum Master* conforme destaca Schwaber (2002) é responsável pelo sucesso direto do *Scrum*, sendo que suas principais atividades podem ser descritas da seguinte forma:

a. organizar as reuniões diárias;
b. representar o gerente do projeto e o técnico;
c. gerenciar todo o processo *Scrum* da organização;
d. remover os impedimentos do projeto.

Scrum Master pode atuar em mais de um projeto dentro de uma mesma organização, sempre buscando diminuir as inconformidades encontradas em cada um deles.

3.2.1.2.2 Product Owner

Tem o objetivo de defender os interesses do negócio tendo como principal tarefa, definir e liberar os itens do *Product backlog* para o *Sprint backlog* que será desenvolvido durante a iteração.

3.2.1.2.3 Equipe Scrum

Equipe *Scrum* também denominada “time”, é composta por no máximo sete pessoas sendo que cada membro da equipe possui características como:

a. ter o conhecimento técnico necessário sobre o processo do

desenvolvimento do *Software*;

b. serem auto-organizadas e gerenciarem seu próprio trabalho;

c. ter a capacidade de analisar, codificar e testar.

Uma equipe *Scrum* dentro do processo de desenvolvimento do *Software* possuí várias responsabilidades, pode-se citar como principais:

a. criação do *Sprint backlog* juntamente com o *Scrum Master* e o *Product Owner*;

b. desenvolvimento dos itens propostos no *Sprint*;

c. verificar os impedimentos que precisam ser retirados.

3.2.1.2.4 Cliente

O cliente como na grande maioria dos métodos ágeis, está presente em todo o desenvolvimento do projeto, participando em tarefas como: elaborar itens a serem desenvolvidos, verificando o progresso do projeto e avaliar a qualidade do produto.

3.2.1.3. Estrutura do *Scrum*

Zanattta (2004) define que a estrutura de funcionamento do *Scrum* é dividida em três etapas que são definidas a seguir, conforme o autor:

a. *pregame:* a fase inicial do *Scrum* é dividida em dois estágios que são: planejamento e arquitetura. O planejamento, consistindo em desenvolver as atividades iniciais para o *Product backlog* e na definição da entrega das versões, estimativas, custos e data das

realizações das tarefas. Na arquitetura é especificado a solução técnica dos itens propostos no *Product backlog* a serem desenvolvidos, definindo os itens que cada membro da equipe é responsável;

b. *game:* é onde se inicia o ciclo iterativo do desenvolvimento, realizando os *Sprints*, sendo que em cada *Sprint* são implementadas mais funcionalidades ao *software* até que o mesmo esteja finalizado. Nessa etapa também é feita a análise de riscos e a inclusão de novos itens ao *Product backlog*, se necessário.
c. *posgame:* é a etapa responsável pela entrega do resultado ao cliente final, algumas de suas tarefas são: documentação feita para o usuário, preparação do material para o treinamento, integração, etc.

3.2.2 Método XP

Criado no final da década de 90 por Kent Beck, o método *Extreme Programming* também conhecido por XP, é um método de desenvolvimento ágil que segundo Teles (2004, p. 21), é voltado para:

a. projetos cujos requisitos são vagos e podem mudar com freqüência;
b. desenvolvimento de sistemas orientados à objeto;
c. equipes pequenas preferencialmente até doze desenvolvedores;
d. desenvolvimento incremental (ou iterativo) onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do processo.

Conforme a afirmação de Beck (2000), o XP é um método baseado em regras ao invés de uma seqüência de processos.

Teles (2004) aponta que o método busca assegurar que o cliente tenha o máximo valor durante todo o processo de desenvolvimento do *software*, organizando um conjunto de práticas que atuam de forma que o cliente receba sempre um auto retorno do seu investimento.

3.2.2.1 Valores do XP

O XP é focado em quatro valores para obter um *software* com qualidade que são: simplicidade, comunicação, coragem e feedback. A seguir esses valores são descritos conforme Teles (2004) e Zanatta (2004):

a. simplicidade: o *software* deve ser o mais simples e inteligível possível, seu código deve ser claro e de fácil compreensão, com nomes simples, problemas complexos divididos em pequenas partes e sem duplicações de código;
b. comunicação: no XP é priorizado o diálogo, ou seja, a conversa face a face considerando o desenvolvimento em um ambiente que requisitos possam mudar constantemente, relatórios se tornam obsoletos rapidamente;
c. coragem: é necessário coragem para pedir ajuda para solução de um problema complexo, simplificar um código e buscar o relacionamento direto com o cliente, quando se tem a necessidade de lhe informar de alguma alteração nos prazos, ou seja, é preciso ousadia para tomar a atitude certa mesmo em momentos que ela não é a mais popular.
d. feedBack: no XP o *feedback* é gerado a cada iteração ou *reasile.* O cliente recebe partes funcionais do *software*, avalia se o mesmo está

realmente atendendo as suas necessidades e repassa essa a informação aos desenvolvedores; esse mecanismo faz com que o cliente conduza o desenvolvimento diariamente, garantindo que a equipe direcione o foco do trabalho corretamente.

3.2.2.2 Atividades do XP

O desenvolvimento de um *software* utilizando o método XP, é dividido em quatro práticas que são: planejamento, *design*, testes e codificação, sendo que cada uma dessas práticas contém uma série de atividades que devem ser seguidas, (ZANATTA 2004).

A seguir é descrito conforme Teles (2004), uma série de atividades dispostas entre as quatro práticas citadas acima:

a. cliente presente: o XP trabalha focado em conduzir o desenvolvimento do projeto a partir do *feedback* do cliente, fazendo com que o mesmo participe ativamente de todo o processo de construção do *software*, viabilizando e simplificando vários aspectos, principalmente a comunicação;
b. jogo do planejamento: é feito no início de cada iteração através de uma reunião onde o cliente avalia as funcionalidades que serão implementadas, e prioriza as que vão ser executadas em uma próxima iteração. Essas funcionalidades são descritas em cartões chamados de estórias. Durante o planejamento, as estórias são estimadas para que o cliente possa conhecer o custo de cada implementação; essa estimativa é feita a partir de uma unidade especial denominada ponto.

A unidade especial ponto pode variar conforme o desempenho da equipe durante o desenvolvimento do *software*, sempre considerando a velocidade com que a equipe foi capaz de implementar na iteração anterior;

c. *stand up meeting:* é uma reunião feita pela equipe de desenvolvimento toda manhã, com o objetivo de avaliar o trabalho que foi executado no dia anterior e priorizar o que será desenvolvido no dia atual. Essa reunião recebe o nome de *Stand Up Meeting* que traduzido para o português, significa “reunião em pé”;
d. programação em par: o desenvolvimento das funcionalidades no XP ocorre em pares, ou seja, diante de cada computador existem dois desenvolvedores que produzirão um mesmo código. Dessa forma o código é revisado constantemente quando é construído. A técnica também é interessante pelo fato dos desenvolvedores se completarem tornando mais fácil a solução de problemas com idéias inovadoras;
e. desenvolvimento guiado pelos testes: no desenvolvimento guiado pelos testes são definidos testes para cada funcionalidade antes mesmo de serem codificadas, fazendo com que os desenvolvedores se aprofundem mais nas necessidades dos clientes, se preocupem com as interfaces externas dos métodos e suas classes, antes de codificá-los e possam contar com uma série de testes que possam ser aplicados durante a codificação do sistema;
f. *refactoring:* para que o *software* tenha uma manutenção mais simples, clara e objetiva, frequentemente a equipe de desenvolvimento tem a necessidade de alterar partes do *software*. Essa alteração é feita por

meio da prática *Refatoring*, que consiste em alterar uma parte do código sem modificar as funcionalidades que são implementadas. No *Refatoring* são utilizados fortemente os testes descritos anteriormente, para garantir que nenhuma funcionalidade implementada foi alterada;

g. código coletivo: a codificação no XP é coletiva, ou seja, os desenvolvedores têm acesso a qualquer parte do código sem restrições, podendo alterá-lo ou utilizar a prática de *Refatoring* para torná-lo mais legível se julgarem necessário, fornecendo dessa maneira, outro mecanismo de revisão de código além de tornar o processo de codificação mais ágil;
h. código padronizado: o fato do código ser coletivo no XP, implica na necessidade de padrões de codificação para que os desenvolvedores possam manipular qualquer parte do *software* de forma mais rápida e segura, além de permitir que o código fique mais homogêneo e de fácil manutenção;
i. *design* simples: para atingir o *feedback* rápido do cliente, a equipe tem que ser ágil no desenvolvimento, fazendo com que os desenvolvedores optem por um *Design* simples, ao invés de criar generalizações de código com o objetivo de se preparar para novas funcionalidades. A equipe conta com a premissa que serão capazes de incorporar novas funcionalidades ao código, se apoiando em práticas do XP como *refatoring*, testes;
j. metáfora: é estabelecido uma linguagem comum entre a equipe de desenvolvimento e o cliente, para que se passa transmitir de forma simples idéias complexas com o obejtivo de construir um *desgin*

simples;

k. ritmo sustentável: Com o objetivo de ter um *software* com maior qualidade, o XP recomenda que a equipe envolvida no processo de desenvolvimento, trabalhe apenas 8 horas por dia, evitando fazer horas extras, pois para o XP é essencial estar descansado a cada manhã;

l. integração contínua: sempre que uma nova funcionalidade é adicionada ao *software*, é possível que ela possa alterar outra funcionalidade que já estava implementada. Para que se possa garantir que essa integração ocorra de forma harmoniosa ao *software*, o XP utiliza a prática de integração contínua, que consiste em integrar parte código ao restante do sistema várias vezes ao dia. Cada vez que é feita uma integração, também são feitos todos os testes para assegurar que a mesma foi concluída de forma satisfatória, e descobrindo eventuais erros para que possam ser resolvidos, facilitando a correção de problemas e evitando futuras dores de cabeça;

m. *releases* curtos: o XP trabalha com *releases* curtos, colocando um conjunto de funcionalidades em produção para o cliente rapidamente, fazendo com que ele se beneficie do *software* logo no início do desenvolvimento e a equipe tendo o feedback rápido do cliente.

3.3 METODOLOGIA DE DESENVOLVIMENTO TRADICIONAL

Nesta sessão inicialmente será abordada à metodologia de desenvolvimento tradicional RUP, onde é apresentada uma breve historia do surgimento do método e após será feita uma descrição geral de seu funcionamento e sua integração com métodos ágeis.

3.3.1 Método RUP

No fim da década de 1980 a empresa *Objectory* AB que pertencia a Ivar Jacobson, definiu um processo para desenvolvimento de *software* baseado em documentos UML chamado *Objectory* v1.0. Passados alguns anos, em 1995 a empresa foi vendida para *Rational* Corporation, que integrou seus produtos ao Objectory v1.0 criando o Rational Objectory Process 4.0 para posteriormente em 1998 lançando o Rational Unified Process 5.0, que foi adquirido pela IMB em 2003 através da compra da Rational Corpotarion (BASSI 2008).

O processo de desenvolvimento RUP, foi criado originalmente para se ajustar à projetos de diversos tamanhos, domínios de aplicação e tecnologias utilizadas para codificação (KRUCHTEN, 2001).

Segundo Cintra (2006) o RUP descreve:

a. **papéis ou perfis de trabalho (quem):** define quem é responsável por cada uma das tarefas;
b. **artefatos (o que):** representa o que será implementado durante o processo de desenvolvimento;
c. **atividades (como):** define como os representantes de cada papel trabalham para atingir os objetivos do projeto e construir o *software*;
d. **fluxo de atividades (quando):** orientam a execução das atividades, definindo quando elas devem iniciar.

Conforme afirmação de Cintra (2006) o RUP define quatro fases, nove disciplinas, mais de 40 papéis e 100 artefatos. Por ser um produto comercial possui mais de 1.000 páginas de orientação e conta com processos adicionais à versão padrão. Devido a essa grande documentação é considerado por muitos autores, um

método pesado, porém o RUP disponibiliza mecanismos de customização, onde é possível definir um sub-conjunto de artefatos, papéis e atividades que realmente são necessários.

