FÁBIO MACHADO DINIZ

IMPLEMENTAÇÃO DE UMA METODOLOGIA ÁGIL NA ÁREA DE DESENVOLVIMENTO DE SOFTWARE DE UMA CONSULTORIA FINANCEIRA

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenheiro de Produção

São Paulo

2010

FÁBIO MACHADO DINIZ

IMPLEMENTAÇÃO DE UMA METODOLOGIA ÁGIL NA ÁREA DE DESENVOLVIMENTO DE SOFTWARE DE UMA CONSULTORIA FINANCEIRA

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenheiro de Produção

Orientador:
Prof. Dr. Renato de Oliveira Moraes

São Paulo

2010

FICHA CATALOGRÁFICA

Diniz, Fábio Machado
Implementação de uma metodologia ágil na área de desen - volvimento de software de uma consultoria financeira / F.M. Diniz. -- São Paulo, 2010.
89 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1. Administração de projetos 2. Desenvolvimento de software 3. Engenharia de produção I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II. t.

À Carla Guillen

AGRADECIMENTOS

Aos meus pais, pelo apoio e confiança oferecidos em todas as etapas de minha formação.

Ao professor Renato, pelas inúmeras dicas, reuniões e referências que tornaram este trabalho possível.

À Escola Politécnica da USP, em especial ao departamento de Engenharia de Produção, que me mostrou o que é a Engenharia e o quanto isso é necessário e ao mesmo tempo escasso no atual mercado de trabalho.

Aos meus ótimos amigos da Engenharia de Produção, por contribuírem com minha formação e com o aprendizado do trabalho em equipe.

Ao TPN, em especial ao Humberto e ao Dennis, que me iniciaram profissionalmente no desenvolvimento de software em um ambiente com técnicas e procedimentos invejáveis mesmo para uma grande empresa de software do mercado trabalho.

À Poli Júnior, por ter me ensinado na prática o que é a gestão de projetos.

Ao Allan, por ter me propiciado a idéia inicial deste trabalho.

À Carla, por estar presente em todos os momentos oferecendo compreensão, apoio e amor.

More than a discipline or a body of knowledge, engineering is a verb, an action word, a way of approaching a problem.

(Scott Whitmire)

RESUMO

Este trabalho apresenta uma análise da área de desenvolvimento de software de uma consultoria do setor financeiro que tem como finalidade melhorar os processos que são executados atualmente na empresa. São apresentados os principais problemas encontrados, seguidos por um breve resumo de técnicas e metodologias de gestão diversas que poderiam ser aplicadas na empresa.

Uma nova análise do contexto da empresa, agora tendo como base toda a revisão teórica, é feita culminando numa proposta de metodologia de gestão dos projetos de desenvolvimento de sistemas específica para o caso em questão, que conta com particularidades como o número reduzido de desenvolvedores e projetos com caráter experimental.

Palavras-chave: Engenharia de Produção, Gestão de Projetos.

ABSTRACT

This paper presents an analysis of the software development area of an financial sector's consultancy which aims to improve the processes currently running in the enterprise. The main issues found are presented, followed by a short brief of techniques and management methodologies that could be applied in the enterprise.

A new analysis of the enterprise's context, now based in the theoretical review, is done, concluding in the proposal of a management methodology of the software development projects specifics to the case that has been analyzed, which has particularities like the low number of developers and the experimental character of the projects.

Keywords: Production Engineering, Project Management.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 **Descrição da Empresa**

O presente trabalho foi desenvolvido numa empresa que atua como consultoria financeira focada em projetos constituídos majoritariamente de atividades de modelagem matemática. Entre outros, os clientes predominantes da empresa são bancos de grande e médio porte, além de grandes empresas passando por situações como fusões e aquisições e órgãos do governo.

A empresa se especializou na modelagem computacional de soluções para o mercado financeiro, contemplando e conciliando necessidades de diversas áreas de empresas do setor financeiro. Seus sistemas possuem como característica a presença de soluções matemáticas desenvolvidas especificamente para o projeto em questão além de um grande foco no tempo de resposta das ações do software.

Além de consultorias e do desenvolvimento de sistemas a empresa possui uma área de treinamentos in-company que são fornecidos para grandes empresas do setor financeiro brasileiro e compreendem temas como finanças pessoais, gestão de risco, tópicos de marcação de mercado, engenharia financeira e finanças quantitativas.

O atual quadro de colaboradores compreende sete pessoas, conforme ilustrado no organograma presente na Figura 1.1, sendo que dois são os atuais sócios da empresa responsáveis pela área comercial, pela seleção de projetos e por decisões de cunho matemático e financeiro.

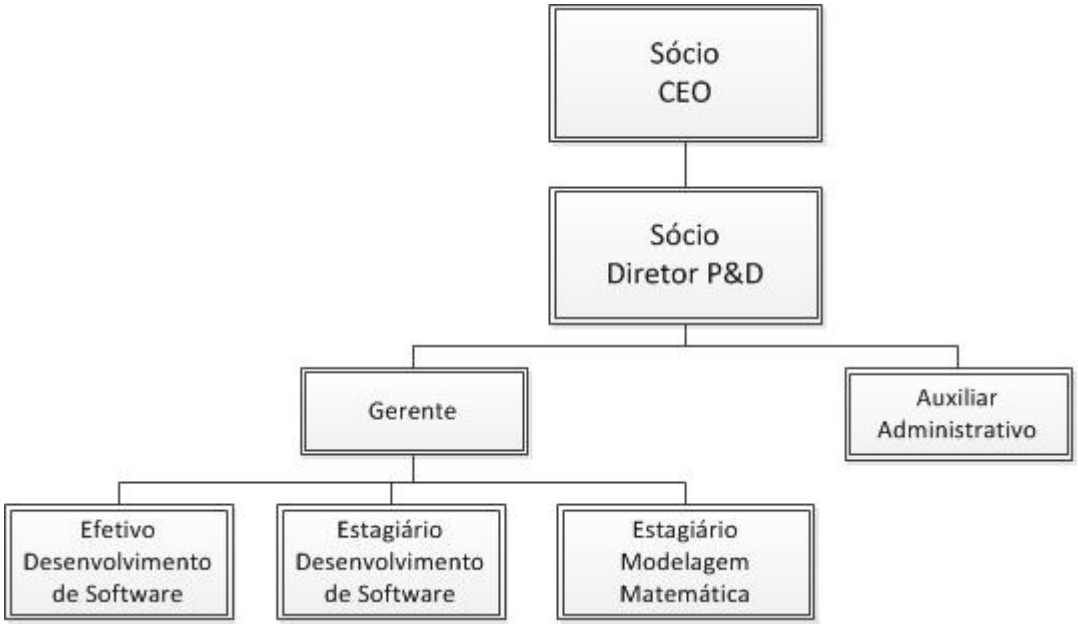

Figura 1.1 – Organograma da Empresa

Além dos sócios há uma pessoa responsável por todas as questões administrativas da empresa, cuidando desde assuntos relacionados aos recursos humanos até a gestão de contas a pagar e receber.

Por fim há a equipe de desenvolvimento de projetos que contém duas pessoas na área de modelagem matemática e financeira, sendo um membro efetivo que também atua como gerente e um estagiário, e duas pessoas na área de desenvolvimento de software, que também se dividem entre um membro efetivo e um estagiário.

Devido ao pequeno tamanho da empresa e ao grande porte dos projetos e clientes, cada colaborador acaba trabalhando em múltiplos projetos, além de auxiliar em tarefas como suporte de produtos da empresa, construção de material dos treinamentos entre outras.

Infelizmente, dada as atuais políticas de sigilo da empresa, esta não poderá ser identificada, porém acredita-se que este sigilo não comprometerá as análises feitas no trabalho.

1.2 Caracterização dos Projetos de Software

Os sistemas desenvolvidos na empresa se dividem em duas categorias: Os sistemas que são implementados como parte de uma consultoria e aqueles que são vendidos como projetos completos.

A primeira categoria trata de softwares altamente especializados em determinada tarefa, sendo que normalmente lidam apenas com automatização de cálculos matemáticos e financeiros. Os tipos de plataforma variam muito de cliente para cliente, sendo que a empresa não adota um padrão fixo para o desenvolvimento.

A maioria destes sistemas são interfaces geradas em VB.NET ou com a união de Excel e VBA e que utilizam bibliotecas implementadas em plataformas voltadas para a otimização do tempo de processamento.

Na categoria dos sistemas que são tratados como projetos completos os softwares normalmente possuem funcionalidades mais abrangentes e devem realizar diversas tarefas distintas. Neste caso, normalmente não são vendidas interfaces para o usuário, sendo que os produtos são bibliotecas de cálculo que serão integradas nos sistemas dos clientes e que foram implementadas em C e podem possuir integração com bancos de dados em MySQL.

Devido à quantidade de esforço necessária para a execução das tarefas de modelagem matemática dos projetos, que normalmente requer a atenção de ambos os diretores da empresa bem como do membro efetivo da área de modelagem a empresa tem uma cultura de priorizar os esforços na área matemática, não dando a devida importância para as práticas de desenvolvimento de software.

Este hábito gera sistemas que são implementados de maneira ad-hoc, sem o devido cuidado de se especificar um escopo do software, que acaba sendo uma parte que se altera constantemente dentro do escopo da consultoria sendo prestada (para sistemas que se encaixam na primeira categoria). Isto significa que a consultoria possui um escopo bem definido com determinadas tarefas e modelos que devem ser implementados, porém não há um acordo inicial de quais partes da consultoria serão transformadas em um sistema para o cliente e nem quais funcionalidades este sistema deve possuir. Com esta falta de definição inicial os sistemas possuem um escopo que varia conforme o desenvolvimento da consultoria, gerando retrabalho e atrasos nas entregas.

Outra conseqüência é a falta de um cronograma para o desenvolvimento destes sistemas, sendo que normalmente é definida a data de entrega do produto e então os programadores vão desenvolvendo as funcionalidades sem seguir nenhum planejamento específico.

A falta de planejamento acaba gerando outros problemas, pois os programadores não sabem, de maneira global, tudo que o software deve incluir, sendo que as especificações são passadas de maneira oral, sem que seja feito nenhum registro. Isto acaba gerando sistemas que são entregues sem todas as funcionalidades que foram pedidas pelo cliente ou com funcionalidades que não estão presentes no escopo.

Esta informalidade na gestão do escopo acaba deixando a empresa desprotegida de possíveis mudanças no escopo por parte do cliente, o que acaba ocorrendo freqüentemente e acaba gerando retrabalho e desmotivação por parte da equipe de desenvolvedores.

No âmbito dos programadores, também não existem boas práticas para o desenvolvimento, sendo que desde a fundação da empresa os colaboradores que faziam este tipo de tarefa eram pessoas da área matemática ou financeira e que não possuíam conhecimentos básicos de engenharia de software, como a gestão do ciclo de vida do projeto e a prática de documentação das tarefas realizadas.

1.3 **Resumo do Estágio**

A experiência de estágio na empresa teve início em outubro de 2009, sendo que a área de atuação é principalmente a de participar no desenvolvimento de softwares na empresa, mas também de avaliar os atuais processos de gestão de projetos e de software.

Devido ao crescimento da empresa nos últimos anos tanto a área de modelagem quanto a de desenvolvimento de software começaram a precisar de pessoas especializadas nestas funções, que adquirissem experiência necessária para manter o ritmo e a qualidade dos projetos que a empresa oferece.

A área de desenvolvimento sofre com a falta de práticas básicas de gestão. O primeiro problema que se nota com os projetos atuais e com os passados é a falta do gerenciamento de escopo, que quando é aliada com a falta de planejamento adequado, utilizando-se de cronogramas e lista de tarefas, acaba tendo como conseqüência softwares que apresentam falhas, erros de modelagem matemática e falta de validações, além atrasos na entrega e a falta de garantia de que o produto irá atender as necessidades do cliente.

Esta área foi criada com a entrada do autor deste trabalho e após três meses recebeu mais um membro efetivo que também fora contratado especificamente para o desenvolvimento de software.

A tarefa de programação de sistemas que antes era feita por qualquer colaborador da empresa agora é desempenhada apenas pelos membros desta área, o que também ocorreu com a área de modelagem matemática e financeira.

As atividades que foram desempenhadas no estágio foram:

- Desenvolvimento de documentação técnica de sistemas da empresa;
- Planejamento e controle dos projetos de software internos e externos da empresa;
- Definição de metodologias e padronizações para o desenvolvimento de software;
- Suporte de sistemas da empresa para os atuais clientes.

1.4 **Definição do objeto de pesquisa**

1.4.1 **Contexto atual**

Existe na empresa um sentimento de insatisfação com o histórico recente do desenvolvimento de software que foi realizado.

Segundo a visão dos sócios, fortemente baseada nas experiências com o sistema sendo finalizado no início do estágio, a empresa foi adquirindo vícios ao longo do tempo nesta área que contribuíram para altos custos nos projetos devido ao gasto de tempo por parte dos desenvolvedores.

Primeiramente há a questão da falta de planejamento dos sistemas, que possuem um desenvolvimento ad-hoc, sem haver certezas de qual deve ser o produto final, pela falta da definição do escopo e das especificações do produto, e qual deve ser o ritmo de implementação. Isto acaba gerando uma série de atrasos no desenvolvimento, já que sem um planejamento adequado os desenvolvedores se perdem em meio a uma lista de funcionalidades que foram definidas de maneira informal com o cliente. Além disso, dado o pequeno número de colaboradores presentes na empresa todos acabam trabalhando em vários projetos de maneira simultânea, o que só acentua as consequências da falta de planejamento, já que não há como se estabelecer um cronograma claro de tarefas de cada colaborador devido

ao grande número de atividades que serão desempenhadas em diversos projetos diferentes sem nenhuma escala de importância relativa entre eles.

Em seguida há a questão da fraca gestão de escopo. Segundo Carvalho e Rabechini (2006) o escopo é o trabalho que deve ser realizado durante o curso do projeto, sendo que do PMI (2004) pode se extrair como processos para a gestão do escopo o planejamento do mesmo, sua definição, a criação de um WBS e por fim a verificação do escopo.

Dado o enfoque marginal que é dado para o desenvolvimento de software dentro dos projetos de consultoria não há uma preocupação em se definir os limites e o escopo do sistema que é desenvolvido.

Ao longo do projeto o escopo do sistema acaba se alterando constantemente, já que no inicio dos projetos não há uma discussão para estabelecer as funcionalidades do sistema e as necessidades do cliente.

Como consequência desde hábito há uma grande taxa de retrabalho que deve ser realizado constantemente, já que quando uma funcionalidade é implementada no sistema não há certeza que ela se enquadra no escopo do produto já que seus limites não estão claramente definidos. Em um dos projetos em andamento no início do desenvolvimento deste trabalho de formatura a taxa de retrabalho já ultrapassava a marca de 70% do sistema tendo sido refeito mais de uma vez. Isto ocorreu devido a mudanças drásticas no escopo por parte do cliente, o que aumentou os custos de desenvolvimento já que 70% das horas de desenvolvimento não foram previstas inicialmente. Isto gerou atrasos, já que estas horas não constavam no cronograma, e uma diminuição dos lucros, já que todo este tempo de desenvolvimento não fazia parte do orçamento do projeto.

Além do retrabalho é importante notar que não há como garantir a satisfação do cliente ao final do projeto sem que um escopo tenha sido definido na fase de planejamento, já que desta maneira os objetivos do projeto não estão claros e nunca há a certeza de que estes foram cumpridos, semelhante ao que é dito por Pressman (2003) de que sem o estabelecimento do escopo não há como se obter um cronograma do projeto que ilustra uma indicação de progresso que tenha algum significado.

Além desta visão vinda dos sócios, há também as manifestações de insatisfação vinda dos desenvolvedores da empresa.