A seguir são definidas as quatro fases do RUP conforme Cintra (2006):

a. **iniciação ou Concepção:** nessa fase é apresentado uma visão geral do produto, é feito uma abrangente análise dos requisitos do sistema e determinado o escopo do projeto;
b. **elaboração:** é responsável pelo planejamento das atividades e recursos necessários, a especificação detalhada dos requisitos e a arquitetura do sistema;
c. **construção:** o foco dessa fase é a implementação do *software*, sendo responsável pela evolução do mesmo e de sua arquitetura até que ele esteja finalizado para entrega ao cliente;
d. **transição:** nessa fase o sistema é repassado aos usuários, são resolvidas as questões de empacotamento, entrega, treinamento, suporte e manutenção.

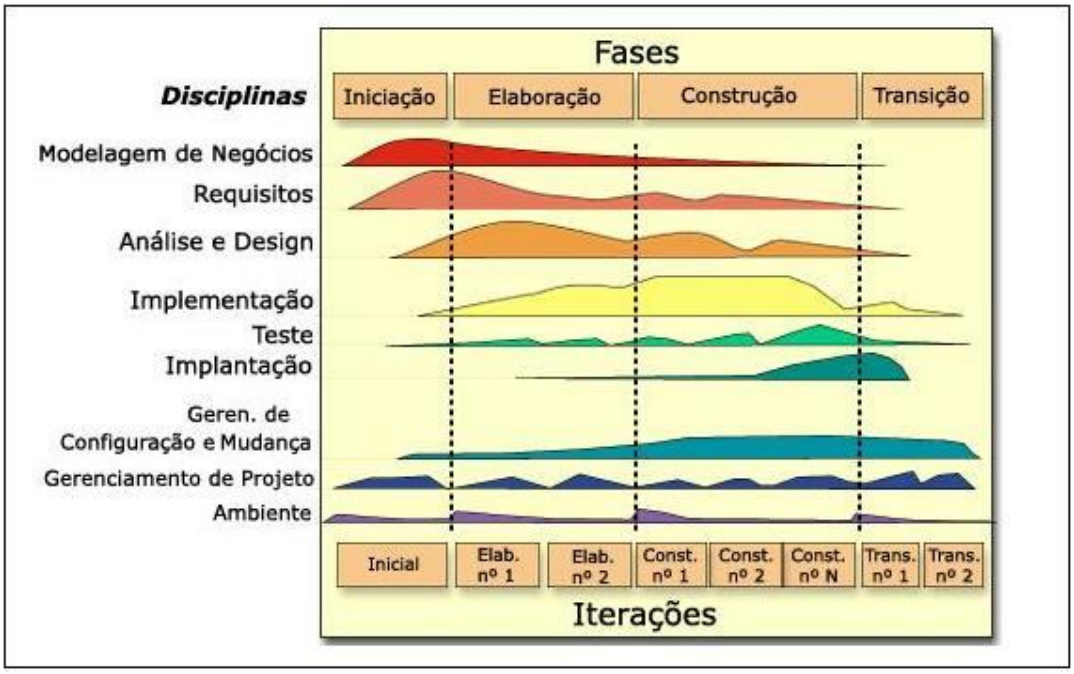


Figura 8. Fases do RUP
Fonte: PEREIRA, E (2005, p.36).

Pode-se ver na Figura 8 a estrutura do RUP, demonstrando no eixo horizontal os seus aspectos dinâmicos com suas quatro fases, e no eixo vertical, seus aspectos estáticos sendo as nove disciplinas.

3.3.1.1 Integração RUP/Métodos ágeis

Conforme foi visto anteriormente, o RUP é processo para desenvolvimento de *software* que pode ser customizado conforme a necessidade da equipe. Segundo Kruchten (2001) o método pode ter um claro alinhamento com princípios e práticas da modelagem ágil.

Algumas práticas da modelagem ágil já são incorporadas ao RUP com certa facilidade, como: inserção de S*takeholders* em papéis de modelagem, criação de protótipos para execução de modelos, adoção de apenas um subconjunto de

artefatos por projeto, porém, outras já se mostram um pouco desafiadoras como práticas de abandono de modelos intermediários de projeto.

Dentre as nove disciplinas que o RUP contém, três delas podem ser aplicadas de forma clara e objetiva, utilizando práticas da modelagem ágil, que são: Análise & Projeto, Modelagem de Negócio e Requisito.

Conforme Cintra (2006), diversas orientações dos métodos ágeis podem ser utilizadas para executar atividades da engenharia de requisitos do RUP. A seguir são descritas algumas dessas orientações:

a. estórias de usuário: é detalhado apenas o suficiente para permitir que o desenvolvedor possa estimar e implementar a funcionalidade descrita;
b. conversação: o conteúdo da versão é definido por meio da conversação ao longo do tempo, geralmente durante o processo;
c. confirmação: é feito através de testes de aceitação feitos pelo cliente;
d. documentação: os requisitos são documentados seguindo o princípio de simplicidade, gerando apenas documentos necessários;
e. stakeholders: participam ativamente definindo, levantando e priorizando os requisitos a serem implementados.

3.4 METODOLOGIA DE DESENVOLVIMENTO ÁGIL

A sessão a seguir tem como objetivo apresentar o método de desenvolvimento ágil Open UP, descrevendo o seu surgimento e apresentado alguns dos seus princípios básicos.

3.4.1 Open UP

Criado a partir do RUP, o método OpenUp também de propriedade da IMB[2] tem como objetivo ser um versão ágil do RUP. Segundo Santos (2010) o OpenUp é um processo mínimo, completo e extensível que está em conformidade com os princípios do manifesto ágil.

O OpenUp por ser um método originário do RUP, utiliza várias práticas contidas nele além de apresentar a mesma distribuição de fases, que consiste em iniciação, elaboração, construção e transição, sendo que cada fase continua com os mesmo objetivos definidos no RUP, porém, utiliza práticas ágeis para atingi-los.

Conforme afirmação de Cunha (2007), o método OpenUp é baseado em quatro princípios fundamentais que são: colaboração, equilíbrio, foco e evolução. Seu conteúdo é organizado em quatro grandes áreas também conhecidas como sub-processos, influenciados pelos princípios vistos anteriormente, que são: Colaboração & Comunicação, Propósito, Gerenciamento e Soluções.

O desenvolvimento no método OpenUp conta com oito atividades desempenhadas durante as interações em suas respectivas fases do ciclo, que são:

a. *initiate project* : inicio do projeto;
b. *define architecture* :definição da Arquitetura;
c. *develop solution*: desenvolvimento de Soluções;
d. *determine architectural feasibilty* : determinação da arquitetura da fase;
e. *manege interation:* gerenciamento da Iteração;
f. *manage requirements* : gerenciamento de requisitos;
g. *ongoing tasks*: tarefas que estão sendo executadas;

2 IBM: Empresa americana da área de Tecnologia da informação

h. *validade build*: validar pacote de entrega do sistema. mais conhecido como *build*.

O método também se baseia em sete tipos de atores que são: Analista, *Any Role*, Arquiteto, Desenvolvedor, Gerente de Projeto, *Stakholder* e *Tester*, sendo responsáveis por gerar os artefatos necessários para o projeto.

4 *FRAMEWORK* EPF *COMPOSER*

EPF *Composer* é uma ferramenta de código aberto criada pela IBM, desenvolvida em Java[3], utilizado para criar, adaptar e publicar métodos para desenvolvimento de *software* (HAUMER, 2010).

A ferramenta procura levar ao profissional uma sólida base de conhecimento, disponibilizando processos pré-definidos, que podem ser utilizados da maneira que forém determinados ou adaptados de acordo com as características de cada projeto.

Segundo Haumer (2010) a separação do conteúdo é feita por meio de nomenclaturas, atores, tarefas e produtos de trabalho sendo organizados através de sequências semi-ordenadas.

Uma grande vantagem da utilização desta ferramenta é a possibilidade de disponibilizar de forma rápida e fácil todo o conteúdo criado em html.

3 Java: Linguagem de programação orientada a objeto

5. TRABALHOS CORRELATOS

Para realização deste trabalho de pesquisa, se fez necessária a leitura e avaliação de alguns trabalhos com o propósito semelhante ao objetivo dessa inquirição, porém com o seu foco voltado para outro objetivo, como são descritos a seguir.

5.1 APLICAÇÃO DE MÉTRICAS DE *SOFTWARE* NO MÉTODO *SCRUM* DA METODOLOGIA ÁGIL

Trabalho de Conclusão de Curso de Mateus Luiz Gamba para obtenção do grau de Bacharel em Ciência da Computação, em 2009, pela Universidade do Extremo Sul Catarinense de Criciúma no Estado de Santa Catarina.

O trabalho consiste na aplicação de métricas de *software* com intuito de se obter estimativas de prazos de tempo de desenvolvimento do método *Scrum* da metodologia ágil. Foi realizado um estudo de caso na empresa Webmais Sistemas para validar as métricas definidas. (GAMBA, 2009)

5.2 X*SCRUM*: UMA PROPOSTA DE EXTENSÃO DE UM MÉTODO ÁGIL PARA GERÊNCIA E DESENVOLVIMENTO DE REQUISITOS VISANDO ADEQUAÇÃO AO CMMI

Dissertação de Alexandre Lazaretti Zanatta para obtenção do grau de Mestre em Ciências da Computação em 2004, pela Universidade Federal de Santa Catarina.

A dissertação apresenta uma análise comparativa dos métodos ágeis XP, *Scrum*, FDD e DSDM, segundo as perspectivas do modelo CMMI, nas áreas de processo Gerenciamento de Requisitos e Desenvolvimento de Requisitos. Sendo utilizado o método *Scrum* para uma proposta extensão denominado *xScrum* visando a resolução de lacunas encontradas na análise do método (ZANATTA, 2004).

5.3 A IMPLANTAÇÃO DE UM PROCESSO DE ENGENHARIA DE REQUISITOS BASEADO NO PROCESSO UNIFICADO DA RAIONAL (RUP) ALCANÇANDO NÍVEL 3 DE MATURIDADE DA INTEGRAÇÃO DE MODELOS DE CAPACIDADE E MATURIDADE (CMMI) INCLUINDO A UTILIZAÇÃO DE PRÁTICAS DE MÉTODOS ÁGEIS

Dissertação de Caroline Carbonell Cintra para obtenção do grau de Mestre em Ciências da Computação em 2006 pela Universidade do Rio Grande do Sul.

O trabalho descreve um processo de engenharia de requisitos que está em conformidade com as áreas de processo do CMMI, Gerência de Requisitos e Desenvolvimento de Requisitos baseados no modelo de desenvolvimento RUP. Sendo definido um processo de engenharia de requisitos baseado em abordagens diferenciadas e aplicado em uma organização especifica, com a intenção de avaliar a validar as abordagens definidas, o trabalho ainda propõe uma melhora no processo na questão de eficiência com a utilização de métodos ágeis (CINTRA, 2006).

6 MODELO DE DESENVOLVIDO

Todo o modelo criado foi documentado por meio de um website, dessa forma facilitando a busca e a visualização do conteúdo descrito, sendo estruturado de maneira com que possibilite ao usuário diferentes modos de adquirir a informação desejada, podendo ser por meio de fluxo de etapas e atividades ou links, onde o usuário pode navegar em busca da informação.

Na Figura 9 é apresentado o fluxo das etapas do modelo no website.

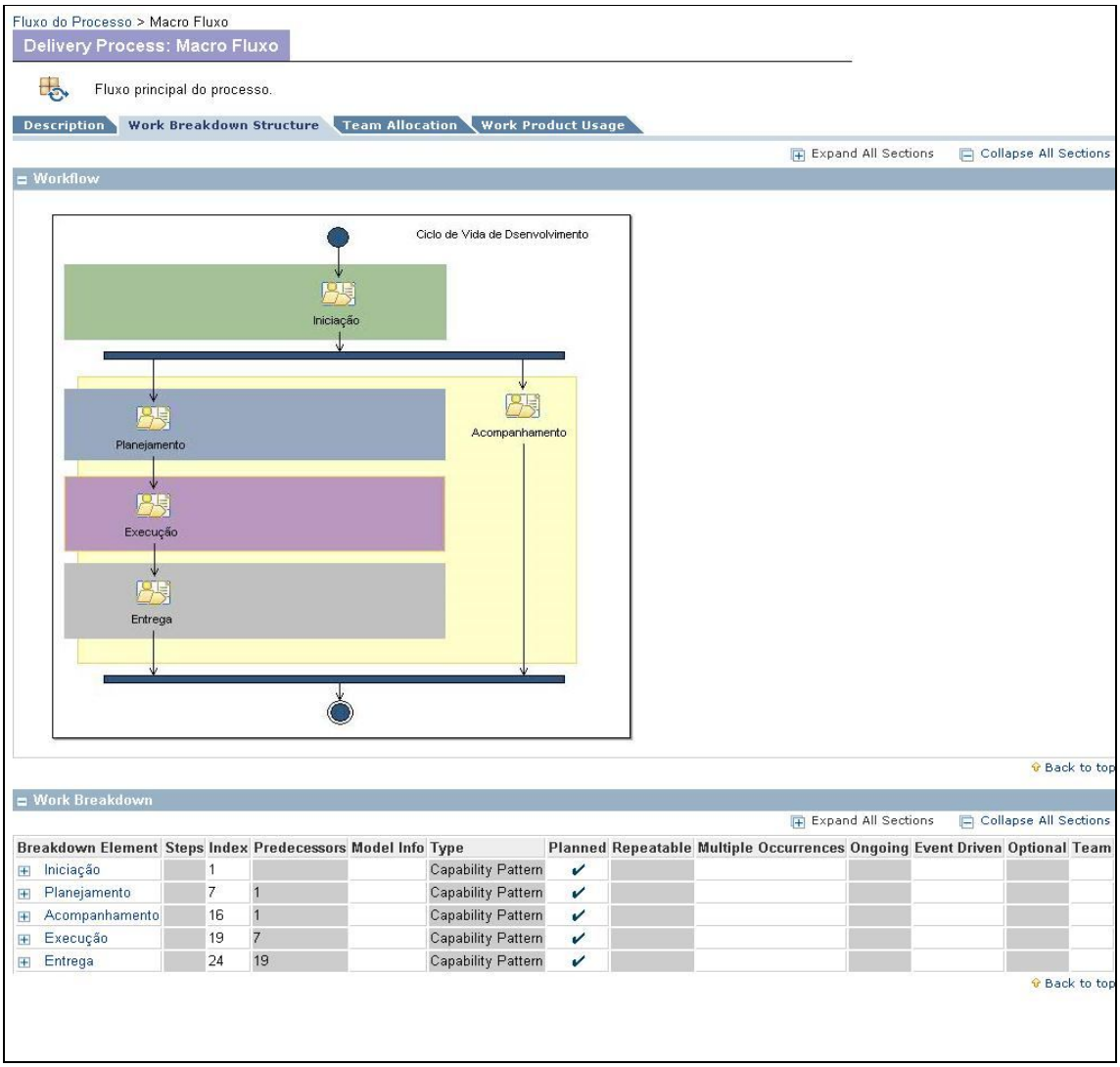


Figura 9. Fluxo de Etapas

Cada etapa do modelo remete ao usuário a um fluxo de tarefas, onde é descrito o trabalho que deve ser executado na aplicação do processo. Uma representação deste fluxo pode ser observado na Figura 10.

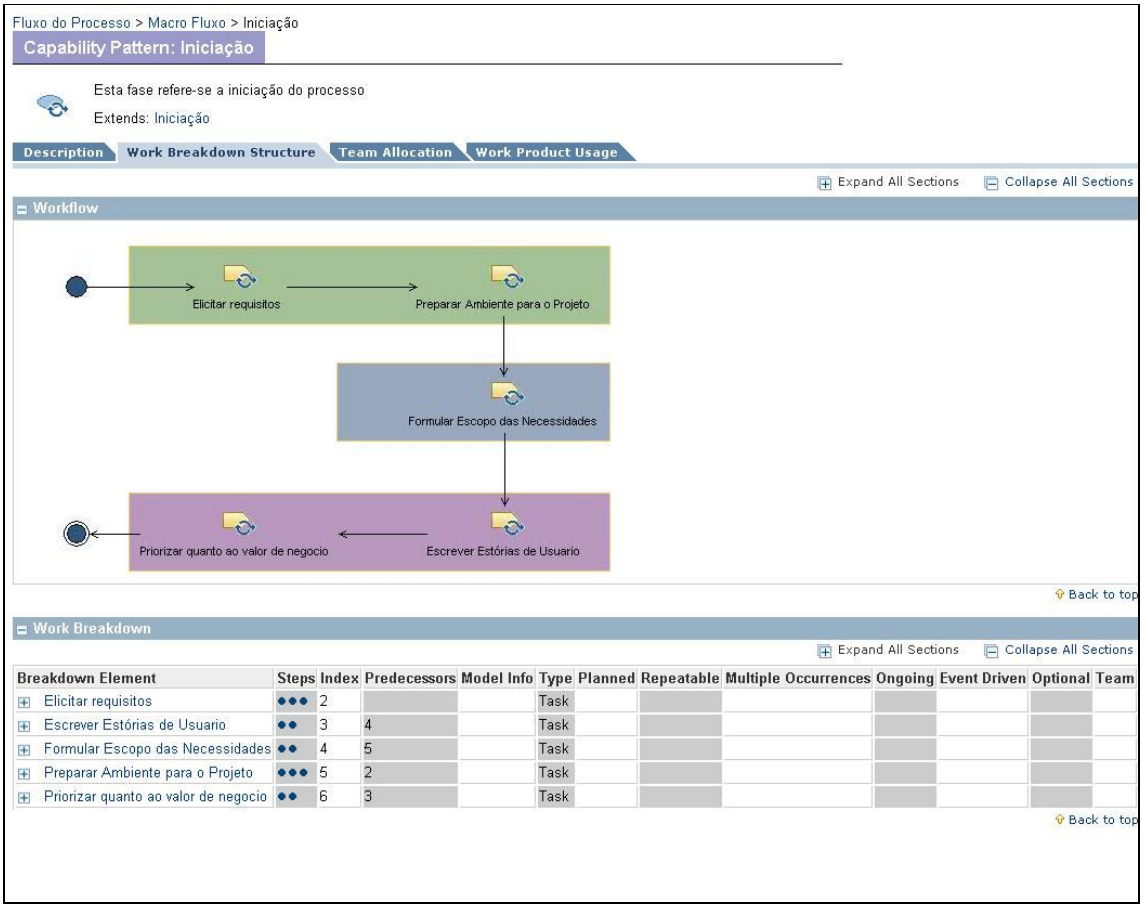


Figura 10. Fluxo de tarefas (Iniciação)

Cada item do fluxo de tarefas leva o usuário uma tarefa a ser desenvolvida. Ao clicar na tarefa desejada o usuário pode ter acesso a sua descrição, como poder ser vista na Figura 11.

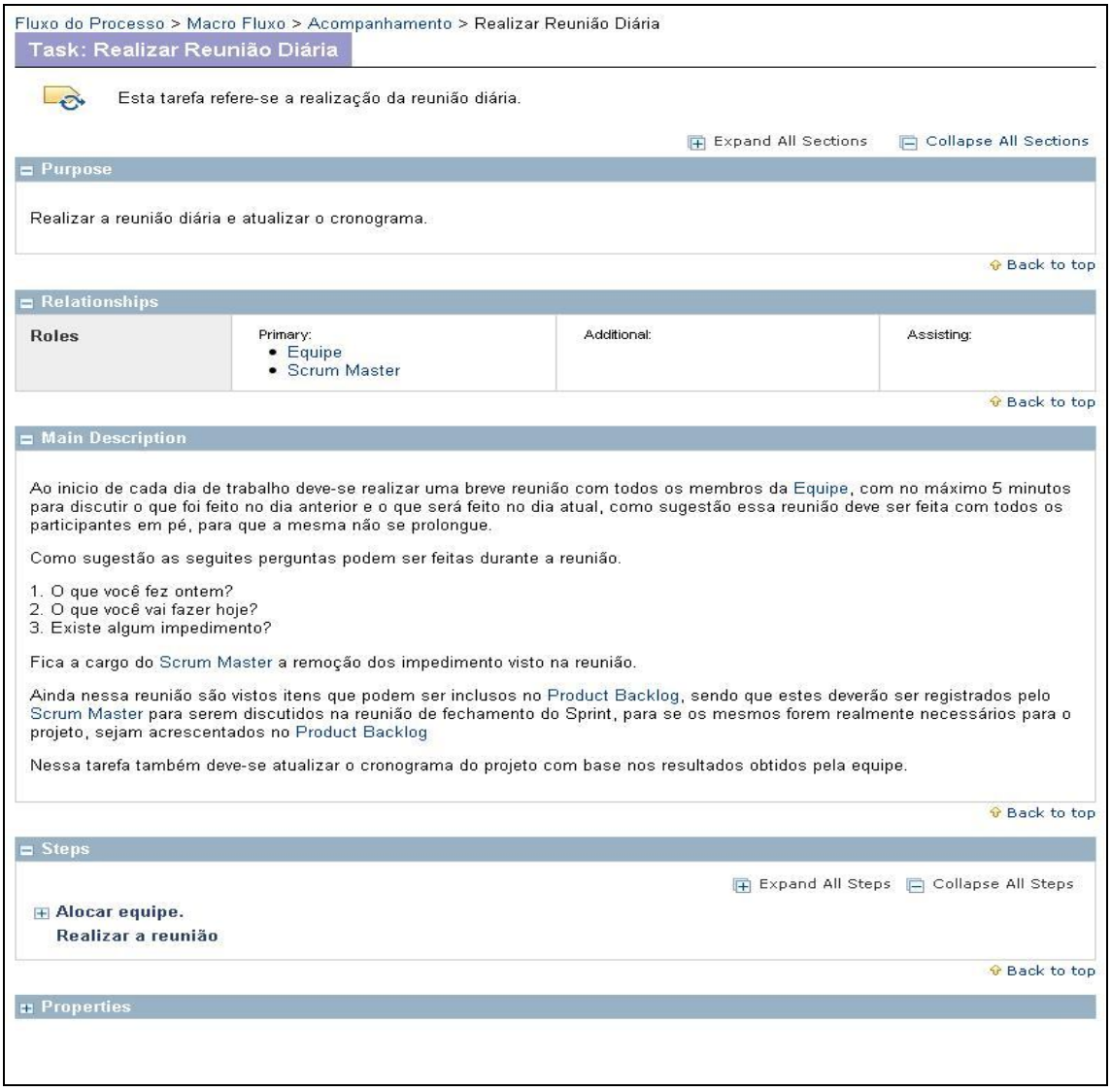


Figura 11. Descrição de uma tarefa

Cada tarefa descrita no modelo, contem definidos seus objetivos, passos para conclusão, descrição principal e relacionamento com outros itens do modelo, como papeis, artefatos e demais tarefas.

Os papeis a serem desempenhados no modelo serão os mesmos utilizados na metodologia de desenvolvimento ágil *Scrum*, que são: *Scrum Master*, *Product Owner* e Equipe, esses papeis foram escolhidos, pois se adéquam melhor a maioria das empresas de pequeno e médio porte.

Dessa forma o *Scrum Master* irá atuar no modelo como responsável pela gerência do projeto, tendo como principais atribuições a organização de reuniões, a remoção de impedimentos do projeto e a identificação de requisitos junto ao *Product Owner*.

O *Product Owner* no modelo pode ser um cliente da empresa ou um membro da empresa que tem contato direto com o cliente final do sistema, sendo de responsabilidade dele, o fornecimento dos requisitos do sistema, acompanhamento do projeto e o *feedback* das funcionalidades do sistema entregues ao longo de desenvolvimento até a conclusão do projeto.

A equipe de desenvolvimento assumira o papel denominado Equipe no modelo e conta como principais responsabilidades: a codificação dos itens propostos no *Product backlog*, a criação do *Sprint backlog* juntamente com o *Scrum Master* e o *Product Owner* e os testes de validação das estórias de usuário.

O modelo também define as etapas do desenvolvimento, sendo elas: iniciação, planejamento, execução, acompanhamento e entrega.

Na etapa de iniciação o modelo tem como objetivo a descrição de tarefas responsáveis por: todo o desenvolvimento inicial dos requisitos do sistema, formulação do escopo das necessidades e priorização dos requisitos quanto ao seu valor de negocio.