Nota-se que estes colaboradores se sentem desmotivados com as constantes mudanças de escopo, já que há uma grande probabilidade de que um trabalho que tenha sido realizado tenha que ser refeito ou tenha que ser descartado devido há uma mudança no escopo do

sistema. Esta desmotivação pode gerar graves problemas para a empresa que vão desde baixa qualidade de código até alta rotatividade dos desenvolvedores.

Outro problema que aflige os desenvolvedores que tem origem no fraco planejamento dos projetos é a questão do suporte técnico. Nada ou muito pouco é definido sobre como será dado o suporte técnico para as soluções em software vendidas pela empresa, fazendo com que constantes contatos sejam realizados pelos clientes após a entrega dos produtos.

Tais contatos têm como natureza a resolução de problemas de uso dos sistemas, esclarecimento de dúvidas sobre os mecanismos matemáticos internos dos softwares, reclamação de erros e pedidos de novas funcionalidades.

Este canal de comunicação com o cliente, segundo Keil e Carmel (1995), é um dos mais importantes em projetos de software vendidos como pacotes, já que é um dos mais utilizados e mais efetivos para a resolução de problemas. Dada esta importância inerente ao canal de comunicação, a falta de foco que é dada a ele no na fase de planejamento do projeto/consultoria e a maneira com a qual é implementada (suporte por tempo indefinido após a entrega do sistema) é um problema para a empresa.

Dada a atual taxa de crescimento do número de projetos da empresa, que faz com que em média sejam executados três projetos simultaneamente, e do porte de cada um deles, em média projetos de quatro a seis meses, as práticas de gestão de projetos se tornam cada vez mais essências para garantir a satisfação dos clientes da empresa.

As recentes contratações realizadas na empresa (estagiário de engenharia e especialista em desenvolvimento de software) indicam a necessidade emergente de se criar um novo foco de conhecimento na empresa que se distancie do atual know-how matemático e financeiro. Este novo foco deve compreender técnicas para se criar uma estrutura de padronização de processos internos tanto administrativos quanto de produção de software.

Este trabalho pretende auxiliar na formação desta nova base de conhecimento da empresa enumerando os atuais pontos deficientes com relação à gestão de projetos de software e apresentar uma metodologia e um plano de implementação que além de corrigir tais problemas trará uma série de benefícios como no curto prazo o aumento da produtividade e no longo prazo a formação de gerentes de projetos dentro da empresa.

1.4.2 **Problema e Proposta de Estudo**

Este trabalho propõe a apresentação de uma metodologia para desenvolvimento e gestão de projetos de software que sirva de base para a implementação de práticas de gestão de projetos dentro da empresa que tenham como alvo os pontos negativos no atual contexto da empresa.

Serão enumerados os problemas presentes no contexto atual e que aspecto da metodologia que pretende diminuir seu impacto ou eliminá-lo.

Além de explicar os conceitos que formam a cultura presente na metodologia este trabalho irá propor questões práticas sobre que mudanças devem ocorrer nas diversas fases da gestão dos projetos de software na empresa.

Por fim serão analisados os primeiros resultados obtidos em um projeto da empresa que adotou as práticas presentes neste trabalho.

1.5 **Estrutura do Trabalho de Formatura**

Após este capítulo introdutório, serão apresentados neste trabalho de formatura outros cinco capítulos, conforme ilustrado na Figura 1.2.

O segundo capítulo tratará da revisão bibliográfica, abordando conceitos relacionados à gestão de projetos em geral, gestão de projetos de desenvolvimento de software além de uma revisão sobre metodologias ágeis, mais especificamente a metodologia SCRUM.

No terceiro capítulo será apresentada a abordagem metodológica do trabalho, enquanto que no quarto capítulo será apresentado o plano de implementação da metodologia em questão na empresa referente a este trabalho.

O quinto capítulo apresentará os primeiros resultados da implementação do que foi proposto neste trabalho na empresa em questão.

Por fim, o último capítulo apresentará as conclusões do trabalho.

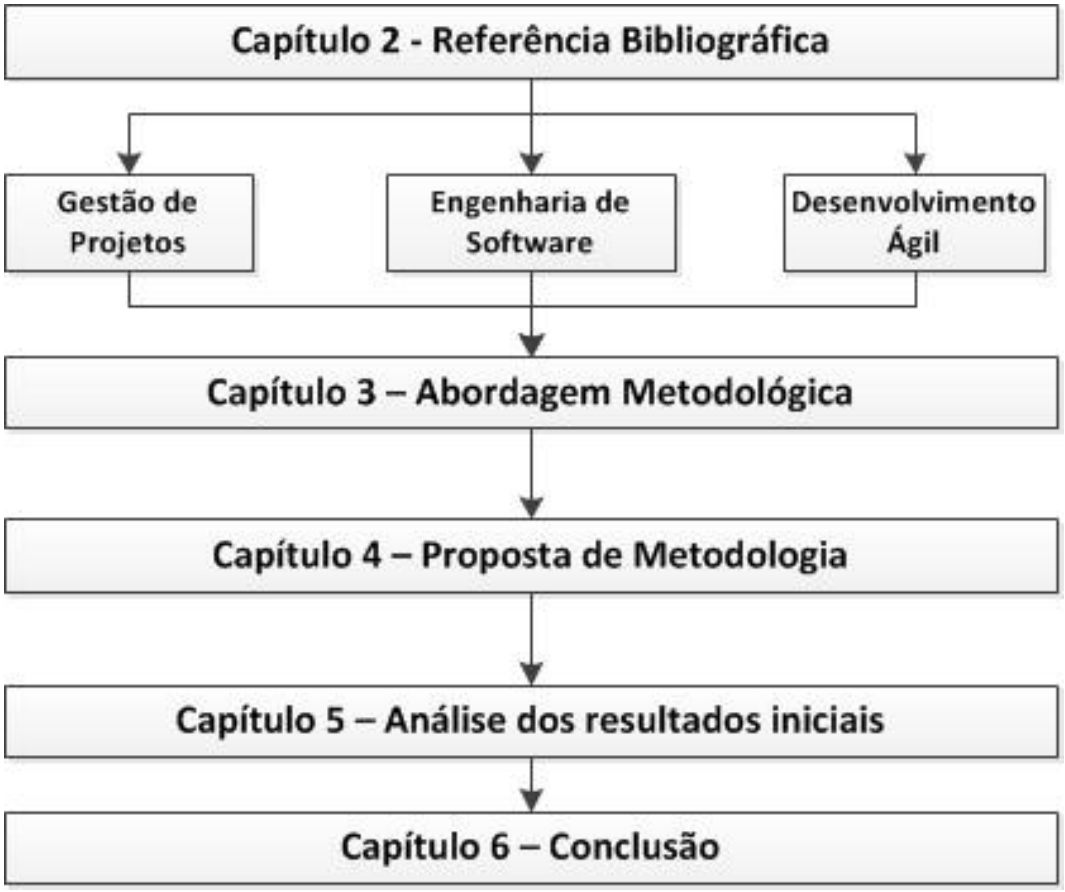

Figura 1.2 - Estrutura do Trabalho de Formatura

2 REVISÃO TEÓRICA

Para servir de base conceitual deste trabalho este capítulo apresentará uma série de conceitos chave com relação tanto à Gestão de Projetos tradicional quanto a Gestão de Projetos de Software além de conceitos sobre Metodologias de Desenvolvimento Ágil e sobre o SCRUM.

2.1 **Projeto**

Uma definição abrangente e, ao mesmo tempo, concisa de projeto pode ser encontrada na ISO 10006, que trata da qualidade na gestão de projetos: “Um processo único, que consiste em um grupo de atividades coordenadas e controladas com datas para início e término, empreendido para alcance de um objetivo conforme requisitos específicos, incluindo limitações de tempo, custo e recursos” (ISO 10006, 1997).

A questão da temporalidade e da singularidade são as mais tradicionais com relação a este conceito. A primeira trata do fato de que um projeto deve possuir uma data de início bem definida além de uma data de término quando os objetivos do projeto foram alcançados ou quando fica claro que os objetivos do projeto não serão ou não podem ser atingidos e então o projeto é terminado. A singularidade aponta para o fato de que um projeto nunca foi feito anteriormente e, sendo assim, é único (PMI, 2004).

2.2 **Gestão de Projetos**

Este conceito é definido pelo PMI (2004) como sendo a utilização de conhecimentos, habilidades, ferramentas e técnicas nas atividades de um determinado projeto de forma a alcançar ou exceder as expectativas e necessidades dos interessados do projeto.

De acordo com Hunter e Stickney (1983) “A gestão de projetos é a aplicação de uma abordagem sistêmica para a gestão de tarefas ou projetos tecnologicamente complexos cujos

objetivos tem seu estado definido explicitamente em termos de tempo, custo e parâmetros de desempenho.

Os fatores chaves para se avaliar o desempenho de um projeto são apresentados pelo triângulo da gestão de projetos, ilustrado na Figura 2.1.

Figura 2.1 - Triângulo da Gestão de Projetos (PMI, 2004)

O balanceamento das características presentes nos vértices do triângulo é que determinará a qualidade do projeto, sendo que os riscos presentes nessa representação são atrasos nas entregas do projeto, não cumprimento do escopo definido com o cliente e problemas de orçamento do projeto (PMI, 2004).

Para Kerzner (2009) pouquíssimos projetos são completados sem que ocorra algum sacrifício de um algum vértice do triângulo em detrimento de outro, porém apesar disso um projeto pode atingir o sucesso sem que o ponto de perfeito balanceamento entre os três vértices seja alcançado, fazendo com que cada projeto tenha um cubo de sucesso, conforme é ilustrado na Figura 2.2.

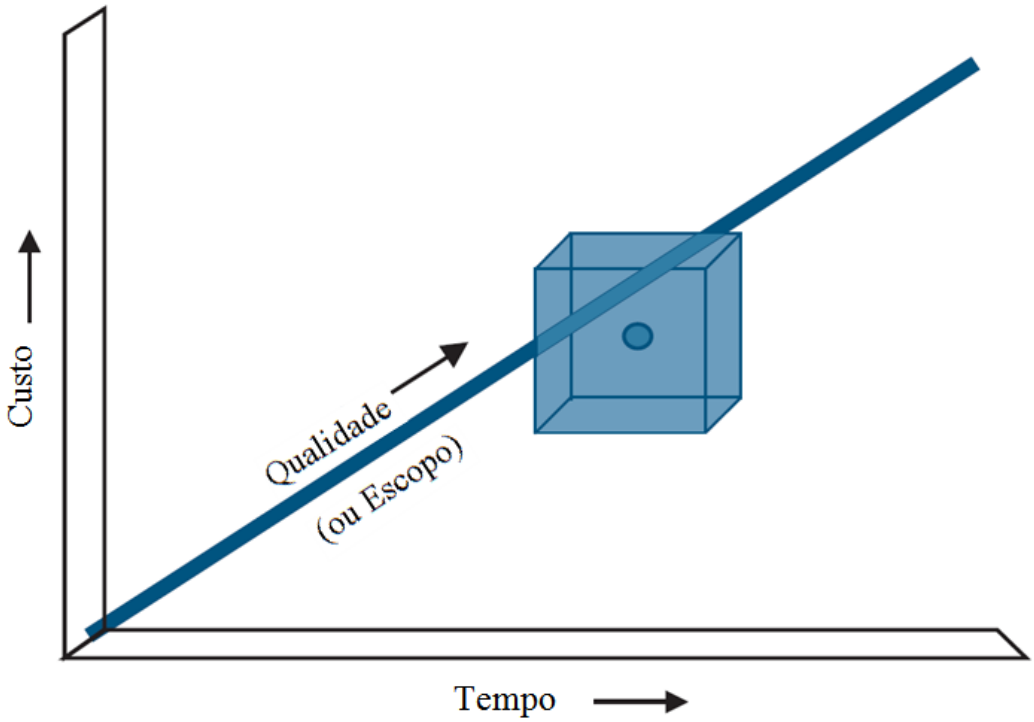

Figura 2.2 - Cubo do Sucesso (Kerzner, 2009)

Segundo Carayannis et al. (2003) a gestão de projetos é aplicada desde a era do Egito antigo, porém apenas a partir da última metade do século XX este conceito passou a ser aplicado sistematicamente para projetos complexos, com o surgimento de técnicas como CPM e PERT e o uso de gestão de projetos na industria aeroespacial. Apesar disso, pode-se notar a presença de conceitos primitivos de gestão de projetos nas cinco funções de um gerente descritas por Henry Fayol em 1916: planejar, organizar, coordenar, controlar e dirigir.

Kerzner (2009) cita motivos pela crescente adoção da gestão de projetos nas últimas décadas. Segundo o autor a GP já se consolidava em setores que trabalhavam tipicamente por projetos, porém em áreas produtivas ou de serviços havia uma grande resistência para a adoção deste conceito. O primeiro fator citado para contribuir para o aumento da adoção da GP foi o fato de que os líderes das empresas perceberam que podiam realizar uma gestão por projetos, podendo ter os benefícios tanto da organização tradicional quanto da organização vinda da gestão de projetos. Isto é comprovado pelo fato de que o grande crescimento da adoção de GP nas empresas na primeira década do século XXI tenha vindo de empresas que não trabalham especificamente no desenvolvimento de projetos.

O segundo fator decisivo foi o fato de que a economia sofrera duas recessões entre as décadas de 70 e 90, auxiliando as empresas a ter a visão de que a gestão de projetos auxilia em áreas importantes como na gestão da qualidade e do tempo.

Outra questão crítica levantada pelo autor para o aumento da adoção da GP foi o crescimento do valor e da complexidade dos projetos a partir da década de 70 conforme exibido na Figura 2.3 com o exemplo do crescimento do valor de projetos em uma empresa de construção.

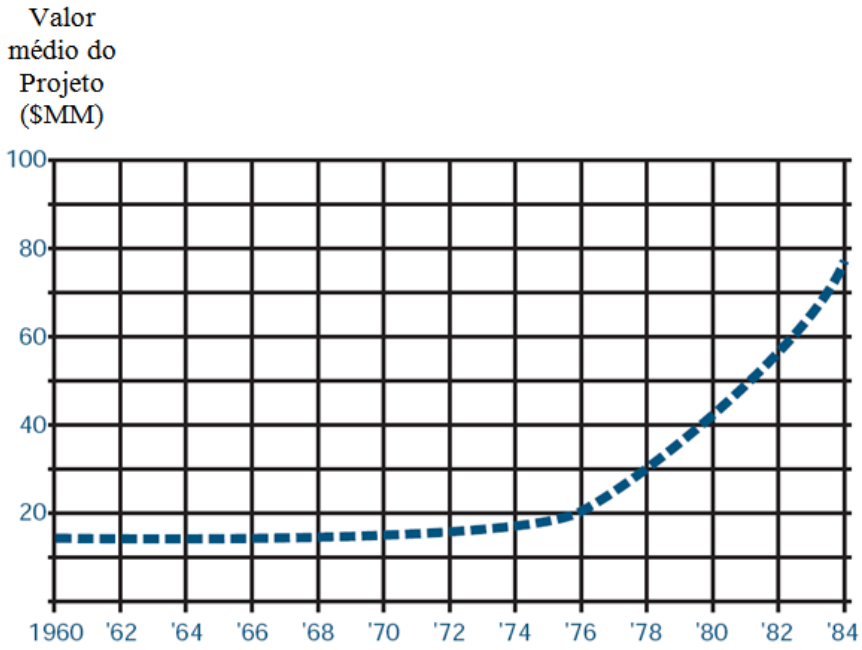

Figura 2.3 - Crescimento do Valor de Projetos (Kerzner, 2009)

A organização presente na época nas empresas não era capaz de gerir projetos deste porte sem que os riscos de prejuízos financeiros fossem altos, o que tornou a GP uma técnica essencial para manter o controle de tarefas complexas nas empresas.

2.3 Ciclo de Vida de Projetos

De maneira a aperfeiçoar o controle de recursos por parte dos gerentes surge o conceito de ciclo de vida de projetos. Este conceito apresenta uma série de fases pelas quais os projetos devem passar até que sua data de término seja alcançada.