A escrita das funcionalidades no processo descrito no modelo será feita através de estórias de usuário, que é uma técnica utilizada na metodologia *Scrum*.

Na etapa de planejamento do projeto, o modelo descreve como responsabilidade, a definição das estimativas de tamanho das estórias de usuário, onde será utilizada a métrica *planing poker* para estimativas de tamanho, sendo que

nesta técnica é utilizado um baralho especial para que seja possível a definição das estimativas.

Ainda nesta etapa o modelo descreve tarefas para criação de vários artefatos provenientes de modelos tradicionais e ágeis como cronograma, matriz de responsabilidades, *product backlog* e *sprint backlog*.

O processo descrito ainda conta com a etapa de execução onde foi definas tarefas com objetivo de codificar e validar as estórias de usuário do *product backlog*, de forma ágil, descrevendo todo processo de codificação que será realizado em ciclos de tempo denominado *sprints*, seguindo a metodologia ágil *Scrum*.

Para completar o ciclo do desenvolvimento, o modelo conta com as etapas de acompanhamento e entrega que são responsáveis por definir tarefas como, a reunião de acompanhamento diário, homologação e liberação de *builds* para o cliente.

Uma descrição detalhada de cada etapa, seus papeis, tarefas, artefatos e templates podem ser observados na publicação do modelo proposto, bem como, todos os fluxos de etapas e tarefas descritos.

6.1 METODOLOGIA

Para criação do modelo proposto inicialmente, foi realizado todo levantamento bibliográfico necessário para que se tornasse possível o estudo de metodologias ágeis e tradicionais, ciclos de vida de desenvolvimento de *software* e ferramentas para o desenvolvimento de projetos de *software* entre outros.

Toda criação do modelo foi executado na ferramenta EPF *Composer*, que

pode ser encontrada no site http://www.eclipse.org, e adquirido gratuitamente. Essa ferramenta foi escolhida por motivos como: possibilitar ao usuário uma ampla visualização do conteúdo; disponibilidade de criação de fluxos de atividades; possibilidade de relacionar papeis; artefatos e tarefas entre si e publicação de todo o conteúdo criado em um website além de muitas outras facilidades da ferramenta.

O aprendizado do funcionamento da ferramenta EPF *Composer* iniciou-se junto com a criação do modelo, sendo que ao concluí-lo também foi concluído o aprendizado da ferramenta.

Abaixo é possível visualizar por meio da Figura 12 como é a criação de uma tarefa na ferramenta EPF *Composer*.

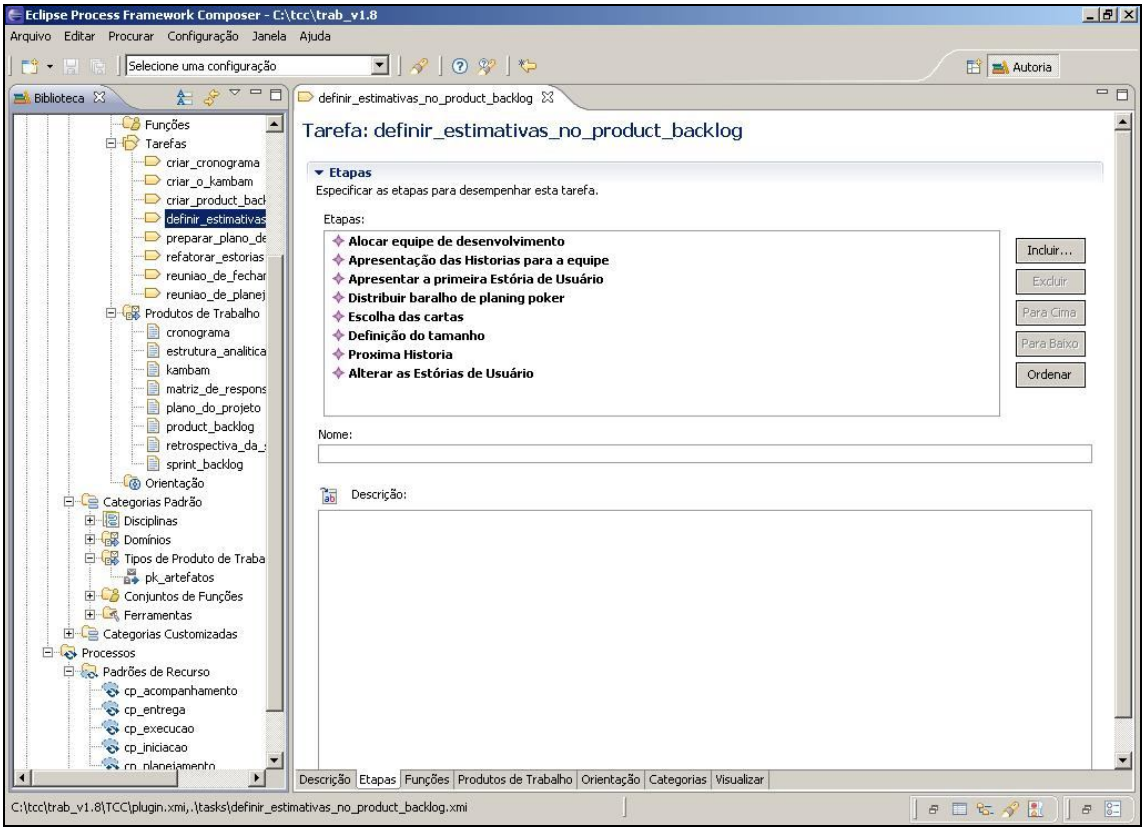


Figura 12. EPF *Composer*

Todas as técnicas descritas no modelo foram definidas com base nos estudos realizados na fase de estudo teórico do trabalho e por meio das orientações

com um engenheiro de *software*, onde, no modelo, foram criadas tarefas, artefatos, papeis e *templates* divididos em etapas de desenvolvimento que se relacionavam entre si. Nesse processo, foram criados três papeis e utilizados em todas as etapas do modelo, isso se dá, pelo fato, da abordagem ágil ser adotada para a definição do modelo proposto.

Todos os artefatos descritos contêm seus *templates* no formato PDF, sendo que esse formato foi definido pela fácil visualização e por ocupar um tamanho em disco relativamente pequeno.

Como forma de validação, o próprio modelo foi utilizado para o desenvolvimento deste trabalho, possibilitando a aplicação das tarefas descritas e a criação dos artefatos do projeto entre outras orientações do processo, em um projeto de trabalho real.

Apenas algumas tarefas do modelo não puderam ser desempenhadas, pois como o modelo foi aplicado no desenvolvimento de um trabalho de conclusão de curso e foi executado por apenas uma pessoa, assim nem todas as tarefas a serem desempenhadas se referiam a tarefas de um projeto de *software*, como por exemplo: a codificação de uma funcionalidade, porém o modelo pode-se adequar de forma satisfatória para o desenvolvimento.

Ao utilizar o modelo pra criação do trabalho, fez-se necessário definir uma ferramenta de produtividade para o gerenciamento do desenvolvimento do projeto, onde foi utilizada, como sugerida no modelo, a ferramenta Pronto. Esta ferramenta obteve um resultado bastante satisfatório no desenvolvimento projeto, e pode ser adquirida gratuitamente através do site http://pronto.bluesoft.com.br.

Para cada tarefa do trabalho foi criada uma estória de usuário na ferramenta Pronto, como pode ser visto na Figura 13 abaixo.

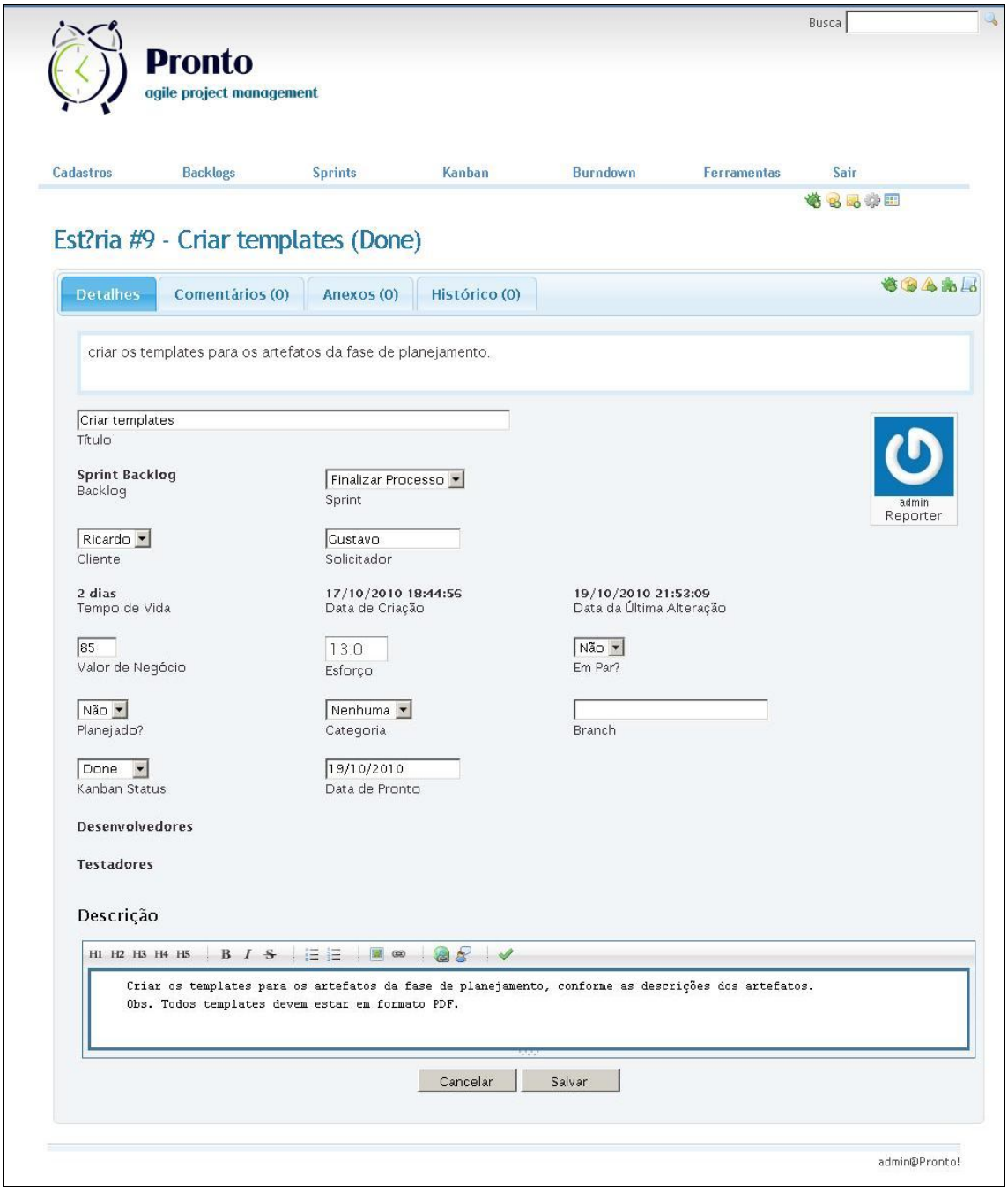
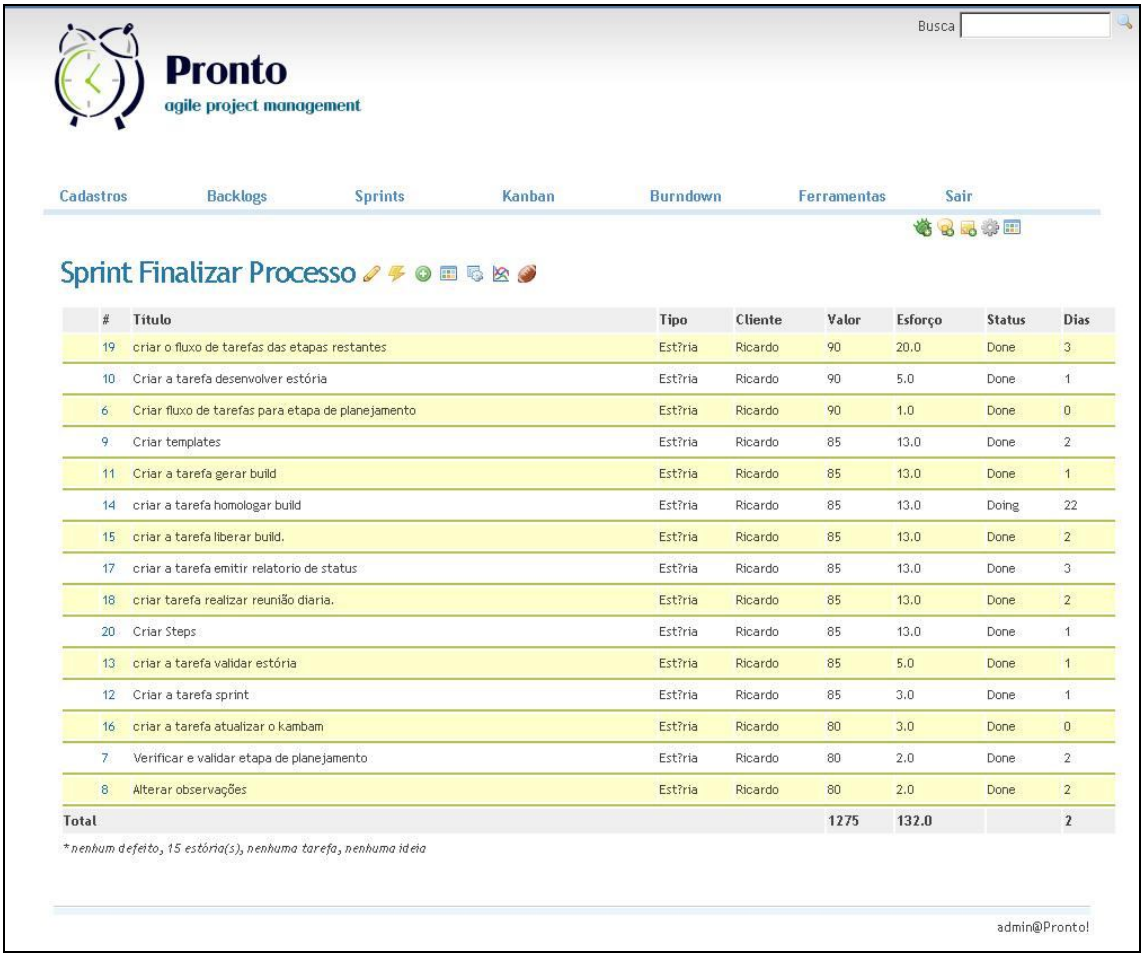