Segundo Kerzner (2009) existem cinco fases teóricos do ciclo de vida de projetos. A primeira é a fase conceitual, onde a idéia principal do projeto é avaliada e uma análise preliminar do risco e de seu impacto no tempo, custo e nos requisitos de desempenho é feita.

A segunda fase é a de planejamento, onde as questões abordadas na primeira fase são refinadas até que aja uma clara identificação dos recursos que são requeridos no projeto. Há também o estabelecimento de parâmetros realísticos de tempo, custo e de desempenho além de uma preparação inicial para a documentação de auxílio no projeto.

Na terceira fase, denominada fase de testes, é feito um esforço final com relação à padronização e à simulação do projeto de maneira que a operação possa ter início. Praticamente toda a documentação deve estar completa no fim desta etapa.

Na próxima fase, que é a de implementação, é feita a integração do produto ou serviço do projeto na organização. Geralmente nesta fase o projeto atinge a taxa máxima de utilização de recursos.

Por fim há a fase final de encerramento do projeto onde os recursos são realocados e novos produtos ou processos devem ser estabelecidos.

Na Figura 2.4 estas fases podem ser vistas frente ao consumo de recursos de cada uma.

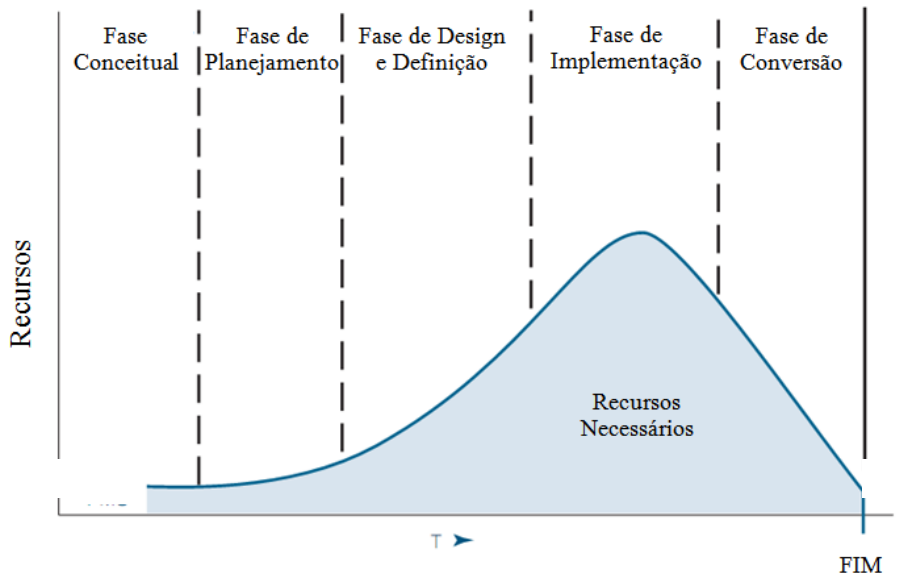

Figura 2.4 - Ciclo de Vida de Projetos (Kerzner, 2009)

2.3.1 **Modelo de ciclo de vida linear**

Este modelo também é conhecido como modelo clássico ou modelo cascata e sugere uma abordagem sistemática e seqüencial de lidar com o desenvolvimento de software.

Pressman (2003) ilustra o modelo com o diagrama da Figura 2.5, apresentando quatro fases que compõem este tipo de ciclo de vida.

Inicialmente há a fase de análise, onde os requisitos do sistema e do software são levantados e documentados.

Em seguida há a etapa de design, onde há um detalhamento das estruturas de dados, da arquitetura do software, das representações das interfaces e dos algoritmos que serão utilizados. Esta fase deve transformar os requisitos em uma representação que possa ter qualidade avaliada antes que a codificação tenha início.

A próxima etapa é a de codificação, sendo que o design deve ser traduzido para alguma linguagem que uma máquina possa compreender. Quanto mais detalhada a etapa de design menos trabalhosa se torna a etapa de codificação, já que os desenvolvedores não possuíram duvidas quanto às funcionalidades que devem ser implementadas.

Por fim há a fase final de testes, que deve avaliar todos os casos de utilização do sistema bem como comparar as saídas documentadas na etapa de design com as saídas reais do software.

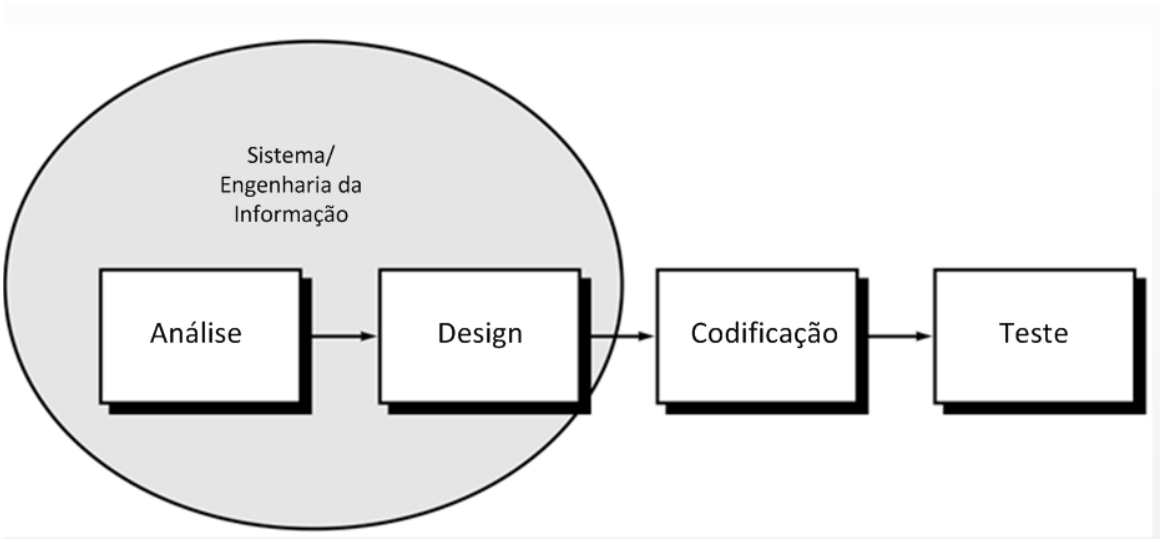

Figura 2.5 - Ciclo de vida linear (Pressman, 2003)

O modelo original, conforme proposto por Royce (1970), se mostrava mais simples em softwares pequenos, conforme ilustrado na Figura 2.6, e mais complexo para projetos de sistemas mais complexos, conforme mostra a Figura 2.7.

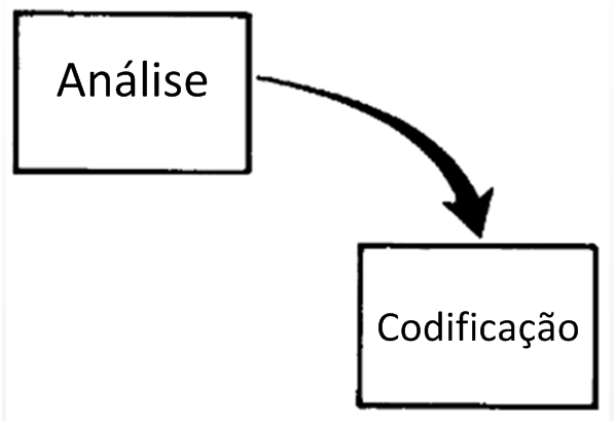

Figura 2.6 - Ciclo de vida linear em pequenos softwares (Royce, 1970)

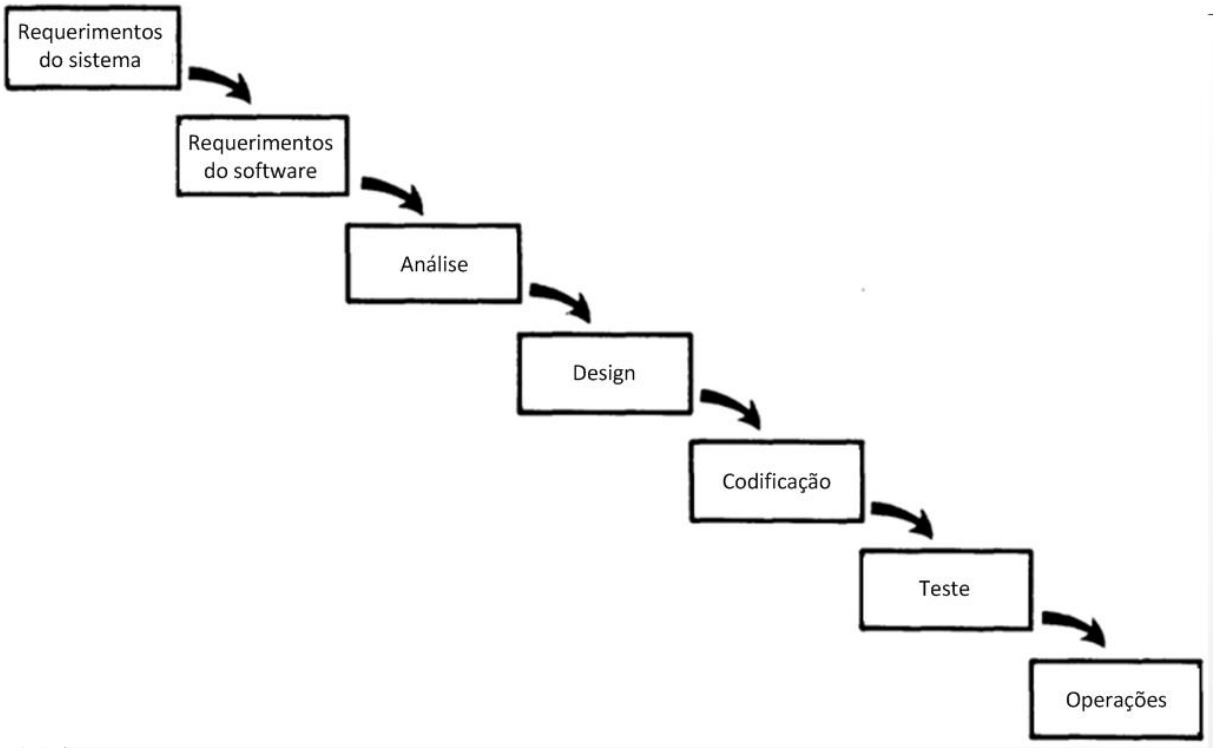

Figura 2.7 - Ciclo de vida linear em sistemas complexos (Royce, 1970)

O autor diz que nenhum dos passos adicionais que são requeridos para sistemas complexos contribuirá diretamente para o produto final da mesma maneira que os passos de análise e codificação contribuem, além do fato de que todos aumentarão os custos de desenvolvimento, porém não utilizá-los certamente levará o sistema ao fracasso.

Royce aponta também para as interações naturais que acontecem entre dois estágios sucessivos do ciclo de vida em cascata conforme mostra a Figura 2.8, já que conforme o

design tem seu detalhamento aprofundado por uma etapa anterior há uma fase de entendimento por parte da etapa posterior.

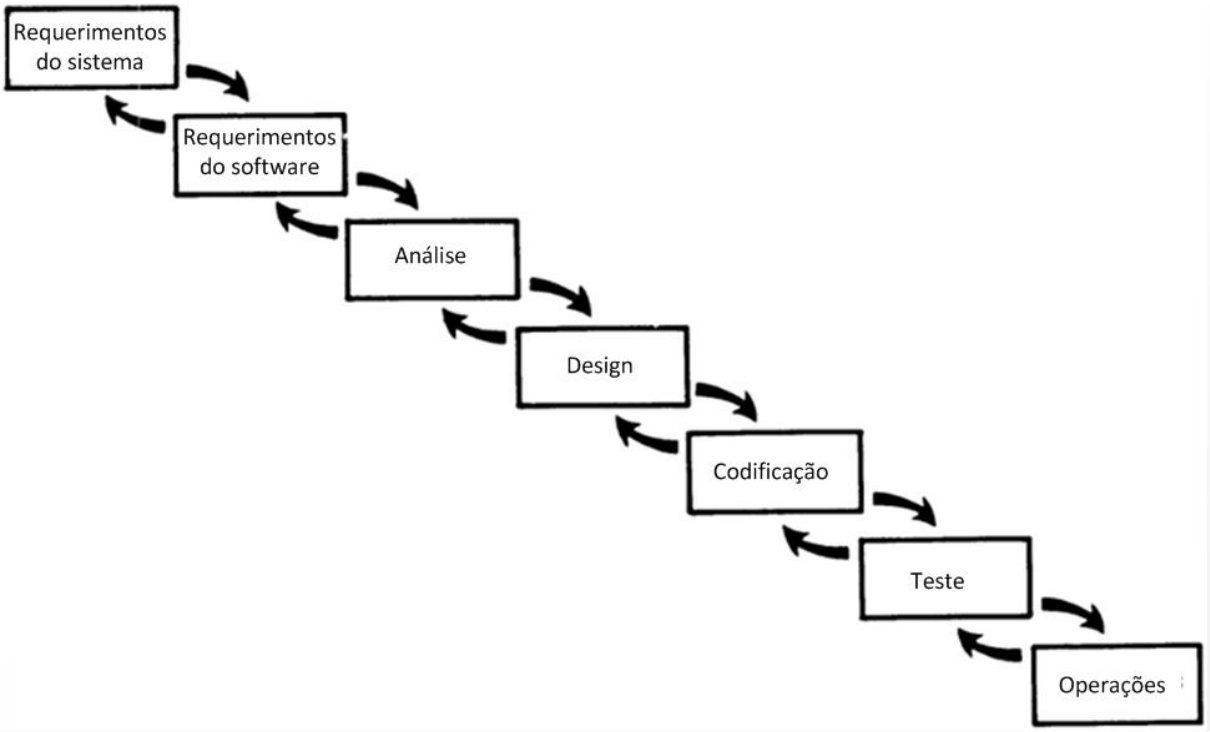

Figura 2.8 - Interações naturais no modelo cascata (Royce, 1970)

Além das interações naturais e necessárias previstas pelo modelo o autor cita também as interações indesejadas que ocorrem freqüentemente devido à natureza do mesmo. O problema mais comum, ilustrado na Figura 2.9, é a interação entre a etapa de testes, a de design do software e a de requisitos do software.

A fase de testes é um momento crítico no desenvolvimento do software, pois é quando se observam as primeiras discrepâncias entre o desempenho real do sistema e o desempenho previsto no design do mesmo. Tais falhas no planejamento podem requisitar mudanças drásticas no design do software. Estas mudanças por sua vez podem acabar alterando os requisitos do software já que novas tecnologias podem se tornar necessárias ou tecnologias já estabelecidas podem ter de ser substituídas por outras mais adequadas.