Figura 13. Estória de Usuário

Todas as estórias de usuário foram agrupadas em um *product backlog*, que ao longo do projeto foi recebendo mais estórias, como o desenvolvimento aconteceu através de *sprints* também foram criados *sprints backlog,* contendo todas

as estórias a serem desenvolvidas em cada *sprint*.

Abaixo é possível visualizar, por meio da Figura 14 um *sprint backlog* utilizado na elaboração do projeto.



#	Título	Tipo	Cliente	Valor	Esforço	Status	Dias
19	criar o fluxo de tarefas das etapas restantes	Est?ria	Ricardo	90	20.0	Done	3
10	Criar a tarefa desenvolver estória	Est?ria	Ricardo	90	5.0	Done	1
6	Criar fluxo de tarefas para etapa de planejamento	Est?ria	Ricardo	90	1.0	Done	0
9	Criar templates	Est?ria	Ricardo	85	13.0	Done	2
11	Criar a tarefa gerar build	Est?ria	Ricardo	85	13.0	Done	1
14	criar a tarefa homologar build	Est?ria	Ricardo	85	13.0	Doing	22
15	criar a tarefa liberar build.	Est?ria	Ricardo	85	13.0	Done	2
17	criar a tarefa emitir relatorio de status	Est?ria	Ricardo	85	13.0	Done	3
18	criar tarefa realizar reunião diaria.	Est?ria	Ricardo	85	13.0	Done	2
20	Criar Steps	Est?ria	Ricardo	85	13.0	Done	1
13	criar a tarefa validar estória	Est?ria	Ricardo	85	5.0	Done	1
12	Criar a tarefa sprint	Est?ria	Ricardo	85	3.0	Done	1
16	criar a tarefa atualizar o kambam	Est?ria	Ricardo	80	3.0	Done	0
7	Verificar e validar etapa de planejamento	Est?ria	Ricardo	80	2.0	Done	2
8	Alterar observações	Est?ria	Ricardo	80	2.0	Done	2
Total				**1275**	**132.0**		**2**

**nenhum defeito, 15 estória(s), nenhuma tarefa, nenhuma ideia*

Figura 14. *Sprint backlog* finalizar processo

Todas as estórias de usuário no decorrer do desenvolvimento, foram atualizadas diariamente em um *kambam* digital, onde foram separadas em estórias não iniciadas, iniciadas, prontas para testes, em testes e prontas. Abaixo pode-se observar por meio da Figura 15 o *kambam* utilizado no desenvolvimento.

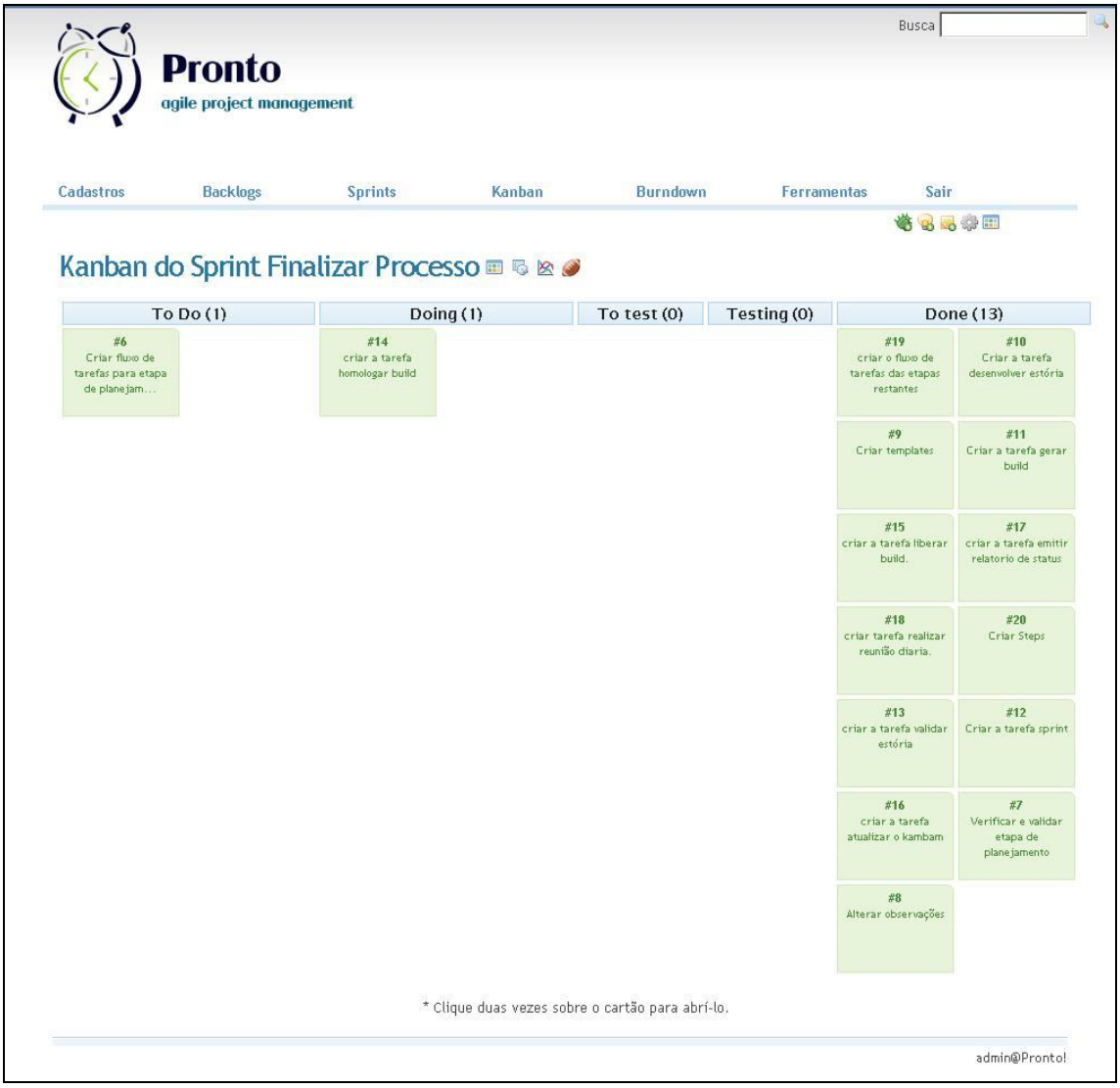


Figura 15. *Kambam* digital

Após todas as estórias de usuário criadas concluídas e validadas, o modelo foi exportado no formato HTML, possibilitando a sua publicação no formato de um website.

A abordagem da publicação do processo em HTML fomenta sua utilização, uma vez que, este, fica disponível para todos os membros da equipe de desenvolvimento consultar e evoluir o processo de acordo com suas necessidades.

6.2 RESULTADOS OBTIDOS

Obteve-se no decorrer deste trabalho, além do modelo proposto, uma melhor compreensão das metodologias ágeis XP, *Scrum* e a metodologia tradicional RUP, sendo que as mesmas foram utilizadas como base para criação do modelo.

Também pode-se compreender o funcionamento de ferramentas como o EPF *Composer* e Pronto que foram necessárias para a criação e validação do modelo criado.

Por meio deste trabalho, adquiriu-se também conhecimento sobre ferramentas de integração contínua que foram testadas e avaliadas para que fosse possível verificar, quais eram mais adequadas ao modelo, e dessa forma apresentá-las como sugestão.

Com a validação do modelo proposto sendo feito através da sua própria utilização no desenvolvimento do trabalho, pode-se verificar que o mesmo atingiu resultados satisfatórios quanto ao seu desempenho e controle das atividades.

Foram utilizadas todas as orientações possíveis do modelo para realização das atividades do trabalho, onde foram criados artefatos, realizadas tarefas e manipulado templates, seguindo o fluxo indicado na modelagem.

Todas as tarefas necessárias para criação do trabalho foram escritas na forma de estória de usuário, gerando um *product backlog* e alguns *sprints backlog* de estória, além de outros artefatos descritos no modelo, possibilitando o desenvolvimento do trabalho com maior agilidade e controle das atividades.

Abaixo, pode ser visualizado através da Figura 16 um *Burndown chart* obtido durante a elaboração do trabalho, mostrando os dias de maior e menor rendimento, bem como o rendimento ideal para o *sprint*.

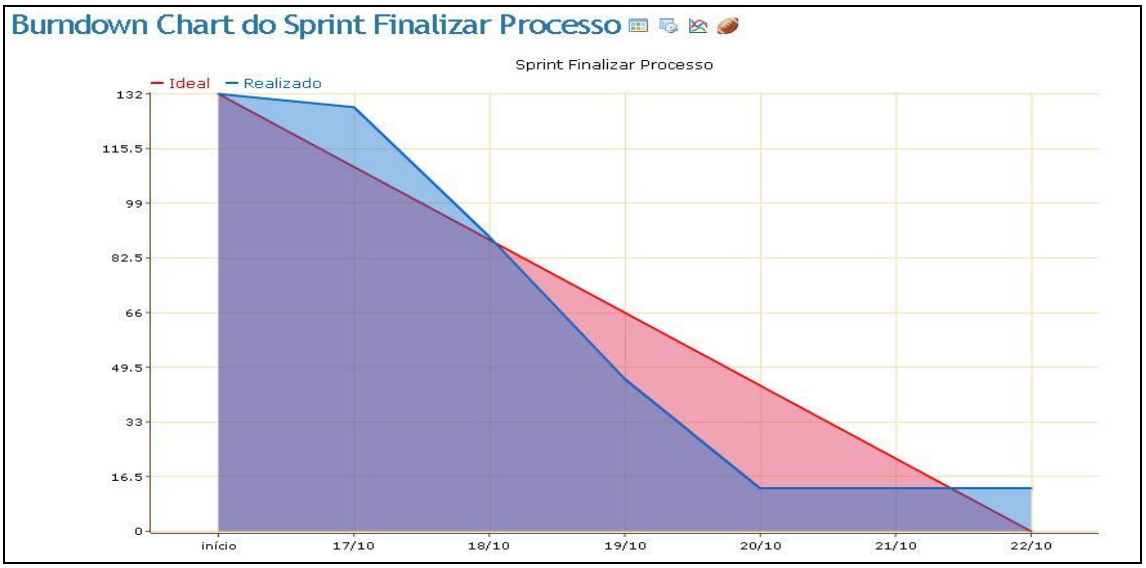


Figura 16. *Burndown chart* do *sprint*

A utilização do *Burndown chart* como ferramenta de acompanhamento do desempenho do projeto, foi interessante pelo fato de apontar os dias de menor rendimento, o que facilitou a identificação das tarefas que estavam gerando atrasos na elaboração do trabalho.