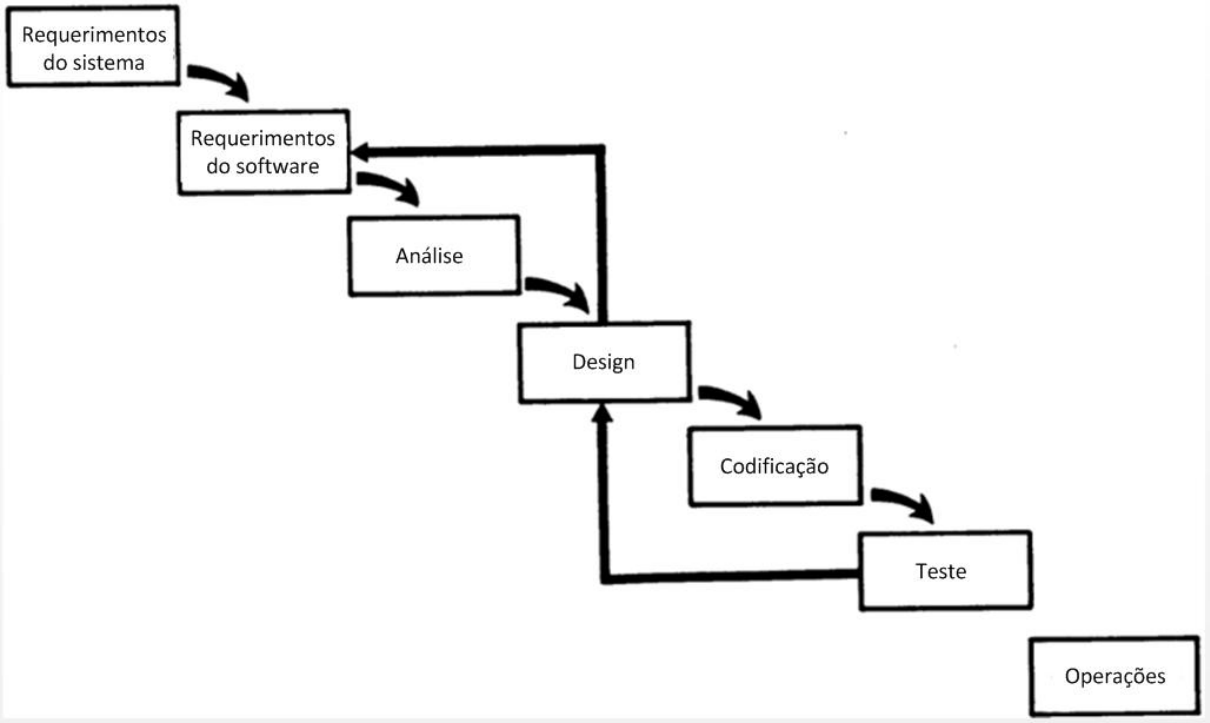

Figura 2.9 - Interação problemática no modelo cascata (Royce, 1970)

Apesar de ser o mais antigo e mais utilizado modelo de ciclo de vida na engenharia de software, o modelo linear tem sua eficácia questionada, segundo Pressman (2003), até mesmo por desenvolvedores que eram ativistas dele. Os problemas começam com o fato de que raramente um projeto possui um fluxo seqüencial conforme o modelo mostra. As interações entre etapas são acomodadas apenas indiretamente pelo modelo, tornando complexas e confusas as alterações durante o desenvolvimento do software. Outro problema levantado pelo autor é o fato de que o modelo requer do cliente que todos os requisitos sejam apresentados nas etapas de planejamento do sistema e não consegue acomodar as incertezas naturais com relação ao escopo do software. Por fim há o fato de que uma versão funcional do software só poderá ser apresentada nas etapas finais do desenvolvimento, tornando o modelo inadequado para clientes sem paciência.

Bradac apud Pressman (2003) levanta outro problema notado em projetos mais recentes. Conforme o ciclo é percorrido o projeto passa por fases de bloqueio quando alguma parte da equipe fica ociosa enquanto espera o término das atividades de outra parte da equipe. O autor nota que estas fases de bloqueio podem acabar tendo um tempo total superior ao tempo total de produção do projeto.

2.3.2 **Modelo de ciclo de vida incremental**

Pressman (2003) apresenta o modelo incremental como uma combinação dos elementos do modelo de ciclo de vida linear com conceitos de desenvolvimento iterativo de software.

Tal união pode ser vista no diagrama do modelo que é apresentado na Figura 2.10. O diagrama, que é representado na forma de um fluxograma em cima de uma escala de tempo, apresenta uma série de seqüências idênticas ao que é proposto pelo modelo linear. A diferença reside no fato de que cada seqüência representa apenas o desenvolvimento de uma porção fracionária, porém funcional, do sistema que está sendo implementado, e não todo o sistema como propõe o modelo clássico.

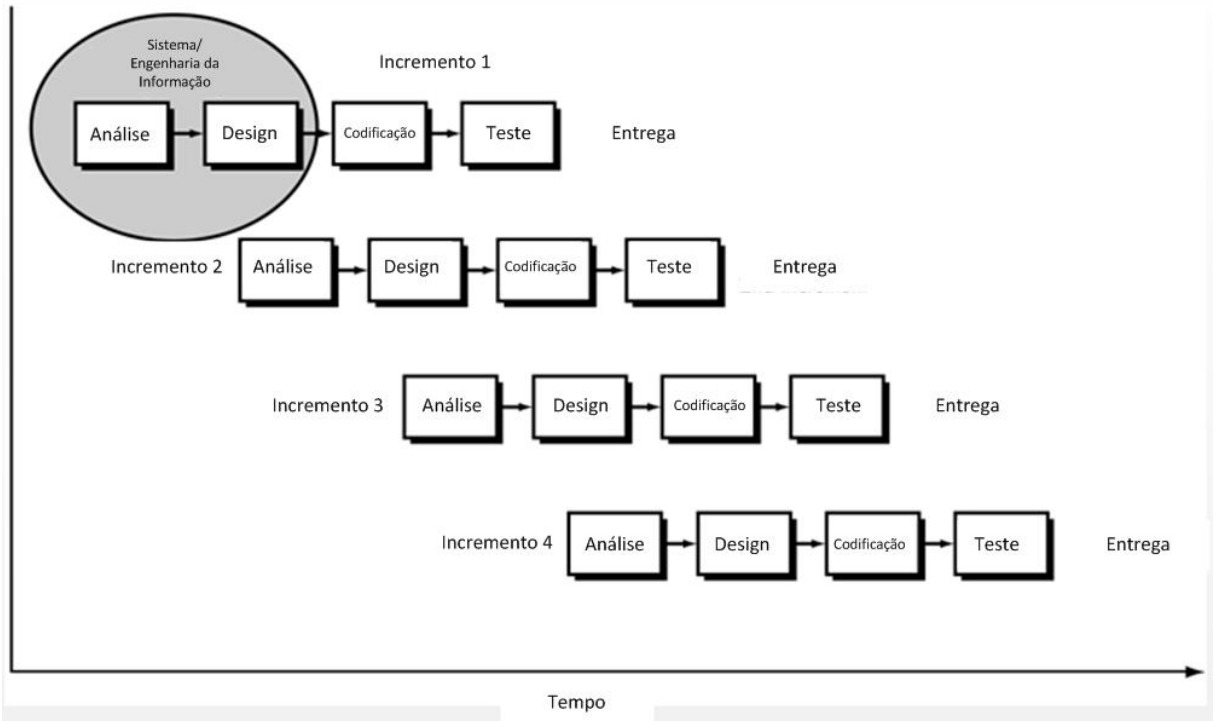

Figura 2.10 - Modelo Incremental (Pressman, 2003)

Este modelo apresenta um número maior de entregáveis do projeto, já que o ao fim de cada seqüência linear, a porção funcional do sistema em questão é entregue ao cliente, aumentando a presença do cliente na fase de desenvolvimento do projeto e diminuindo o risco de fuga do escopo.

Geralmente o primeiro incremento representa um núcleo primário do sistema, apresentado apenas uma série de funcionalidades básicas. No fim da primeira seqüência o cliente deve começar a utilizar este núcleo de maneira a auxiliar no planejamento do próximo

incremento. Este processo é então repetido até que o sistema tenha sido desenvolvido por completo.

O principal conceito presente no modelo de ciclo de vida incremental é o de entregas funcionais de partes do sistema fazendo com que o cliente possa avaliar o produto ainda durante a fase de desenvolvimento.

2.3.3 **Modelo de ciclo de vida espiral**

O modelo de ciclo de vida espiral proposto por Boehm (1988) apresenta um tipo de ciclo de vida que mesmo possuindo conceitos de desenvolvimento iterativo e tentando aprimorar o modelo linear ainda sim se mostra uma alternativa do modelo incremental apresentado acima.

Ainda são feitas seqüências de desenvolvimento em cadeia, cada uma baseada no modelo proposto por Royce (1970), porém não há o conceito de se desenvolver incrementos a cada iteração, e sim versões preliminares do software, que não necessariamente precisam ser funcionais até que se alcancem as etapas finais da metodologia

Conforme ilustrado no diagrama do modelo, apresentado na Figura 2.11, cada iteração passa por uma fase de análise da operação e dos requerimentos, seguida de um plano de desenvolvimento, culminando no desenvolvimento de um protótipo.

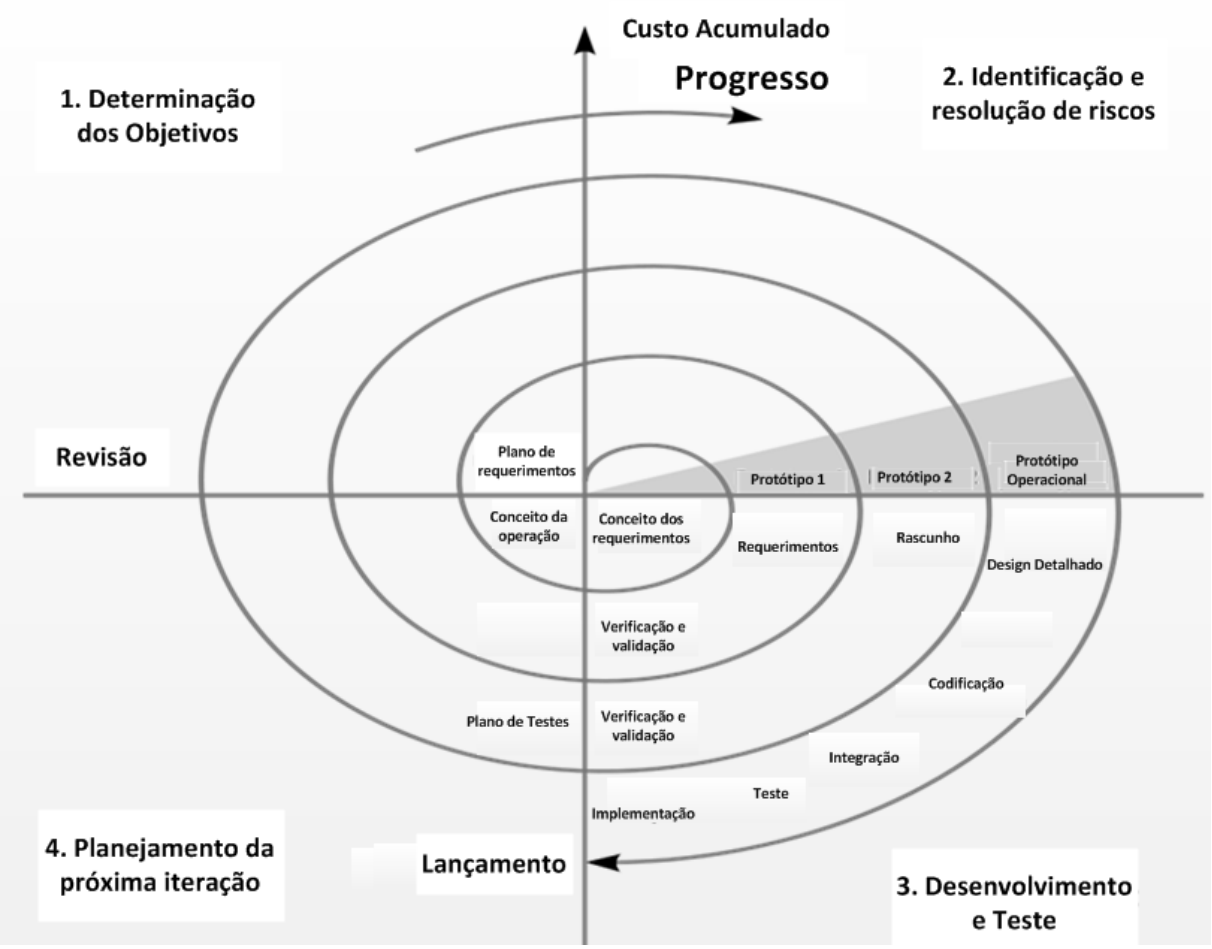

Figura 2.11 - Modelo Espiral (Boehm, 1988)

Em cada iteração do modelo estas fases se alteram de acordo com o escopo do protótipo a ser desenvolvido. As primeiras iterações podem ser iniciadas com uma análise simples dos requerimentos do sistema, enquanto a iteração final deve começar com um plano detalhado de todo o projeto.

Cada protótipo desenvolvido nas iterações representa uma versão preliminar do sistema como um todo, podendo ser iniciado como modelos em papel, passando por telas não funcionais até que se tenha uma versão funcional do sistema.

Pressman (2003) apresenta uma versão atualizada deste modelo propondo regiões de atividades que devem ser visitadas uma vez a cada iteração do modelo. O diagrama desta versão, para um desenvolvimento envolvendo seis regiões, está na Figura 2.12.

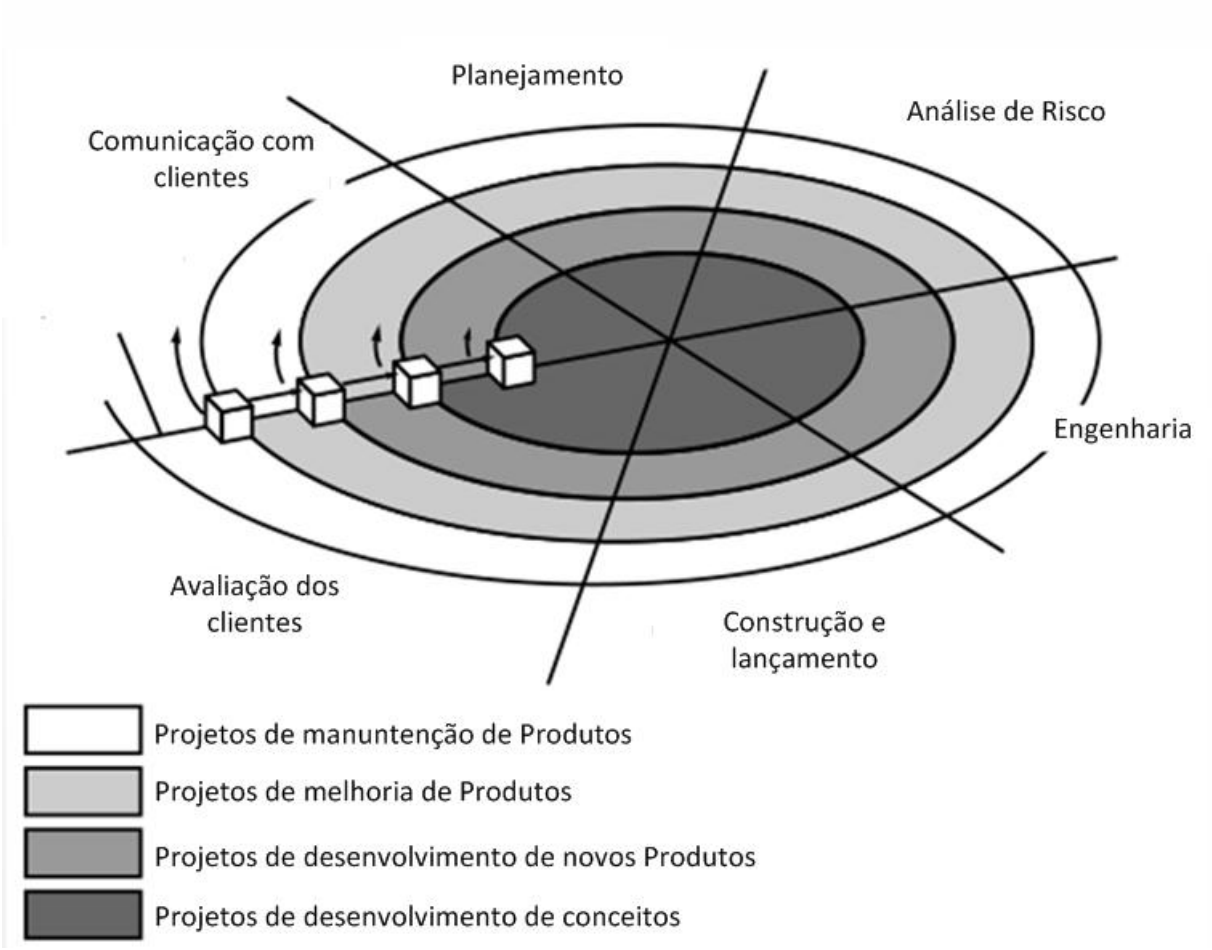

Figura 2.12 - Modelo Espiral revisitado (Pressman, 2003)

Primeiramente há o estágio de comunicação com o cliente, onde são realizadas tarefas que envolvem o estabelecimento de um contato entre ele e os desenvolvedores.