O uso de cronogramas com datas estipuladas para criação de determinadas tarefas, pode proporcionar a regularidade no desenvolvimento, sendo quase todas que foram iniciadas na data correta estipulada e concluída no prazo planejado.

Apenas algumas dificuldades se apresentaram no momento da validação do modelo, como não foi possível aplicá-lo com uma equipe de desenvolvimento em um projeto de *software*, algumas tarefas não puderam ser validadas como deveriam. Como por exemplo, a tarefa de definir as estimativas de tamanha das estórias, pois essa tarefa se baseava no conjunto da equipe para definir as estimativas.

Com a adoção do princípio proveniente do método RUP de descrever o modelo em tarefas, papeis, artefatos e fluxo de atividades, foi possível demonstrar claramente o funcionamento do modelo, que utilizou fortemente as atividades do método *Scrum*.

CONCLUSÃO

O desenvolvimento de projetos de *software* é uma atividade complexa e repleta de tarefas a serem executadas, por esse motivo existem metodologias de desenvolvimento de *software* que buscam trazer aos profissionais envolvidos nesse tipo de projeto, uma base sólida de conhecimento para que esses possam desempenhar suas tarefas da forma correta.

A metodologia de desenvolvimento ágil *Scrum*, pode ser considerada uma ótima alternativa para projeto de *software* de pequeno e médio porte, pois atua melhor com constantes mudanças nos requisitos através do *feedback* do cliente que está altamente envolvido no projeto. Também mostrou-se adaptável, pois suas atividades puderam ser desempenhadas em conjunto com algumas tarefas de metodologias, também ágeis como XP e tradicionais como RUP.

A metodologia tradicional RUP, tornou-se uma base pra criação do modelo, pois o mesmo foi elaborado com a definição de papeis, etapas, tarefas e artefatos que se relacionavam entre si. Também apresentou-se eficaz pela utilização de artefatos na questão de controle de responsabilidades, facilitando assim a identificação de impedimentos e a falta de comprometimento dos membros da equipe.

Após o estudo dos dois tipos de metodologia, foi possível verificar que a criação de um modelo baseado na integração do ágil com o tradicional, poderia trazer a agilidade e o controle necessário para projetos de pequeno e médio porte.

Como forma de documentação do modelo, foi definida a utilização de um website, onde todo o modelo é descrito, sendo que esta forma de documentação mostrou-se ideal para o conceito do modelo, pois torna ágil a visualização do

conteúdo e nos moldes de uma modelagem tradicional, definindo o processo por etapas, papeis, tarefas e artefatos.

A utilização de etapas, papeis, tarefas e artefatos para criação do modelo se apresentou como um forma de controle de responsabilidades e fluxo de tarefas pois indica como, quem, quando e o que será feito cada atividade no projeto.

Após a validação do modelo por meio da sua própria utilização na elaboração do trabalho, pode-se concluir que esse se apresentou de forma satisfatória quanto ao controle de prazos estipulados e do rendimento do trabalho, também se pode verificar que o modelo não apenas pode ser aplicado em projetos direcionados a área *software*, mas podendo também ser aplicado também em outros tipos de projetos.

Para validação do modelo se fez necessário a utilização de uma ferramenta denominada Pronto para gerenciar todo o processo, que se apresentou muito útil para o desenvolvimento, criando artefatos automaticamente e trabalhando muito bem com as estórias de usuário criadas na própria ferramenta.

Por ser um modelo genérico, que agrega técnicas de engenharia de *software* modernas e ágeis, o presente trabalho pode ser utilizado como base de processos para diversas organizações desenvolvedoras de *software* da área de tecnologia da informação, tendo estas que customizar e adequar as atividade e tarefas propostas neste modelo, a sua realidade. De acordo com os estudos realizados, a utilização das boas práticas reunidas no processo fruto deste trabalho, pode auxiliar significativamente no ganho de qualidade do desenvolvimento dos sistemas nas organizações, proporcionando o crescimento da qualidade dos produtos desenvolvidos e potencializando o crescimento financeiro nestas organizações.

Como sugestão de trabalhos futuros sugere-se:

- Validar o modelo proposto em um estudo de caso com a aplicação do processo em um projeto de desenvolvimento de *software*.
- Definir uma metodologia de gerenciamento apoiado por uma ferramenta automática que apóie a execução do modelo proposto;
- Ajustar o processo definido, de acordo com o modelo de referência para a Melhoria de Processo de *Software* Brasileiro MPS.BR

.

REFERÊNCIAS

Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. ***Agile Software Development Methods: Reviews and Analysis*. Espoo: VTT Publications, 2002**. Disponível em: <http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf>. Acesso em: 27 abr. 2010.

BASSI, Dairton Luiz. **Experiências com desenvolvimento ágil.** 2008. 154 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo.

BECK, Kent. **Extreme Programming:** Embrace Change. Addison Wesley. 2000.

BEZZERA, Eduardo. **Princípios de Análise e Projeto de Sistemas com Uml**. Rio de Janeiro: campus, 2002.

CASTRO, Flavio Steffens de. **Gráfico Burndown – Sugestão de uso.** Disponível em: <http://www.agileway.com.br/2009/08/18/grafico-burndown-sugestao-de-uso/>. Acesso em: 27 abr. 2010.

CINTRA, Caroline Carbonell. **A Implantação de um Processo de Engenharia de Requisitos Baseado no Processo Unificado da Rational (RUP) Alcançando Nível 3 de Maturidade da Integração de Modelos de Capacidade e Maturidade (CMMI) Incluindo a Utilização de Práticas de Métodos Ágeis.** 2006. 160 f. Dissertação (Mestrado) - Universidade do Rio Grande do Sul, Porto Alegre.

CRISP. **Planning Poker.** Disponível em: <http://www.crisp.se/planningpoker/>. Acesso em: 27 abr. 2010.

CUNHA, Carlos Eduardo Albuquerque da. **Utilizando OpenUp/Basic para Desenvolvimento de Aplicações WEB.** 2007. 69 f. Monografia (Graduação) - Universidade Federal de Pernambuco, Recife.

GAMBA, Mateus Luiza. **APLICAÇÃO DE MÉTRICAS DE *SOFTWARE* NO MÉTODO *SCRUM* DA METODOLOGIA ÁGIL.** 2009. 103 f. Monografia (Graduação) Universidade do Extremo Sul Catarinense, Criciúma.

HAUMER, P., **Eclipse Process Framework *Composer* Part 1: Key Concepts**. Eclipse, [S. l.], 15 dezembro 2005. Disponível em <http://www.eclipse.org/epf/general/ getting_started.php>. Acesso em: 10 maio 2010.

KINIBERG, Henrik. ***Scrum* e XP Direto das Trincheiras**: como nós fazemos *Scrum*. InfoQ. 2008. 147p.

KRUCHTEN, P. The Rational Unified Process: An Introduction. 2nd ed. Reading, MA: Addison-Wesley, 2001.

LINDA, Rising; NORMAN, Janoff S. **The SCRUM Software Development Process for Small Teams.** IEEE Software. AGO. 2000.

MANIFESTO para o desenvolvimento ágil de *software*. 2001. Disponível em: < www.manifestoagil.com.br >. Acesso em 13 mar. 2009

PEREIRA, Eliana Beatriz. **Uma proposta para adaptação de processos de desenvolvimento de *software* baseados no *rational unified process.*** 2005. 125f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio Grande do Sul

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

PRESSMAN, Roger S. **Engenharia de *Software***. São Paulo: Makron Books, 1995. 1056 p.

SANTOS, Sandra Sergi. Http://www.ibm.com/developerworks/br/rational/local /open_up/index.html. Disponível em: <http://www.ibm.com/developerworks/br/rational /local/open_up/index.html>. Acesso em: 05 maio 2010.

SCHWABER, ken; beedle, mike. Agile ***Software* Development With *Scrum***. Prentice hall, 2002.

SOMMERVILLE, Ian. **Engenharia de *softwarel***. São Paulo: Addison Wesley, 2003.

TELES, Vinícius. M. ***Extreme Programming*: aprenda como encarar seus usuários desenvolvendo *software* com agilidade e alta qualidade.** São Paulo: Novatec Editora Ltda, 2004.

ZANATTA, Alexandre Lazaretti. **X*Scrum*: uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI.** 2004. 144 f. Dissertação (Mestrado) - Universidade Federal de Santa Catarina, Florianópolis.

APÊNDICE A - ARTIGO

Utilização do Framework Epf *Composer* para a Criação de um Modelo de Processo de Desenvolvimento de *Software* Baseado na Integração de Metodologias Ágeis com Tradicionais

Ricardo Brina Mondardo[1], Gustavo Bisognin[2]

[1]Acadêmico(a) do curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias - Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC - Brasil
[2]Professor(a) do curso de Ciência da Computação - Unidade Acadêmica de Ciências, Engenharias e Tecnologias - Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brasil

ricardo9300@gmail.com, gbisog@gmail.com

Resumo. Durante o desenvolvimento de projetos de software, a compreensão incorreta dos requisitos do sistema, a falta de experiência de equipes de desenvolvimento, os prazos e custos excedidos, fazem com que seja comprometida a qualidade final do software que esta sendo criado. Diante dessas afirmações a utilização de um modelo de desenvolvimento de software que utilize abordagens ágeis e tradicionais se torna algo que pode minimizar significativamente esses problemas, trazendo mais qualidade no processo de desenvolvimento e no software implementado.

Abstract. During the development of software projects, the incorrect understanding of system requirements, lack of experience of development teams, deadlines and cost overruns, make it compromised the quality of the software that is being created. Faced with these statements using a model of software development using agile and traditional approaches becomes something that can significantly reduce these problems. Bringing more quality into the development process and software implemented.

Palavras-Chave: engenharia de *software*; metodologias de desenvolvimento tradicionais; metodologias de desenvolvimento ágeis.

1. Introdução

Na construção de um componente de *software*, nem sempre é possível compreender e desenvolver corretamente os requisitos do sistema, havendo impacto em prazo, custo e esforço. Outro fator crítico de sucesso para o desenvolvimento de sistemas computacionais é a pouca experiência das equipes de desenvolvimento e a inexistência de recursos para o planejamento de projetos.

Para resolução dos problemas citados acima, a engenharia de software sugere o uso de metodologia de desenvolvimento, podendo ser tradicionais, ou ágeis.

Conforme aborda Pressman (2006), a escolha de um modelo para o desenvolvimento *software* não é uma decisão fácil, as alternativas disponíveis para os engenheiros de *software* são os chamados métodos tradicionais como RUP, que tem na ordem e consistência do projeto, seus tópicos dominantes.

Outra possibilidade de escola de um modelo para o desenvolvimento são os chamados métodos ágeis, que têm sido apontados como uma alternativa às metodologias tradicionais de desenvolvimento. Uma vez que esses possuem o foco na agilidade do desenvolvimento e na qualificação automática dos produtos de *software* enquanto as metodologias tradicionais possuem foco no processo e na forte documentação do sistema, consumindo um esforço consideravelmente grande para o seu desenvolvimento, que muitas vezes, é construído sem qualidade.

Baseando-se nos fatores acima apresentados e no grande anseio dos engenheiros de *software* pela utilização de métodos que remetam a organização e colaboração em projetos de *software* de forma ágil e consistente, contextualiza-se a proposta para a criação de um processo modelo abordando as melhores práticas das metodologias ágeis e tradicionais.

3. Engenharia de software

Em meados de 1970, o mercado de *software* passava por uma crise, e como forma de solucioná-la surge a engenharia de *software* que propõe um tratamento mais sistemático e controlado para construção de *softwares* complexos, que envolvem um conjunto de componentes de *software*, como estruturas de dados, algoritmos, procedimentos, funções entre outros (PRESSMAN 2006).

Inicialmente os primeiros modelos usados para o desenvolvimento de projetos na área de *software*, eram baseados em técnicas utilizadas com sucesso na engenharia industrial em série, porém com a evolução da engenharia de software, foram surgindo novos modelos de desenvolvimento de *softwares* com novos procedimentos que são mais adaptáveis a situações reais vividas nas empresas

Engenharia de *software* é uma área do conhecimento que tem por objetivo a padronização e organização do desenvolvimento de *software* em projetos, buscando um aumento da produtividade e da qualidade dos produtos e processos envolvidos na concepção de sistemas computacionais.

Pressman (2006) afirma que a engenharia de *software* oferece ao profissional envolvido no processo uma base de conhecimento para construção de um produto de qualidade. Sendo que para isso ocorrer, essa disciplina dispõe de quatro elementos fundamentais que são: métodos, ferramentas, procedimentos e pessoas.

4. Metodologias de Desenvolvimento

A construção de um *software* pode ser algo muito complexo, pois envolve várias tarefas e procedimentos. Em consequência disso, existem metodologias de desenvolvimento que são responsáveis por fazer com que a equipe envolvida no projeto, possa interagir com todos os procedimentos e tarefas necessários na construção do *software* de forma objetiva e sistemática.

4.1 Método Scrum

Método de desenvolvimento ágil *Scrum* criado por Jeff Sutherland e Ken Schwaber no início da década de 1990, tem seu nome originário de um jogo de *Rugby*[4]. Segundo Schwaber (2002) o método trabalha o desenvolvimento de *software* de forma

4 *Rugby*: Esporte originário da Inglaterra, onde a disputa de bola se chama *Scrum*

incremental e iterativa, focando nas pessoas e ambientes em situações onde os requisitos surgem ou mudam constantemente.

Conforme Schwaber (2002) o *Scrum* tem como características, requisitos instáveis ou até desconhecidos, desenvolvimento dividido em iterações que são chamadas de *Sprint*s e duram de duas a quatro semanas.

Um característica importando do método, é sua equipe de desenvolvimento, que tem como pré-requisito a auto-organização e a comunicação entre os membros que ocorre de forma rápida e objetiva possível, sendo realizada na maioria das vezes face a face.

O desenvolvimento no método *Scrum* se baseia nos papéis, *Product Owner*, *Scrum Master* e *Scrum* Team, seu processo é dividido pelas fases PreGame, Game e PostGame, onde são aplicadas as práticas gerenciais Sprint, Sprint Planning Meeting, Daily *Scrum*, Sprint Review Meeting, que são responsáveis pelos artefatos Product Backlog, *Sprint backlog*, *Burndown Chart e* Planning Poker, que é usado como métrica para mensuração das estimativas.

A seguir pode ser visto por meio da Figura 1 o funcionamento do *Scrum*.

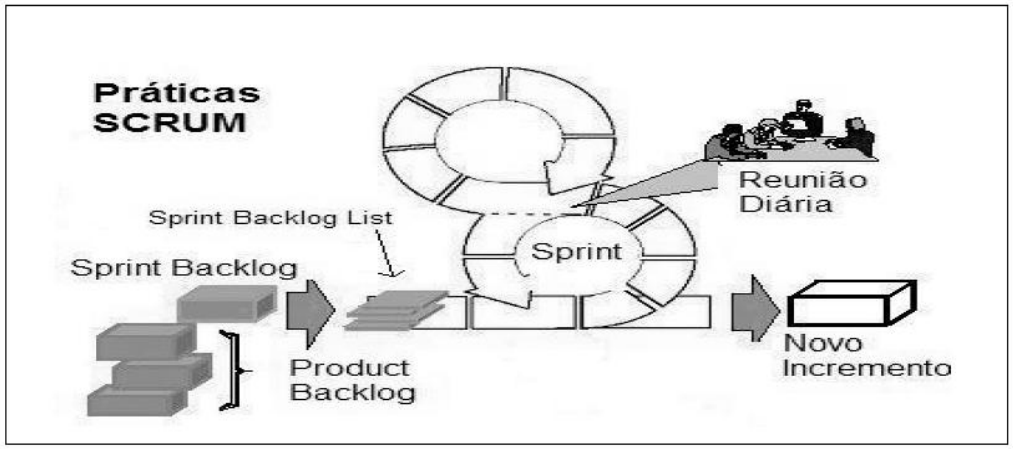


Figura 1. Funcionamento do *Scrum*
Fonte: ZANATTA (2004, p.48).

4.2 Método XP

Criado no final da década de 90 por Kent Beck, o método *Extreme Programming* também conhecido por XP, é um método de desenvolvimento ágil que segundo Teles (2004, p. 21), é voltado para:

a. projetos cujos requisitos são vagos e podem mudar com freqüência;

b. desenvolvimento de sistemas orientados à objeto;

c. equipes pequenas preferencialmente até doze desenvolvedores;

d. desenvolvimento incremental (ou iterativo) onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do processo.

Teles (2004) aponta que o método busca assegurar que o cliente tenha o máximo valor durante todo o processo de desenvolvimento do *software*, organizando

um conjunto de práticas que atuam de forma que o cliente receba sempre um auto retorno do seu investimento.

Conforme a afirmação de Beck (2000), o XP é um método baseado em regras ao invés de uma seqüência de processos.

Segundo Teles (2004) e Zanatta (2004) a metodologia de desenvolvimento ágil XP é focada em quatro valores para obter um *software* com qualidade que são: simplicidade, comunicação, coragem e feedback.

Conforme Zanatta (2004) o desenvolvimento de um *software* utilizando o método XP, é dividido em quatro práticas que são: planejamento, *design*, testes e codificação, sendo que cada uma dessas práticas contém uma série de atividades que devem ser seguidas.

4.3 RUP

No fim da década de 1980 a empresa *Objectory* AB que pertencia a Ivar Jacobson, definiu um processo para desenvolvimento de *software* baseado em documentos UML chamado *Objectory* v1.0. Passados alguns anos, em 1995 a empresa foi vendida para *Rational* Corporation, que integrou seus produtos ao Objectory v1.0 criando o Rational Objectory Process 4.0 para posteriormente em 1998 lançando o Rational Unified Process 5.0, que foi adquirido pela IMB em 2003 através da compra da Rational Corpotarion (BASSI 2008).

A metodologia de desenvolvimento RUP, foi criado originalmente para se ajustar à projetos de diversos tamanhos, domínios de aplicação e tecnologias utilizadas para codificação (KRUCHTEN, 2001).

Segundo Cintra (2006) o RUP descreve:

a. **papéis ou perfis de trabalho (quem):** define quem é responsável por cada uma das tarefas;

b. **artefatos (o que):** representa o que será implementado durante o processo de desenvolvimento;

c. **atividades (como):** define como os representantes de cada papel trabalham para atingir os objetivos do projeto e construir o *software*;

d. **fluxo de atividades (quando):** orientam a execução das atividades, definindo quando elas devem iniciar.

Conforme afirmação de Cintra (2006) o RUP define quatro fases, nove disciplinas, mais de 40 papéis e 100 artefatos. Por ser um produto comercial possui mais de 1.000 páginas de orientação e conta com processos adicionais à versão padrão. Devido a essa grande documentação é considerado por muitos autores, um método pesado, porém o RUP disponibiliza mecanismos de customização, onde é possível definir um sub-conjunto de artefatos, papéis e atividades que realmente são necessários.

A seguir são definidas as quatro fases do RUP conforme Cintra (2006):

a. **iniciação ou Concepção:** nessa fase é apresentado uma visão geral do produto, é feito uma abrangente análise dos requisitos do sistema e determinado o escopo do projeto;

b. **elaboração:** é responsável pelo planejamento das atividades e recursos necessários, a especificação detalhada dos requisitos e a arquitetura do sistema;

c. **construção:** o foco dessa fase é a implementação do *software*, sendo responsável pela evolução do mesmo e de sua arquitetura até que ele esteja finalizado para entrega ao cliente;

d. **transição:** nessa fase o sistema é repassado aos usuários, são resolvidas as questões de empacotamento, entrega, treinamento, suporte e manutenção.

Pode-se ver na Figura 2 a estrutura do RUP, demonstrando no eixo horizontal os seus aspectos dinâmicos com suas quatro fases, e no eixo vertical, seus aspectos estáticos sendo as nove disciplinas.

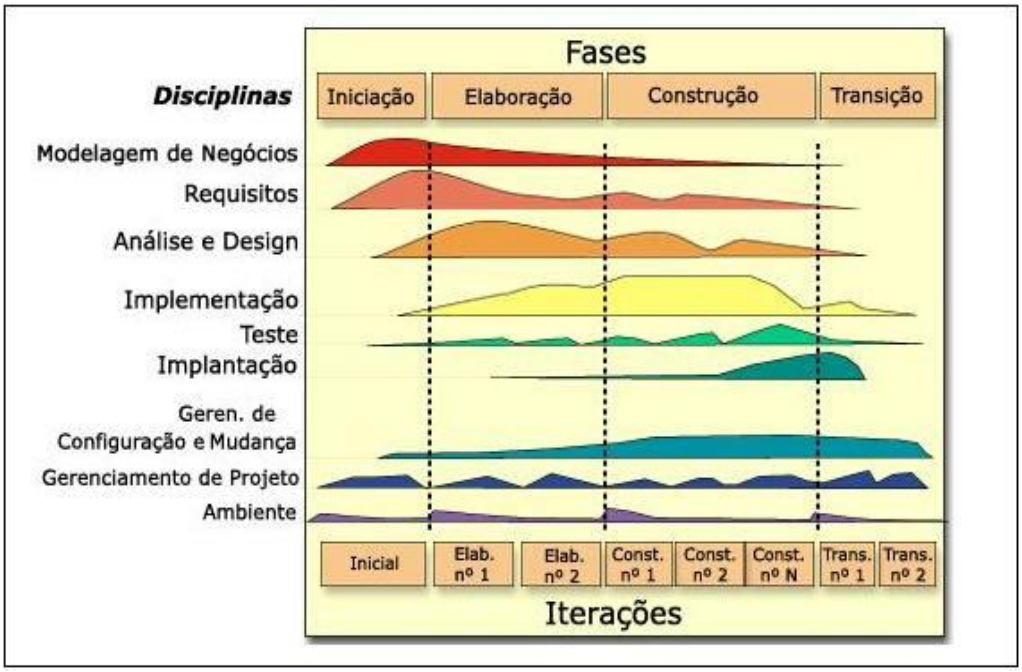


Figura 2. Fases do RUP
Fonte: PEREIRA, E (2005, p.36).

5. EPF Composer

EPF *Composer* é uma ferramenta de código aberto criada pela IBM, desenvolvida em Java[5], utilizado para criar, adaptar e publicar métodos para desenvolvimento de *software* (HAUMER, 2010).

A ferramenta procura levar ao profissional uma sólida base de conhecimento, disponibilizando processos pré-definidos, que podem ser utilizados da maneira que forém determinados ou adaptados de acordo com as características de cada projeto.

Segundo Haumer (2010) a separação do conteúdo é feita por meio de

5 Java: Linguagem de programação orientada a objeto

nomenclaturas, atores, tarefas e produtos de trabalho sendo organizados através de sequências semi-ordenadas.

Uma grande vantagem da utilização desta ferramenta é a possibilidade de disponibilizar de forma rápida e fácil todo o conteúdo criado em html.

6. Modelo Desenvolvido

Todo modelo desenvolvido por meio deste trabalho, foi documentado através de um website, com a intenção de facilitar a busca e a visualização do conteúdo descrito. sendo estruturado de maneira com que possibilite ao usuário diferentes modos de adquirir a informação desejada, podendo ser por meio de fluxo de etapas e atividades ou links, onde o usuário pode navegar em busca da informação.