Em seguida há a região de planejamento, onde são realizadas atividades para a definição de recursos, prazos e outras informações relacionadas com o projeto.

A terceira região é a da análise de riscos, com atividades relacionadas a riscos técnicos e de gestão.

A seguir a iteração passa pela fase de engenharia, onde as atividades se focam em construir uma ou mais representações do sistema.

A quinta região é a de construção e lançamento, com atividades para construção, testes, instalação e suporte do usuário.

Por fim há a etapa de avaliação do cliente, formada de atividades que recuperam feedbacks do cliente relacionados com as experiências obtidas a partir das representações criadas na região de engenharia e implementadas na de construção e lançamento.

Concluindo, o modelo Espiral apresenta uma maneira realista para o desenvolvimento de sistemas grandes e complexos. Conforme o software evolui os riscos surgem e são analisados antes de causarem problemas que causem um grande impacto no orçamento do projeto. Essa faceta também uma das dificuldades do modelo já que ele exige pessoas capacitadas para lidar com a gestão de riscos de maneira altamente eficiente, além de que nem sempre o cliente estará convencido de que esta abordagem evolucionária é passível de controle.

2.4 **Engenharia de Software**

Bauer apud Pressman (2003) define engenharia de software como a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais.

Segundo o autor que apresentou esta citação esta definição apresenta questões importantes para o entendimento do conceito: Quais são os sólidos princípios de engenharia? Como construir softwares econômicos e como os tornar confiáveis? Como tornar um programa eficiente em várias máquinas reais?

De acordo com o glossário de engenharia de software apresentado no padrão 610.12 pelo IEEE (1990) a engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software, o que pode ser entendido como a aplicação da engenharia ao software.

Para Pressman (2003) a engenharia de software é uma tecnologia de camadas, conforme ilustrado na Figura 2.13, em que toda prática de engenharia deve possuir como base inicial a garantia da qualidade. Em seguida há a fundação para a engenharia de software que é a camada de processos. Ela mantém as camadas tecnológicas integradas e permite um desenvolvimento racional de software.

A camada de métodos oferece as maneiras de se desenvolver software, incluindo analises de requerimentos, esforços de design, e atividades de programação, teste e suporte.

Por fim, as ferramentas apresentam maneiras automatizadas para auxiliar os processos e métodos. Quando ferramentas são integradas de maneira com que a informação criada por uma ferramenta possa ser utilizada por outra é estabelecido um sistema de suporte para o desenvolvimento denominado “computer-aided software engineering” (CASE).

Figura 2.13 - Camadas da Engenharia de Software (Pressman, 2003)

Além das camadas que compõe este conceito o autor apresenta as três fases pelas quais as atividades da engenharia de software passam independentemente da área de aplicação, tamanho do projeto ou complexidade. Cada fase contempla questões que devem ser apresentadas para qualquer atividade de engenharia como “Qual é o problema que deve ser resolvido?” ou “Como a solução será desenvolvida?”.

A primeira fase é a de definição, já que durante ela a engenharia de software tenta identificar qual informação será processada, qual é o desempenho desejado, qual comportamento é esperado, quais interfaces serão estabelecidas e qual critério de avaliação será usado para identificar um sistema de sucesso. Apesar dos métodos aplicados nesta fase variarem dependendo do paradigma sendo utilizado três grande tarefas ocorrerão de alguma maneira: engenharia de informação ou sistemas, planejamento do projeto de software e análise de requerimentos.

A fase seguinte é a de desenvolvimento, e nela o engenheiro de software tenta definir como os dados serão estruturados, como detalhes dos procedimentos serão implementas, como as interfaces serão caracterizadas, como o design será traduzido para a linguagem de programação e como serão desempenhados os testes. Geralmente três atividades técnicas devem sempre ocorrer nesta fase, design de software, geração de código e teste de software.

A última fase é a de suporte, tendo como foco as mudanças associadas com correções de erros, adaptações necessárias durante o desenvolvimento e mudanças nos requisitos passados pelo cliente. São feitas mudanças de correção, para corrigir defeitos, adaptação, para adaptar o sistema em novos ambientes, aprimoramento, para adicionar novas funcionalidades e prevenção, para facilitar todos os outros tipos de mudanças futuras.

Para solucionar os problemas que irá encontrar o engenheiro de software precisa dominar uma estratégia de desenvolvimento que leve em conta as camadas e as fases

genéricas que foram descritas. Tal estratégia é denominada modelo de processo ou paradigma de engenharia de software.

2.5 **Desenvolvimento Ágil**

O termo que representa este conjunto de metodologias e conceitos tem o sentido de destreza, ou seja, a capacidade de alteração rápida de um determinado estado para outro (Abrahamsson et al, 2002) .

O desenvolvimento ágil, bem como a aliança ágil, foi definido em fevereiro de 2001 num evento que reuniu 17 desenvolvedores, gerentes e engenheiros de software para debater sobre as metodologias emergentes que tentavam propor alternativas ao modelo clássico de desenvolvimento (Cockburn, 2002). O produto deste evento é o chamado Manifesto Agile, presente no anexo 8.1.

O contexto desta reunião é um mercado de engenharia de software permeado de ceticismo com relação a novas metodologias que são difíceis de serem compreendidas e acabam não sendo utilizadas (Wiegers apud Abrahamsson et al, 2002). Os métodos que eram apresentados se mostravam como idealizações. Ao invés de se apresentarem como frameworks práticos, eles transmitiam idéias como se fossem cases e exemplos. (Trux et al, 2000).

O manifesto tem como enfoque uma lista de valores que foram levantados, debatidos e acordados entre todos os presentes e representam a base que guia e une todas as metodologias ditas ágeis.

O primeiro valor diz respeito ao favorecimento dos indivíduos e das interações sobre os processos e as ferramentas. Cockburn (2002), que foi um dos que desenvolveram o manifesto, afirma que este valor prega o hábito de tratar pessoas como membros de uma equipe e não como cargos em um diagrama de processos. Abrahamsson et al (2002) levanta o ponto de que as metodologias ágeis favorecem relacionamentos mais intensos entre pequenos times como forma de estimular as interações e o espírito de equipe.

O segundo valor presente no manifesto é o de favorecer software que esteja funcionando sobre uma documentação extensiva. Este valor se justifica pelo fato de que apenas o funcionamento do sistema ilustra o que a equipe construiu, enquanto que a documentação é uma maneira de se construir imagens futuras de próximos estágios de

desenvolvimento (Cockburn, 2002). Segundo Abrahamsson et al (2002) a questão da documentação tem sua prioridade diminuída frente ao constante esforço que a equipe deve possuir de manter o software testado e funcional.

O terceiro valor prega a colaboração com o cliente frente à negociação em contratos. Cockburn (2002) ilustra este valor dizendo que no desenvolvimento ágil não há a distinção entre quem desenvolve o software e quem o requisitou, e sim apenas uma ou mais empresas presentes num esforço integrado para a implementação da solução de um problema. Abrahamsson et al (2002) atenta para o fato de que ainda sim um contrato passar a ganhar mais importância frente a colaboração conforme o tamanho do sistema cresce.

O último valor é sobre dar mais importância para a resposta sobre uma mudança do que seguir algum plano. Cockburn (2002) cita que as metodologias ágeis aceitam alterações de prioridades entre ciclos de desenvolvimentos que normalmente tem apenas entre duas e quatro semanas. Estes ciclos curtos permitem a construção de um software funcional ao mesmo tempo em que favorece a mudança de prioridades por parte dos patrocinadores do projeto.

Segundo Highsmith and Cockburn apud Abrahamsson et al (2002), "o que é novo sobre os métodos agile não são as práticas utilizadas, mas seu reconhecimento de que as pessoas são as principais responsáveis pelo sucesso do projeto juntamente com um foco intenso na eficácia e na capacidade rápida de mudança. Isso culmina em uma nova combinação de valores e princípios que define uma visão do mundo de forma ágil."

Miller apud Abrahamsson et al (2002) cita os fatores que influenciam os processos ágeis de forma a diminuir o ciclo de vida dos projetos:

a) Modularidade no nível do processo de desenvolvimento.
b) Desenvolvimento iterativo através de ciclos curtos, permitindo verificações e correções rápidas.
c) Ciclos iterativos com duração entre uma e seis semanas.
d) Processos enxutos de desenvolvimento com a remoção de atividades desnecessárias.
e) Adaptabilidade frente a novos riscos emergentes.
f) Abordagem de desenvolvimento incremental que permite a construção de aplicativos sempre funcionais em pequenos passos.
g) Abordagem convergente e incremental minimiza os riscos.
h) Metodologia orientada a pessoas, as favorecendo sobre os processos e sobre a tecnologia.
i) Estilo de trabalho comunicativo e colaborativo.

Serão detalhadas nas próximas subseções duas metodologias ditas ágeis que compartilham os mesmos valores, porém apresentam práticas diferentes.

2.5.1 **Programação Extrema**

Segundo Beck apud Abrahamsson et al (2002), a Programação Extrema (XP) se desenvolveu devido aos problemas causados por longos ciclos de desenvolvimento dos modelos tradicionais. As práticas pregadas por esta metodologia não eram novas quando ela estava sendo desenvolvida, porém elas foram integradas de maneira que formaram uma nova maneira de se desenvolver software. O termo “extremo” vem de lidar com princípios comuns em níveis extremos.

Cohen et al (2004) enumera as doze regras da XP revelando que elas podem ser suficientes para o entendimento da metodologia:

a) **Jogo de planejamento**: No início de cada iteração de desenvolvimento os clientes, gerentes e desenvolvedores devem fazer uma reunião para estimar e priorizar os requerimentos da próxima entrega do projeto. Estes requerimentos são chamados de estórias de uso e são registrados em cartões de estória numa linguagem compreensível por todos os interessados do projeto.
b) **Entregáveis pequenos**: O projeto deve ser divido em pequenas entregas, sendo que uma versão inicial do sistema é posta em produção após as primeiras iterações inicias.
c) **Metáfora**: Os clientes, gerentes e desenvolvedores constroem metáforas para modelar o sistema e explicar seu funcionamento.
d) **Design simples**: Os desenvolvedores devem ser fortemente encorajados a manter o design do sistema tão simples quanto for possível.
e) **Testes**: Os desenvolvedores devem escrever testes de aceitação antes das funcionalidades em questão. Os clientes também devem escrever testes de funcionalidades que devem ser executados no fim de cada iteração.
f) **Refatoração**: Durante o trabalho de desenvolvimento, o design deve evoluir para que continuem tão simples quanto for possível.

g) **Programação em par**: Cada unidade de desenvolvimento deve ser composta por dois programadores compartilhando uma mesma máquina.
h) **Integração contínua**: A integração de código novo no sistema funcional deve ser freqüente. Os testes funcionais devem ser verificados após a integração ou o código novo deve ser descartado.
i) **Posse coletiva**: Todo o código é de posse coletiva de todos os desenvolvedores, sendo que eles podem fazer alterações em qualquer lugar sempre que for necessário.
j) **Cliente presente**: Algum representante do cliente deve trabalhar com a equipe de desenvolvimento em tempo integral para responder questões da equipe, efetuar testes de performance e assegurar que o progresso do desenvolvimento está ocorrendo conforme o esperado.
k) **Jornada de 40 horas semanais**: A lista de requerimentos de uma iteração deve ser selecionada de forma que não seja necessário realizar hora extra.
l) **Ambiente de trabalho comum**: Os desenvolvedores devem trabalhar em um ambiente comum com estações de trabalho individuais que compartilham máquinas de desenvolvimento no centro.

Segundo Abrahamsson et al (2002) o ciclo de vida da XP possui, conforme ilustrado na Figura 2.14, seis fases: Exploração, Planejamento, Iterações para a próxima entrega, Produção, Manutenção e Morte

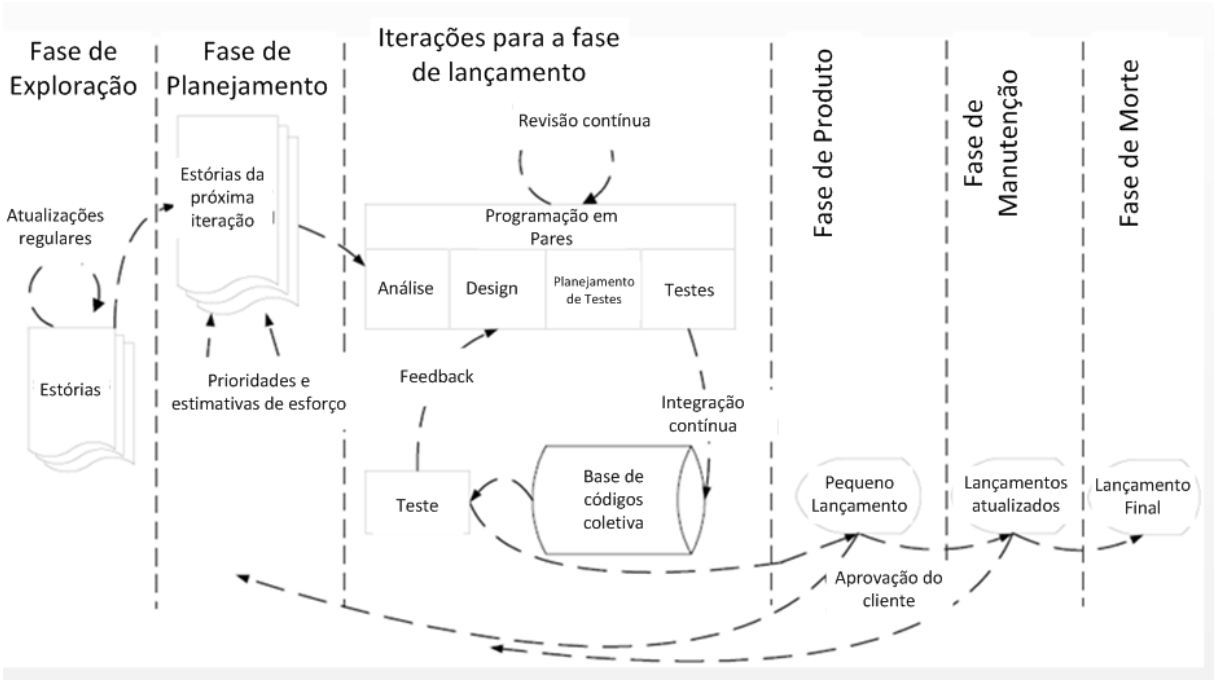

Figura 2.14 - Ciclo de vida da Programação Extrema (Abrahamsson et al, 2002)

Na fase de exploração os clientes determinam as estórias que devem ser incluídas na primeira entrega do projeto. Cada estória é registrada num cartão de estória que descreve a funcionalidade sendo requisitada. Ao mesmo tempo a equipe de desenvolvimento se familiariza uns com os outros, com a tecnologia de testes que será utilizada e com as possibilidades de arquitetura do sistema através da construção de um protótipo inicial.

Beck (2004) diz que o propósito da fase seguinte de planejamento é realizar um acordo entre os desenvolvedores e os clientes sobre o prazo de entrega da primeira iteração.

A fase de Iterações para a próxima entrega é a responsável por gerar um cronograma de desenvolvimento dividido em iterações com duração entre uma e quatro semanas. Beck sugere que a primeira iteração tenha funcionalidades que façam com que a arquitetura de todo o sistema seja definida. As iterações seguintes, segundo o autor, devem ter estórias escolhidas em uma ordem tal que as estórias que agreguem mais valor sejam desenvolvidas antes.

No decorrer das iterações deve ser verificado se o planejamento inicial está cumprido para que alterações necessárias no plano sejam implementadas.

Ao fim de cada iteração devem ser executados testes funcionais criados pelo cliente de forma a verificar que o sistema está pronto para ser posto em produção.

Na fase de produção a equipe deve se assegurar que o sistema está pronto para ser entregue através da criação de mais testes e da implementação de melhorias no desempenho.

Beck revela que iterações de duração curta (uma semana) podem ser necessárias já que nesta fase a opinião do cliente será um fator crítico para o sucesso.

Abrahamsson et al (2002) cita que nesta fase as funcionalidades adiadas devem ser documentadas para uma posterior implementação.

A fase de manutenção, segundo Beck, é o estado normal da metodologia. Ela se inicia com a primeira iteração após o sistema ter entrado em produção, ou seja, após o cliente começar a utilizar o software. O autor cita que grandes mudanças podem ser planejadas a partir desta etapa, já que os prazos são maiores e as experiências de uso são mais concretas. Também é dita a importância de se estabelecer um rodízio entre os desenvolvedores no suporte ao cliente, já que este canal de comunicação será usado com mais freqüência.

A fase da morte, segundo Abrahamsson et al, tem como principal sintoma a falta da necessidade de implementar novas estórias por parte do cliente. Isto mostra que o sistema satisfaz os requisitos do cliente tanto no quesito de funcionalidades quanto confiança e desempenho.

A documentação necessária é finalmente escrita já que não haverá mudanças no código.

Esta fase pode também ser alcançada caso o sistema não alcance os resultados esperados ou caso o desenvolvimento se torne muito custoso.

2.5.2 **SCRUM**

O termo “scrum” tem origem em uma estratégia no rugby utilizada para reiniciar a partida com trabalho de equipe após a bola ter saído do jogo (Schwaber and Beedle apud Abrahamsson et al, 2002).

Segundo Schwaber (2004), um dos co-autores da metodologia, o SCRUM possui todas as suas práticas penduradas em um esqueleto de um processo iterativo e incremental, ilustrado na Figura 2.15.

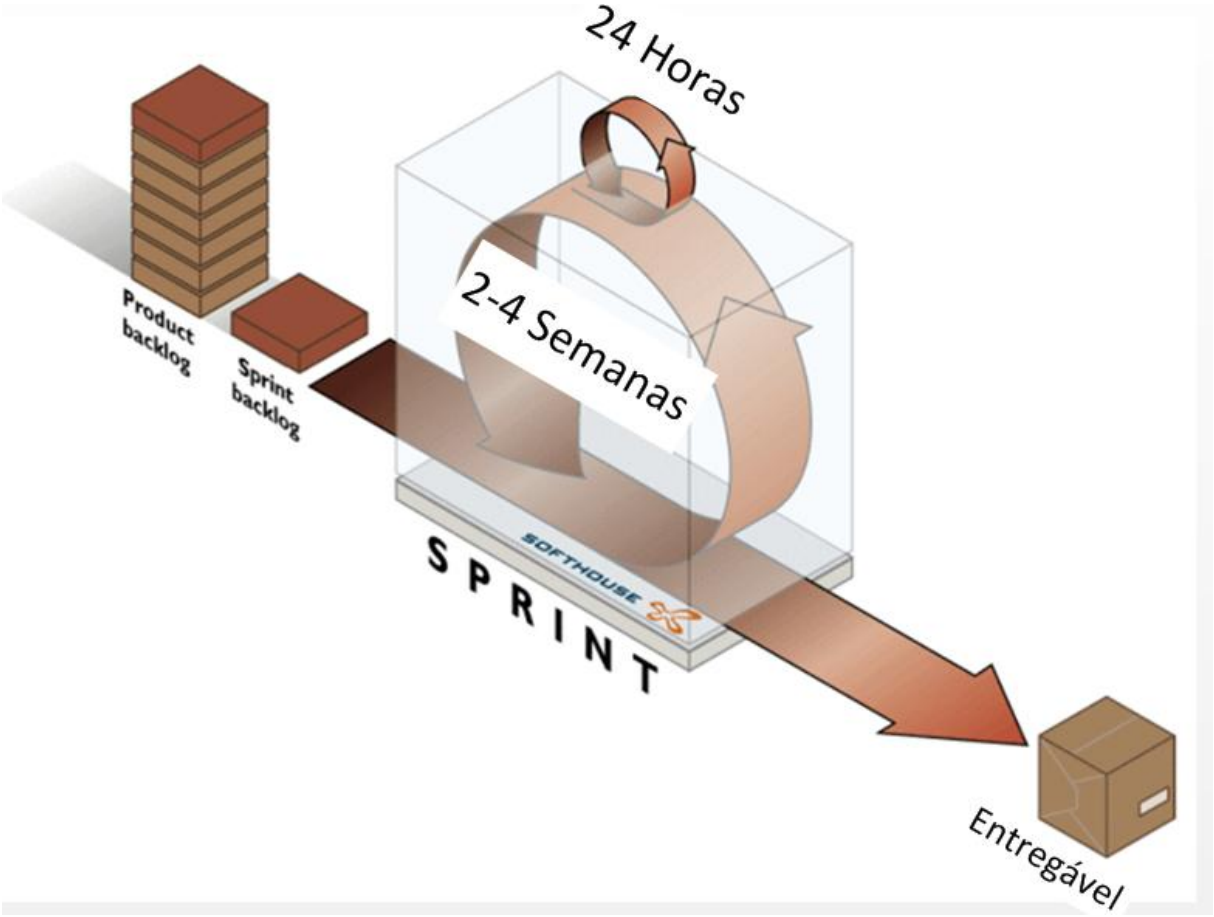

Figura 2.15 - Esqueleto do SCRUM (Sutherland e Schwaber, 2007)

Neste esqueleto há um ciclo inferior que diz respeito às iterações da metodologia, cada uma tendo como produto um incremento de funcionalidade no sistema. O ciclo superior representa a inspeção diária que é realizada dentro das iterações em que toda a equipe de desenvolvimento se encontra para inspecionar as atividades uns dos outros e realizar as adaptações necessárias. Como entrada de todo o processo há uma lista de requerimentos, e, por fim, as iterações se repetem até que não aja nenhuma funcionalidade a ser implementada.

O processo se resume da seguinte maneira:

a) No início de cada iteração o time revisa as funcionalidades que ainda devem ser implementadas.
b) São selecionadas então as funcionalidades que podem potencialmente formar uma porção do sistema funcional e passível de entrega no fim da iteração.
c) O time concentra todos os esforços no desenvolvimento durante todo o resto da iteração.

d) No fim da iteração, o time apresenta o incremento de funcionalidade construído para os stakeholders do projeto.
e) Uma nova iteração é iniciada conforme o primeiro passo.

Abrahamsson et al (2002) cita que a idéia principal desta metodologia reside no fato de que o desenvolvimento de sistemas envolve diversas variáveis técnicas e de ambiente que podem se alterar durante o processo de desenvolvimento. Isto torna o processo imprevisível e complexo, requerendo flexibilidade frente as possíveis mudanças (ou seja, a agilidade das metodologias ágeis).

A metodologia apresente três diferentes papéis: Membro da equipe de desenvolvimento, Scrum Master e Product Owner.

O Product Owner é o principal responsável pela lista de requerimentos do projeto, chamada de Product Backlog. Esta lista deve conter todas as funcionalidades que serão parcialmente selecionadas por cada iteração até que todas tenham sido implementadas. Este papel deve apresentar a visão inicial do sistema, bem como os objetivos financeiros relacionados com o retorno sobre o investimento e os cronogramas de entrega. O product owner deve garantir que as funcionalidades que agreguem mais valor sejam produzidas em primeiro lugar através da priorização adequadas dos itens no Product Backlog.

O Scrum Master é o responsável pelo processo e sua aderência à metodologia. Abrahamsson et al (2002) cita que ele deve remover quaisquer impedimentos que prejudiquem o desempenho da equipe de desenvolvimento. Schwaber (2004) complementa com o fato de que é o Scrum Master que deve ensinar as práticas do SCRUM, assegurar que elas estejam sendo aplicadas e analisar a melhor maneira de se aplicar esta metodologia no contexto da cultura organizacional em questão.

O membro da equipe, por fim, é responsável por alcançar os objetivos de cada iteração, que nesta metodologia é chamada de Sprint, através do desenvolvimento em si. A equipe, durante o decorrer de cada Sprint, deve ser auto-gerenciável de forma a encontrar a melhor maneira de transformar um item do Product Backlog em um incremento de funcionalidade.

O fluxo detalhado desta metodologia, conforme apresentado por Schwaber (2004), é ilustrado na Figura 2.16.

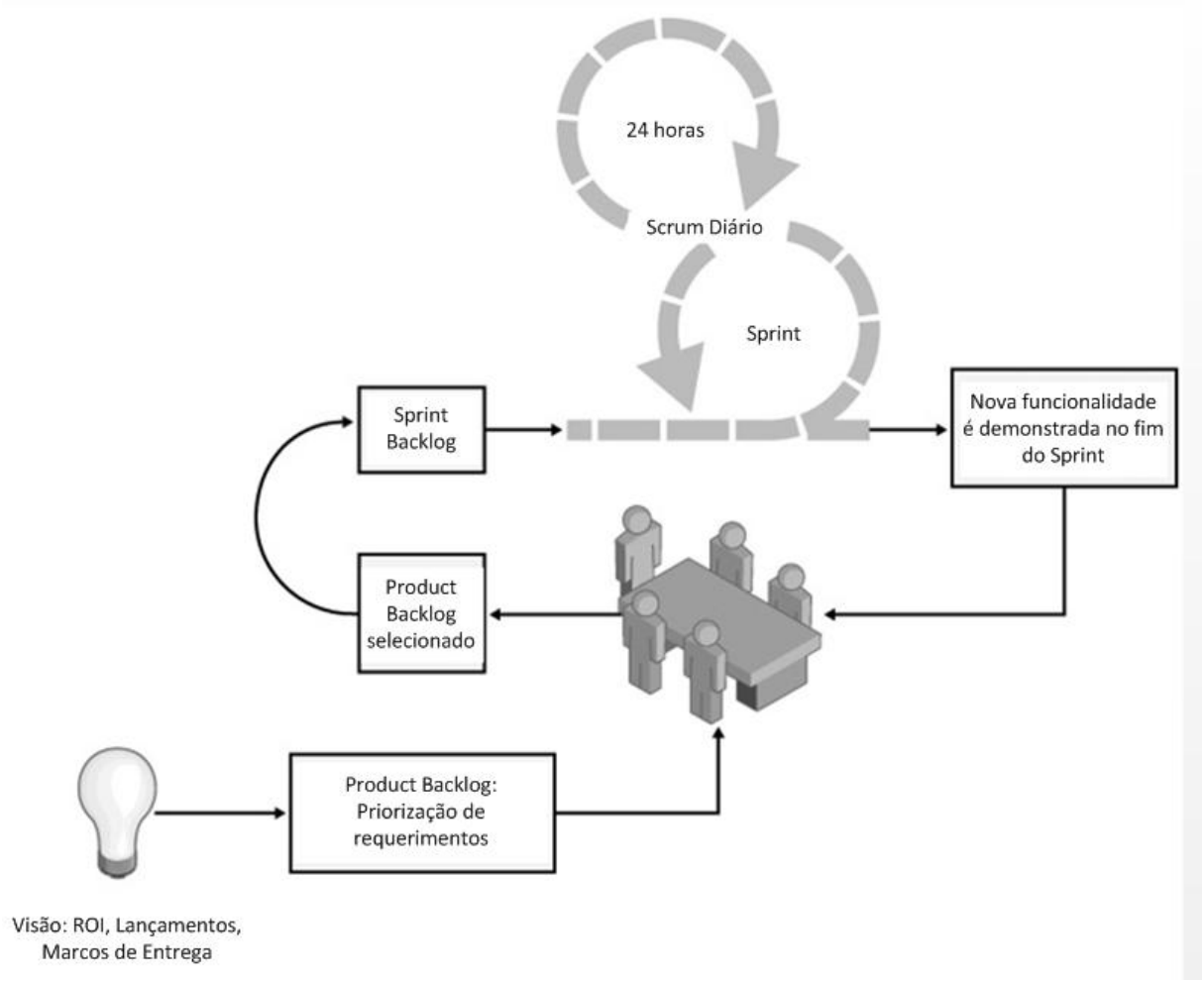

Figura 2.16 - Fluxo do SCRUM (Schwaber, 2004)

Inicialmente há a concepção da visão do sistema a ser desenvolvido. Esta visão ainda não é concreta, e pode ser descrita em termos de mercado ao invés de termos de sistema. O Product Owner deve desenvolver esta visão de forma a maximizar o retorno sobre o investimento neste projeto.

O resultado deste processo inicial é a geração do Product Backlog. Este elemento do SCRUM deve listar todos os requerimentos funcionais e não-funcionais que, quando forem implementados, irão compor o mesmo sistema que foi visionado inicialmente.

Conforme dito anteriormente, o Product Owner deve priorizar os itens do Product Backlog de forma que os itens que possuem maior probabilidade de gerar valor se encontrem no topo da lista de prioridades.

Esta lista de requerimentos priorizada é o ponto de partida para a próxima fase do fluxo da metodologia.

Antes do início do Sprint ocorre a reunião de planejamento do mesmo, onde tanto o Product Owner quanto a equipe de desenvolvimento trabalham colaborativamente para selecionar quais itens farão parte desta iteração. Partindo dos itens que agregarão mais valor, o Product Owner diz quais itens são desejados neste Sprint, e o time, em contrapartida, diz qual parcela destes itens pode ser implementada no período de tempo da iteração. Terminada esta etapa da reunião, o time deve desenvolver o Sprint Backlog, que é um documento semelhante ao Product Backlog, mas que contém as atividades que terão de ser desempenhadas para que o Sprint seja terminado.

Em cada dia do Sprint deve ser realizado o Daily Scrum, que é uma reunião de 15 minutos em que cada membro da equipe deve dizer o que ele fez desde o último Daily Scrum, o que ele fará até o próximo e quais os impedimentos que ele está encontrando. O propósito desta reunião é compartilhar o andamento do projeto com toda a equipe e repassar todos os impedimentos para o Scrum Master.

No término do Sprint, uma reunião de revisão do mesmo deve ser realizada. Nesta reunião o time deve apresentar para o Product Owner e outros stakeholders interessados tudo que foi desenvolvido durante o Sprint.

Antes de se iniciar a próxima iteração com mais uma reunião de planejamento de Sprint, o Scrum Master deve realizar uma reunião de retrospectiva do Sprint com a equipe. Nesta reunião o Scrum Master propõe que o time revise, dentro das práticas do SCRUM, o processo de desenvolvimento para que este seja mais efetivo e aproveitável no próximo Sprint.

Dada a falta de especificidade com relação à necessidade do uso de práticas técnicas, o SCRUM pode ser utilizado na gestão de qualquer tarefa de engenharia (Schwaber e Beedle apud Abrahamsson et al, 2002).

É valido notar, entretanto, que a divisão de papéis presentes na metodologia pode alterar drasticamente as responsabilidades originais dos cargos da empresa. O gerente de projetos, por exemplo, se torna o Scrum Master e não mais lidera o time, sendo que este é agora auto-gerenciável.

Abrahamsson et al (2002) cita que esforços para integrar o SCRUM com a Programação extrema tem sido feitos em várias áreas.

Enquanto o primeiro propicia um framework para a gestão ágil de projetos, o segundo fornece práticas para o desenvolvimento, formando um pacote integrado para os times de desenvolvimento de software.

3 ABORDAGEM METODOLÓGICA

3.1 **Considerações iniciais**

Este capítulo apresentará a abordagem metodológica que será utilizada para que sejam alcançados os objetivos deste trabalho.