Abaixo é possível visualizar através da figura 3 o fluxo de etapas do modelo.

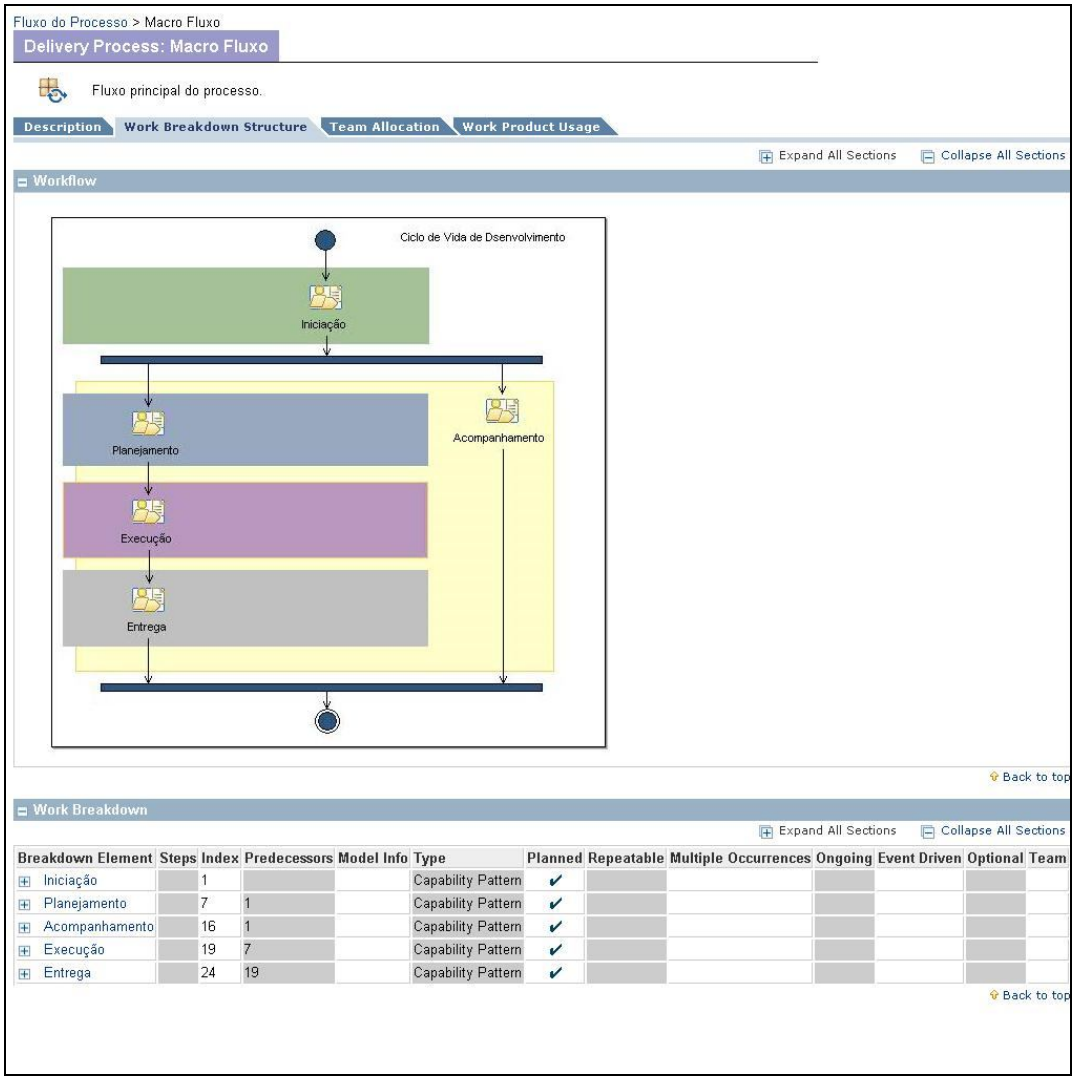


Figura 3. Fluxo de Etapas.

Cada etapa do modelo remete ao usuário a um fluxo de tarefas, que é demonstrado no mesmo formato que o fluxo de etapas, indicando um ordem pré-definida para realização das tarefas.

Cada tarefa apresentada uma definição de seus objetivos, passos para conclusão, descrição principal e relacionamento com outros itens do modelo, como papéis, artefatos e demais tarefas.

Os papéis a serem desempenhados no modelo serão os mesmos utilizados na metodologia de desenvolvimento ágil *Scrum*, que são: *Scrum Master*, *Product Owner* e Equipe, esses papéis foram escolhidos, devido a se adequarem melhor a maioria das empresas de pequeno e médio porte.

O modelo também define as etapas do desenvolvimento, sendo elas: iniciação, planejamento, execução, acompanhamento e entrega.

Na etapa de iniciação o modelo tem como objetivo a descrição de tarefas responsáveis por: todo o desenvolvimento inicial dos requisitos do sistema, formulação do escopo das necessidades e priorização dos requisitos quanto ao seu valor de negócio.

A escrita das funcionalidades no processo descrito no modelo será feita através de estórias de usuário, que é uma técnica utilizada na metodologia *Scrum*.

Na etapa de planejamento do projeto, o modelo descreve como responsabilidade, a definição das estimativas de tamanho das estórias de usuário, onde será utilizada a métrica *planing poker* para estimativas de tamanho, sendo que nesta técnica é utilizado um baralho especial para que seja possível a definição das estimativas.

Ainda nesta etapa o modelo descreve tarefas para criação de vários artefatos provenientes de modelos tradicionais e ágeis como cronograma, matriz de responsabilidades, *product backlog* e *sprint backlog*.

O processo descrito ainda conta com a etapa de execução onde foi definas tarefas com objetivo de codificar e validar as estórias de usuário do *product backlog*, de forma ágil, descrevendo todo processo de codificação que será realizado em ciclos de tempo denominado *sprints*, seguindo a metodologia ágil *Scrum*.

Para completar o ciclo do desenvolvimento, o modelo conta com as etapas de acompanhamento e entrega que são responsáveis por definir tarefas como, a reunião de acompanhamento diário, homologação e liberação de *builds* para o cliente.

Como forma de validação, o próprio modelo foi utilizado para o desenvolvimento deste trabalho, possibilitando a aplicação das tarefas descritas e a criação dos artefatos do projeto entre outras orientações do processo, em um projeto de trabalho real.

Apenas algumas tarefas do modelo não puderam ser desempenhadas na validação, pois como o modelo foi aplicado no desenvolvimento de um trabalho de conclusão de curso e foi executado por apenas uma pessoa, assim nem todas as tarefas a serem executadas se referiam a tarefas de um projeto de *software*, como por exemplo: a codificação de uma funcionalidade, porém o modelo pode-se adequar de forma satisfatória para o desenvolvimento.

Ao utilizar o modelo para a criação do trabalho, fez-se necessário definir uma ferramenta de produtividade para o gerenciamento do desenvolvimento do projeto, onde foi utilizada, como sugerida no modelo, a ferramenta Pronto. Esta ferramenta obteve um resultado bastante satisfatório no desenvolvimento projeto, e pode ser adquirida gratuitamente através do site http://pronto.bluesoft.com.br.

Para cada tarefa do trabalho foi criada uma estória de usuário na ferramenta Pronto, cada estória foi agrupada em um *product backlog*, que ao longo do projeto foi recebendo mais estórias, como o desenvolvimento aconteceu através de *sprints* também foram criados *sprints backlog,* contendo todas as estórias a serem desenvolvidas em cada *sprint*.

Todo gerenciamento das estórias foi realizado por meio de um *kambam* digital, onde essas foram separadas em estórias não iniciadas, iniciadas, prontas para testes, em testes e prontas.

Após todas as estórias de usuário criadas concluídas e validadas, o modelo foi exportado no formato HTML, possibilitando a sua publicação no formato de um website.

7. Conclusão

A metodologia de desenvolvimento ágil *Scrum*, pode ser considerada uma ótima alternativa para projeto de *software* de pequeno e médio porte, pois atua melhor com constantes mudanças nos requisitos através do *feedback* do cliente que está altamente envolvido no projeto. Também mostrou-se adaptável, pois suas atividades puderam ser desempenhadas em conjunto com algumas tarefas de metodologias, também ágeis como XP e tradicionais como RUP.

Pode-se considerar como base para criação do modelo, a metodologia de desenvolvimento tradicional RUP, pois o mesmo foi elaborado com a definição de papéis, etapas, tarefas e artefatos que se relacionavam entre si. Também apresentou-se eficaz pela utilização de artefatos na questão de controle de responsabilidades, facilitando assim a identificação de impedimentos e a falta de comprometimento dos membros da equipe.

A utilização de etapas, papéis, tarefas e artefatos como descreve a metodologia RUP para criação do modelo se apresentou como uma boa forma de controle de responsabilidades e fluxo de tarefas pois indica como, quem, quando e o que será feito cada atividade no projeto

Após o estudo dos dois tipos de metodologia, foi possível verificar que a criação de um modelo baseado na integração do ágil com o tradicional, poderia trazer a agilidade e o controle necessário para projetos de pequeno e médio porte.

No decorrer do trabalho algumas dificuldades foram encontradas, onde é possível citar, a compreensão da metodologia RUP, que é processo grande, envolvendo muitos artefatos, tarefas e responsabilidades e o entendimento da ferramenta EPF *Composer*, que por ser um algo novo no mercado ainda não possua muita documentação a seu respeito.

Com a validação do modelo proposto sendo feito através da sua própria utilização no desenvolvimento do trabalho, pode-se verificar que o mesmo atingiu resultados satisfatórios quanto ao seu desempenho e controle das atividades.

Como não foi possível aplicá-lo com uma equipe de desenvolvimento em um projeto de *software*, algumas tarefas não puderam ser validadas como deveriam. Como por exemplo, a tarefa de definir as estimativas de tamanha das estórias, pois essa tarefa se baseava no conjunto da equipe para definir as estimativas.

De acordo com os estudos realizados, a utilização das boas práticas reunidas no processo fruto deste trabalho, pode auxiliar significativamente no ganho de qualidade do desenvolvimento dos sistemas nas organizações, proporcionando o crescimento da qualidade dos produtos desenvolvidos e potencializando o

crescimento financeiro nestas organizações.

Como sugestão de trabalhos futuros sugere-se:

- Validar o modelo proposto em um estudo de caso com a aplicação do processo em um projeto de desenvolvimento de *software*.
- Definir uma metodologia de gerenciamento apoiado por uma ferramenta automática que apoie a execução do modelo proposto;
- Ajustar o processo definido, de acordo com o modelo de referência para a Melhoria de Processo de *Software* Brasileiro MPS.BR

Referências

BASSI, Dairton Luiz. **Experiências com desenvolvimento ágil.** 2008. 154 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo.

BECK, Kent. **Extreme Programming:** Embrace Change. Addison Wesley. 2000.

CINTRA, Caroline Carbonell. **A Implantação de um Processo de Engenharia de Requisitos Baseado no Processo Unificado da Rational (RUP) Alcançando Nível 3 de Maturidade da Integração de Modelos de Capacidade e Maturidade (CMMI) Incluindo a Utilização de Práticas de Métodos Ágeis.** 2006. 160 f. Dissertação (Mestrado) - Universidade do Rio Grande do Sul, Porto Alegre.

HAUMER, P., **Eclipse Process Framework *Composer* Part 1: Key Concepts**. Eclipse, [S. l.], 15 dezembro 2005. Disponível em <http://www.eclipse.org/epf/general/ getting_started.php>. Acesso em: 10 maio 2010.

PEREIRA, Eliana Beatriz. **Uma proposta para adaptação de processos de desenvolvimento de *software* baseados no *rational unified process.*** 2005. 125f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio Grande do Sul

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

SCHWABER, ken; beedle, mike. Agile ***Software* Development With *Scrum***. Prentice hall, 2002.

TELES, Vinícius. M. ***Extreme Programming*: aprenda como encarar seus usuários desenvolvendo *software* com agilidade e alta qualidade.** São Paulo: Novatec Editora Ltda, 2004.

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.