A base desta fundamentação é o método da pesquisa-ação, um tipo de pesquisa qualitativa para abordagem de problemas orientada para a ação. Este método não será utilizado em completa conformidade com relação ao que é sugerido pela literatura, ele servirá de norte para definição da abordagem metodológica deste trabalho, sendo assim não necessariamente todos os passos previstos no modelo serão seguidos.

A pesquisa-ação, segundo Turrioni e Mello (2010), é um tipo de pesquisa social com base empírica que esta atrelada com uma ação ou com a resolução de um problema. Os autores citam que o termo “ação” esta presente para se referir à modificação da realidade causada pela aplicação deste método.

As características que definem este método, segundo os mesmo autores, são:

a) Utilização de abordagem científica para estudar a resolução de importantes assuntos sociais ou organizacionais juntamente com aqueles que experimentam esses assuntos diretamente.
b) Existência da colaboração entre membros do sistema sendo estudado e pesquisadores.
c) Presença de ciclos de coleta, realimentação e análise de dados, planejamento de ações, implementação e avaliação.
d) Condução da pesquisa, na maioria dos casos, em tempo real.

Esta integração dos pesquisadores com o meio sendo estudado e a busca pela transformação da realidade foram os fatores responsáveis pelo uso deste método de pesquisa neste trabalho.

Por fim, é valido classificar a modalidade de pesquisa-ação que será adotada com base nos tipos descritos na Tabela 3.1.

O tipo de pesquisa-ação mais adequada para o presente estudo é a modalidade técnica, já que o principal objetivo deste trabalho esta relacionado com o desenvolvimento profissional. As outras modalidades foram descartadas por possuírem objetivos muito

grandiosos como a transformação da organização, ou objetivos que não se aplicam neste caso, como a transformação da consciência, já que no caso em questão os membros da empresa já possuem a mentalidade necessária, sendo precária a presença de ferramentas e metodologias para servirem de base para o desenvolvimento.

Tipo de pesquisa-ação	Objetivos	Papel do pesquisador	Relacionamento entre pesquisador e participantes
1. Técnica	• Eficácia/eficiência da prática profissional • Desenvolvimento profissional	• Especialista externo • Toma uma prática existente de algum outro lugar e a implementa em sua própria esfera de prática para realizar uma melhora (age de forma mecânica)	• Co-opção (dos praticantes que dependem do pesquisador)
2. Prática	• Compreensão dos praticantes • Transformação da consciência dos praticantes • Além dos objetivos do tipo 1	• Papel socrático, encorajando a participação e a autorreflexão • Escolhe ou proteja as mudanças feitas	• Cooperação (“consultoria” do processo)
3. Emancipatória	• Emancipação dos participantes das regras de tradição, autodecepção e coerção • Sua crítica da sistematização da burocracia • Transformação da organização e seus sistemas • Além dos objetivos dos tipos 1 e 2	• Moderador do processo (responsabilidade compartilhada igualmente com os participantes)	• Colaboração (comunicação sistemática)

Tabela 3.1 - Modalidades da Pesquisa-ação (Tourrini e Mello, 2010)

A seqüência para a condução da pesquisa-ação é apresentada no diagrama da Figura 3.1 e consiste em um ciclo de cinco fases, sendo que em paralelo há a metáfase de monitoramento.

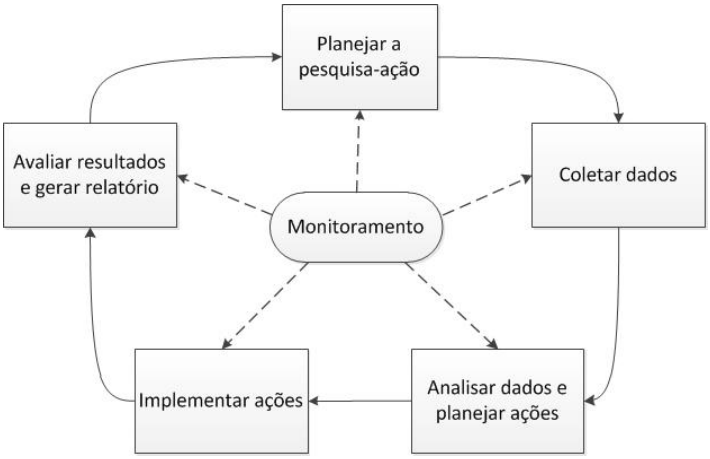

Figura 3.1 - Estrutura da condução da pesquisa-ação (Tourrini e Mello, 2010)

O detalhamento de cada etapa esta presente na Figura 3.2.

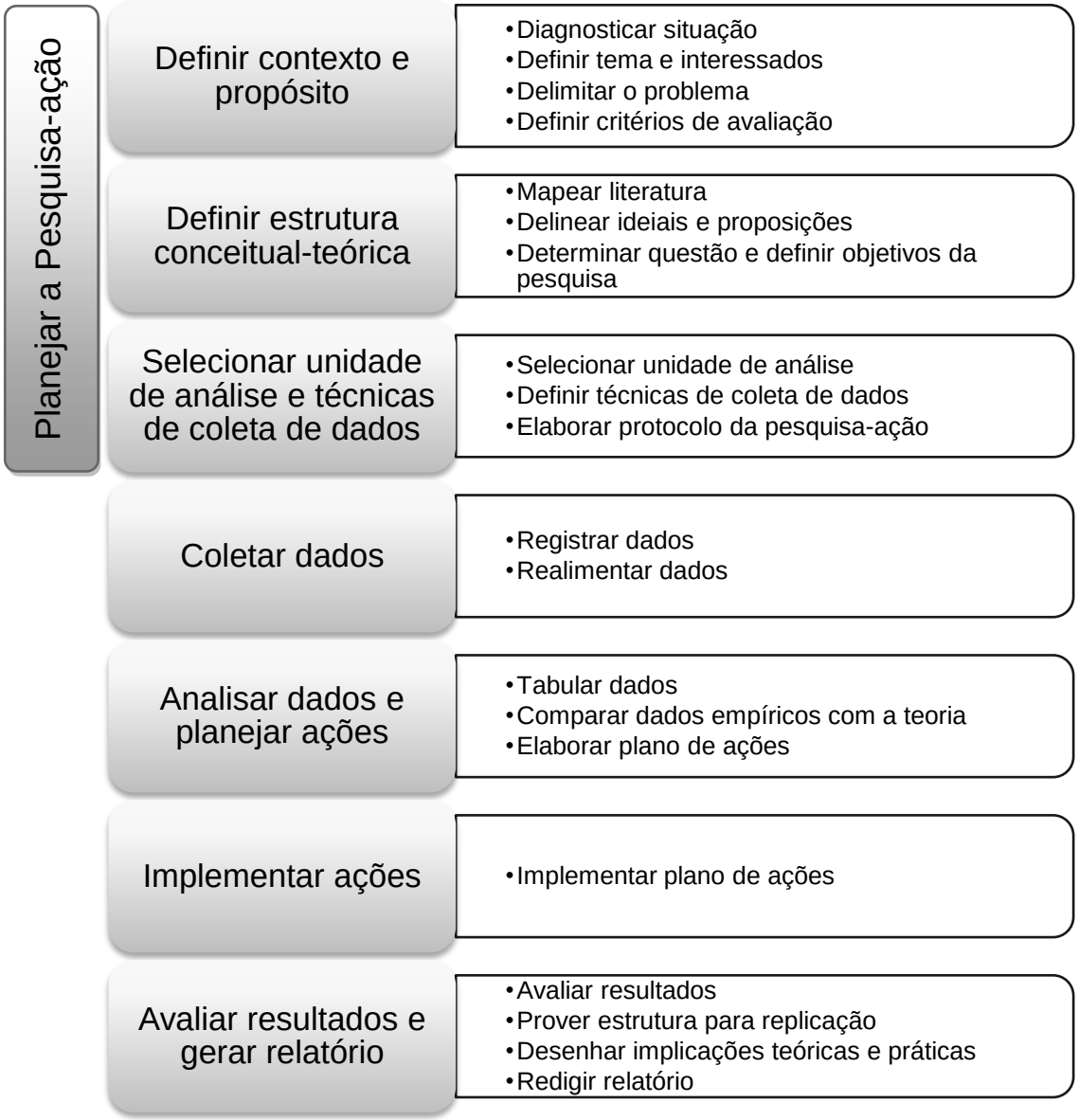

Figura 3.2 - Detalhamento das etapas da pesquisa-ação (Tourrini e Mello, 2010)

3.2 **Planejar a Pesquisa-ação**

Esta etapa de metodologia de pesquisa se inicia com a definição do propósito da pesquisa, além da verificação de sua real necessidade.

Esta técnica será aplicada com o intuito de se estabelecer melhorias no processo de desenvolvimento de software da empresa alvo do estudo para que os problemas levantados anteriormente sejam eliminados. Estas melhorias estarão estruturadas na forma de metodologias de gestão de projetos de software bem como de desenvolvimento, conforme as metodologias presentes na referência bibliográfica.

A análise da necessidade da pesquisa é equivalente à introdução deste trabalho, em que o contexto atual foi apresentado para justificar a necessidade das proposições deste trabalho.

A revisão da literatura, parte desta etapa da pesquisa-ação, foi realizada no capítulo anterior. Demo apud Turrioni e Mello (2010) cita que este referencial teórico deve ser utilizado para se levantar proposições sobre a pesquisa e para sustentar a argumentação destas hipóteses.

A hipótese mais importante que pode ser levantada no momento é a da adoção de uma metodologia de desenvolvimento ágil por parte da empresa. Esta classe de metodologias tem como características ciclos curtos de desenvolvimento encadeados, facilitando o planejamento e gerenciamento que hoje não ocorre na empresa e permitindo uma gestão de escopo mais flexível, o que corrige outro problema levantado anteriormente com relação a este tópico.

Com base no propósito da pesquisa e na revisão bibliográfica realizada é possível definir a questão de pesquisa deste estudo, que compreende a enumeração dos objetivos gerais e específicos deste projeto de pesquisa.

A questão pesquisa neste estudo é como seria uma metodologia de desenvolvimento de software adequada para a correção dos problemas presentes na empresa em questão. Os objetivos gerais desta pesquisa são:

a) Proporcionar uma metodologia para guiar a gestão de novos projetos de software
b) Gerar um novo foco de conhecimento de gestão por parte dos membros da área de desenvolvimento

Os objetivos específicos relacionados são:

- Apresentar uma maneira de se realizar a gestão de escopo nos projetos.

- Preparar os membros das equipes para possíveis mudanças nos escopos do projeto através da adição desta questão na metodologia sendo desenvolvida.
- Elaborar técnicas para diminuir a taxa de falha dos sistemas (erros de cálculo, omissão de validações e funcionalidades).
- Diminuir o retrabalho nos projetos através de práticas embutidas na metodologia.

Para a finalização desta etapa é feita a definição das técnicas de coleta de dados, que no caso se resumirá à técnica de observação participante. Segundo Turrioni e Mello (2010) esta é a técnica mais empregada na pesquisa-ação e consiste na participação real do pesquisador no grupo objeto da pesquisa, o que é condizente com a situação em questão em que o pesquisador realiza um estágio na empresa.

3.3 **Coletar Dados**

A coleta de dados neste caso se resume a analise do contexto em que se encontra a empresa, realizada no capítulo introdutório deste trabalho.

Além da coleta dos dados que foi realizada, há uma etapa de realimentação que é feita, com reuniões informais com membros da empresa para apresentação da analise do contexto que foi realizada.

3.4 **Etapas subseqüentes da pesquisa-ação**

As etapas de análise de dados e planejamento de ações e implementação do plano de ação serão descritas no capítulo seguinte, onde o contexto da empresa será confrontado com os conceitos presentes na literatura de forma a apresentar um plano de ação na forma de um projeto de metodologia de desenvolvimento.

Por fim, a etapa de avaliação de resultados será descrita no capítulo posterior, onde serão apresentados os primeiros resultados coletados.

4 PROPOSTA

4.1 **Análise do contexto**

A primeira questão citada na parte de definição do contexto encontrado na empresa é sobre a falta de planejamento nas fases iniciais do projeto da empresa. Espera-se que com a adoção da proposta de metodologia sendo desenvolvida este problema seja resolvido já que serão apresentadas práticas específicas para esta fase no ciclo de vida de um projeto de software.

Em seguida foi citado o problema que é o mais crítico no atual contexto, o da falta de gestão do escopo. A definição do escopo realizada para os projetos de software é fraca e não consegue alinhar as diferentes visões presentes nas equipes.

Além deste fato, há uma freqüente mudança de escopo no decorrer do projeto, que provavelmente tem como causa a fraca definição inicial e que acaba aumentando os custos do projeto devido ao retrabalho que se torna necessário para adequar o sistema sendo desenvolvido aos novos requisitos dos clientes.

A falta de uma definição concreta do escopo, conforme citado por Pressman (2003), faz com que não seja possível obter um controle sobre o progresso do projeto.

Conforme apresentado na referência bibliográfica, as metodologias ágeis têm como característica básica a capacidade de alteração rápida de um estado para outro, as tornando altamente recomendáveis para o ambiente em que a empresa em questão se encontra. O último valor apresentado no manifesto que define os princípios desta categoria de metodologias trata exatamente sobre esta questão da importância de se estar preparado para mudanças no plano definido inicialmente, dizendo que isso deve possuir uma prioridade mais elevada do que seguir o plano em si.

As alterações no escopo e no planejamento fazem parte das metodologias ditas ágeis, ocorrendo entre os períodos de tempo das iterações de desenvolvimento, permitindo o desenvolvimento de um sistema sempre funcional e alinhado com o escopo atual, representativo das necessidades do cliente.

4.2 **Considerações iniciais**

Neste item serão descritos os papéis, termos e documentos utilizados na proposta. Alguns papéis possuem membros fixos na empresa, enquanto que outros devem ser nomeados de acordo com o projeto em questão.

4.2.1 **Papéis**

4.2.1.1 Gerente Comercial

O gerente comercial é o responsável pelo projeto antes dele entrar na etapa de planejamento e implementação. Ele deve se assegurar de que uma minuta do projeto tenha sido elaborada e aprovada pelo cliente para que a fase inicial de planejamento possa partir de um escopo predefinido.

Este papel é desempenhado pelo CEO da empresa, que sempre lidou com esta etapa nos projetos realizados.

4.2.1.2 Product Owner

O Product Owner é aquele que deve garantir que o trabalho sendo desempenhado pela equipe agregue valor para o cliente.

Ele é o principal responsável pelo Product Backlog, sendo que ele deve mantê-lo disponível para todos os outros membros da equipe. A priorização das tarefas neste documento é de exclusiva responsabilidade do Product Owner, já que apenas ele deve representar o ponto de vista do cliente durante o desenvolvimento.

Este papel, na maioria dos casos, deve ser desempenhado pelo Diretor de P&D da empresa, já que sempre foi de sua responsabilidade cuidar dos aspectos técnicos dos projetos.

O sucesso do Product Owner depende do quanto suas ações e decisões são respeitadas pelos outros membros da empresa, do mesmo jeito que ele deve respeitar opiniões como as de estimativas de tarefa feitas pela equipe de desenvolvimento.

Sutherland e Schwaber (2007) citam as seguintes responsabilidades do Product Owner:

a) Definição das funcionalidades do produto,data de lançamento e conteúdo
b) Garantir a lucratividade do produto
c) Priorizar as funcionalidades de acordo com o valor de mercado
d) Alterar funcionalidades e prioridades a cada 30 dias caso seja necessário
e) Aceitar ou rejeitar os resultados apresentados pela equipe de desenvolvimento

4.2.1.3 Scrum Master

O Scrum Master assume o papel de líder da equipe de desenvolvimento, porém não é o análogo do gerente de projetos de uma metodologia tradicional aderente ao PMBoK, já que a equipe neste caso deve ser auto-gerenciável.

Ele deve garantir que a equipe siga os valores, práticas e regras do Scrum descritas no item 2.5.2, sendo que é sua responsabilidade auxiliar a empresa na adoção do Scrum.

Durante o desenvolvimento dos projetos, o Scrum Master deve mostrar como a equipe pode ser mais produtiva e como eles podem produzir sistemas com mais qualidade.

O Scrum Master deve fazer o papel de interface da equipe de desenvolvimento com o resto da empresa e com outros fatores externos. Isto significa que ele deve remover qualquer impedimento que possa prejudicar o ritmo de desenvolvimento de algum membro além de servir como escudo de interferências externas que poderiam afetar a equipe (Sutherland e Schwaber, 2007).

Rising e Janoff (2000) citam que o Scrum Master deve: ser responsável por manter as reuniões diárias curtas e dentro do foco que será descrito no item 4.5.3, se assegurar de que a equipe esta fazendo progresso e gravar as decisões de design feitas em reuniões.

Este papel será desempenhado pelo autor deste trabalho, já que é o Scrum Master o responsável por transmitir os conhecimentos de metodologias ágeis para a equipe.

4.2.1.4 Equipe de desenvolvimento

A equipe de desenvolvimento é formada por profissionais que são responsáveis por construir o produto requisitado pelo cliente.

Segundo Sutherland e Schwaber (2007), a equipe deve ser formada por membros de todas as áreas necessárias para a construção do sistema, sendo que além de programadores podem ser necessários pesquisadores, designers de interface e etc.

Os autores também citam a importância da interação entre a equipe e o Product Owner, que facilita a transformação dos desejos do cliente em requisitos de software.

Não há nenhum papel na metodologia que pode dizer como as estórias requisitadas pelo Product Owner poderão se tornar elementos incrementais do sistema, sendo que apenas a equipe pode se auto-gerenciar, com cada membro utilizando suas competências e a sinergia com o resto da equipe para conseguir resolver os problemas durante o desenvolvimento.

4.2.2 **Termos e documentos**

4.2.2.1 Product Backlog

O Product Backlog, apresentado na Figura 4.1, é um documento que contém uma lista com todos as requisições de funcionalidades que devem ser implementadas pela equipe de desenvolvimento. Seu conteúdo, disponibilidade e priorização de tarefas são de responsabilidade exclusiva do Product Owner.

Este documento e sua flexibilidade com relação a mudanças tornam a metodologia preparada para mudanças de escopo e para as evoluções naturais dos conceitos do sistema no decorrer do projeto.

Cada tarefa do Product Backlog deve possuir um campo que indica sua prioridade, para que as tarefas no topo da lista de prioridades sejam implementadas antes de outras.

Estoria	Prioridade	Estimativa
Estoria 1	X	Y
Estoria 2	X	Y
Estoria 3	X	Y
Estoria 4	X	Y

Figura 4.1 - Modelo de Product Backlog

4.2.2.2 Sprint

O Sprint é o período de tempo fixo em que a equipe se concentrará para desenvolver uma parte das tarefas do Product Backlog.

Este período garante o aspecto iterativo da metodologia, sendo que ao final de cada iteração é feita uma demonstração funcional do sistema para os representantes do cliente de modo a possibilitar um realinhamento das expectativas do cliente com a visão do sistema por parte da equipe.

Cada Sprint possui um escopo fixo, uma lista de tarefas bem definida, pessoas responsáveis por cada atividade e reuniões diárias que facilitam o fluxo de informação na equipe.

Ao término de cada Sprint a equipe deve realizar uma reunião de demonstração para os representantes do cliente e uma reunião de retrospectiva que enumera aspectos positivos que devem continuar nos próximos Sprints e aspectos negativos que precisam ser melhorados.

4.2.2.3 Sprint Backlog

O Sprint Backlog, apresentado na Figura 4.2, é um documento que contém de forma detalhada as tarefas que farão parte do próximo ciclo iterativo de desenvolvimento.

Este documento detalha cada tarefa de forma a diminuir o impacto negativo da ocorrência de dúvidas sobre determinada tarefa durante o desenvolvimento da mesma por parte de membros da equipe de desenvolvimento.

Cada tarefa também pode ser subdividida em atividades menores para facilitar o controle do projeto ou a integração com ferramentas de gestão que são utilizadas na empresa.

Estoria	Prioridade	Estimativa	Atividade	Responsável
Estória 1	X	Y	Atividade 1	XXXX
			Atividade 2	XXXX
			Atividade 3	XXXX
			Atividade 4	XXXX
			Atividade 5	XXXX
			Atividade 6	XXXX
Estória 2	X	Y	Atividade 1	XXXX
			Atividade 2	XXXX
			Atividade 3	XXXX
			Atividade 4	XXXX
			Atividade 5	XXXX
			Atividade 6	XXXX
Estória 3	X	Y	Atividade 1	XXXX
			Atividade 2	XXXX
			Atividade 3	XXXX
			Atividade 4	XXXX
			Atividade 5	XXXX
			Atividade 6	XXXX

Figura 4.2 - Modelo de Sprint Backlog

4.3 Etapa de planejamento inicial do projeto

4.3.1 **Primeira reunião de desenvolvimento do projeto**

A primeira reunião interna sobre o projeto tem como objetivo repassar para a equipe e para o Scrum Master as definições gerais sobre o produto que será desenvolvido.

A presença do gerente comercial na reunião é obrigatória, já que ele será o responsável por repassar para os membros da equipe de desenvolvimento todo o andamento do projeto até este ponto.

O escopo inicial já deve ter sido definido juntamente com o cliente através de uma minuta de projeto, de forma que os membros da equipe de desenvolvimento possam perceber de forma clara quais são os objetivos gerais do projeto.

Esta reunião também deve designar todos os papéis entre os interessados do projeto, definindo o Scrum Master, o Product Owner, e a equipe de desenvolvimento.

A reunião deve se aprofundar até certo nível que permita ao Scrum Master e à equipe elaborar um esboço de Product Backlog, que será refinado em uma reunião posterior. Sendo assim, devem ser discutidas possíveis divisões em módulos do sistema sendo desenvolvido e questões básicas de arquitetura e ambiente, como o sistema operacional alvo e a plataforma de execução (software para desktop, sistema web, etc.).

A reunião estará concluída quando os membros da equipe de desenvolvimento e o Scrum Master estiverem assegurados que possuem informações suficientes para iniciar o ciclo iterativo de desenvolvimento, quando a data e o horário da reunião de definição inicial do Product Backlog tiverem sido agendados e quando os meios de comunicação da equipe e do Scrum Master com o Product Owner tiverem sido claramente definidos

4.3.2 **Definição inicial do Product Backlog**

A reunião para definição inicial do Product Backlog tem como objetivo a criação de uma versão inicial do Product Backlog, documento que servirá de base para a designação de tarefas para todos os ciclos de desenvolvimento e que define o progresso da fase de implementação do projeto.

A presença do Product owner, do Scrum Master e da equipe de desenvolvimento são obrigatórias, já que eles serão os responsáveis pelo preenchimento do documento.

O Product Owner é o dono deste documento, sendo que após a reunião ele será o responsável por torná-lo disponível para os interessados, priorizar as tarefas contidas nele e atualizá-lo.

Durante a reunião os participantes devem utilizar o modelo de Product Backlog (apresentado na Figura 4.1) e, com algum programa de planilha eletrônica, criar registros das tarefas iniciais que devem ser desempenhadas no projeto.

Cada registro, que pode ser chamado de estória, representa alguma funcionalidade aliada com algum uso do sistema por parte do usuário final. Sendo assim são exemplos de nomes de estórias: “Acessar sistema remotamente”, “Cadastro de novo produto”. Nomes técnicos como “Indexação do banco de dados para aumento de performance” devem ser evitados através de um questionamento feito para o Product Owner sobre o motivo desta estória ser necessária, o que resultará em um nome mais adequado.

O campo “Prioridade” de cada estória deve ser preenchido apenas pelo Product Owner, já que apenas ele deve representar o ponto de vista do cliente. Este é um campo numérico que apenas ilustra se uma estória possui uma prioridade maior ou menor do que outra, sendo que uma tarefa de prioridade quatro não é necessariamente duas vezes mais importante que uma de prioridade dois.

É de exclusiva responsabilidade da equipe de desenvolvimento a definição do valor do campo “Estimativa”, que também é um campo numérico. O valor deste campo deve representar uma estimativa do tempo que a equipe acredita ser necessário para que esta estória possa ser desenvolvida de forma a estar funcional para ser apresentada para o Product Owner. A unidade deste campo é homem · dia.

O Product Backlog é um documento que sempre está em um estado incompleto, e vai sendo atualizado conforme o escopo do projeto evolui para definições mais concretas e novas funcionalidades são adicionadas, sendo assim os participantes desta reunião não devem se preocupar em deixar o documento completo.

A reunião pode ser encerrada quando houver estórias suficientes para a montagem de um Sprint Backlog de pelo menos três semanas, que é o tamanho mínimo que pode ser designado para o ciclo de desenvolvimento

4.4 Início do ciclo de desenvolvimento

4.4.1 Definição do Sprint Backlog

A reunião para definição do Sprint Backlog do próximo sprint só pode ter início quando o Scrum Master estiver certo de que o Product Backlog contenha estórias não implementadas em número suficiente.

Esta reunião deve possuir a presença obrigatória do Scrum Master, do Product Owner e da equipe de desenvolvimento.

O objetivo da reunião é a construção do Sprint Backlog, documento que define as estórias que serão implementadas no próximo sprint, bem como o cronograma e a divisão de responsabilidades deste ciclo de desenvolvimento.

Assim como na definição do Product Backlog, deve-se utilizar o modelo de Sprint Backlog (apresentado na Figura 4.2), que deve ser possuir seus campos preenchidos no decorrer da reunião.

A primeira parte da reunião se resume em definir o objetivo do Sprint e selecionar quais estórias farão parte deste Sprint Backlog.

O objetivo do Sprint deve ser uma frase de até duas linhas que resuma os propósitos deste ciclo de desenvolvimento, garantindo um maior alinhamento entre o Product Owner e a equipe de desenvolvimento.

A seleção de estórias deve ser feita com base nas prioridades de cada estória e no tamanho desejado para o sprint. Em ordem decrescente de prioridade as estórias vão sendo incluídas no sprint até que a equipe, utilizando o campo “Estimativa” de cada estória, perceba que o tempo total de implementação das estórias alcançou o tempo desejado para o sprint.

A segunda parte da reunião tem como objetivo detalhar cada estória selecionada para o sprint.

A equipe deve passar por cada estória, dividindo-as em atividades menores para facilitar a mensuração do tempo de implementação e a divisão de responsabilidades.

Cada atividade deve possuir apenas um responsável por sua implementação, sendo que este possui liberdade de fazer um cronograma próprio de desenvolvimento das suas atividades.

A divisão em atividades é feita exclusivamente com o propósito de facilitar o desenvolvimento para a equipe e para o Scrum Master. Ela não precisa ser aprovada pelo Product Owner, que deve acompanhar apenas o estado das estórias em si.

A equipe deve se assegurar de tirar todas as dúvidas com o Product Owner de modo a possuir informações em quantidade suficiente para manter o ritmo de implementação até o término do sprint.

A interação do Product Owner com a equipe é a principal função desta reunião, já que estes papéis definem o triângulo de dimensões de cada estória que será implementada no próximo sprint, conforme apresentado na Figura 4.3.

Figura 4.3 - Dimensões da estória (Kinberg, 2007)

Estas dimensões são fortemente dependentes umas das outras, sendo que o escopo e a importância são definidos pelo Product Owner e a estimativa é definida pela equipe.

A definição deste triângulo se inicia com a formação do objetivo do sprint, que acaba afetando o escopo de toda estória selecionada posteriormente.

A dimensão da importância é então utilizada para a seleção das estórias que serão posteriormente detalhadas e dividas pela equipe de desenvolvimento, que por possuir agora um conhecimento mais profundo das atividades pode acabar alterando as estimativas iniciais de cada estória.

Esta alteração de estimativa pode fazer o Product Owner redefinir a importância de determinada estória com a finalidade de deixá-la para um próximo sprint. Outra ação possível por parte dele pode ser a mudança do escopo da estória, deixando mais ou menos complexa, tornando necessária mais uma vez um processo de redefinição da estimativa da estória por parte da equipe.

Antes de se encerrar a reunião devem ser definidos o local e a data da apresentação do sprint e a forma de verificar se uma estória esta completa.

Cada estória deve possuir uma maneira clara de se verificar seu término, tanto por parte da equipe quanto por parte do Product Owner. Como os tipos de estórias são variados (desde design de interfaces até implementação de algoritmos numéricos) é melhor assegurar que cada uma possui um método de verificação de que o combinado foi desenvolvido.

Na reunião de apresentação do sprint o Product Owner verificará se todas as estórias foram desenvolvidas, marcando assim o fim do sprint.

4.5 **Etapa de desenvolvimento**

4.5.1 **Rotina de desenvolvimento**

Com o término da definição do Sprint Backlog a equipe pode iniciar o ciclo de desenvolvimento conforme planejado.

Durante o desenvolvimento o Scrum Master é responsável por garantir que a equipe não tenha de enfrentar obstáculos para manter o ritmo de desenvolvimento. São exemplos de obstáculos: Valores de referências para testes de algoritmos numéricos, informações a serem obtidas com o Product Owner, necessidade de uso de determinada ferramenta não disponível atualmente e etc.

Estas requisições por parte da equipe de desenvolvimento podem ser repassadas para o Scrum Master assim que forem notadas, porém a reunião diária, denominada Scrum diário, reserva parte de seu tempo para tratar deste tipo de questão.

A iteração com o Product Owner durante o desenvolvimento do sprint é fundamental, sendo que cabe ao Scrum Master revelar o estado de progresso do desenvolvimento das tarefas bem como obter informações que possam auxiliar a equipe na implementação das atividades.

A disposição da equipe durante o ciclo de desenvolvimento deve facilitar a comunicação entre os membros. É obrigatório que todos possam comunicar entre si sem que seja necessário utilizar alguma ferramenta como bate-papo virtual ou telefone. A união da equipe permitirá a manutenção do ritmo acelerado de desenvolvimento por parte da equipe, já que pequenas reuniões para discussão sobre a implementação serão feitas de maneira rápida e cada membro poderá requisitar auxílio de outro colega de maneira instantânea.

Logo após o término da reunião de definição do Sprint Backlog o Scrum Master deve reservar algum quadro branco para ser utilizado como monitor do progresso do sprint bem como preparar post-its representando as tarefas derivadas das estórias presentes neste sprint.

Este quadro deve possuir quatro áreas, conforme ilustrado na Figura 4.4:

a) **A fazer**: Esta área abriga as tarefas que fazer parte do sprint mas que ainda não tiveram seu desenvolvimento iniciado por parte do membro da equipe responsável.
b) **Fazendo**: Área que abriga todas as tarefas que foram iniciadas. Cada membro pode possuir mais de uma tarefa que nesta área caso seja necessário desenvolvê-las simultaneamente.
c) **Feito**: Área em que se encontram todas as tarefas que estão completas segundo as definições combinadas nas reuniões anteriores.
d) **Tarefas não-previstas**: Esta área abriga tarefas que são necessárias para que o objetivo do sprint seja alcançado, mas que não foram previstas e incluídas no Sprint Backlog. O registro deste tipo de tarefa auxiliará o debate que será feito na reunião de retrospectiva do Scrum sobre pontos a serem melhorados.
e) **Burndown**: Esta área abriga o gráfico de Burndown que será detalhado no item 4.5.2.

Figura 4.4 - Modelo de quadro

Segundo Cockburn (2002), quadros como este são radiadores de informação na empresa, mostrando informações que podem ser vistas e interpretadas por qualquer pessoa que passa por ele.

O autor cita que todo radiador de informação deve possuir informações que são alteradas conforme o tempo, tornando interessante acompanhar seu progresso por parte de stakeholders do projeto, além de possuir esta informação disponibilizada de forma que seja consumido pouco tempo para interpretação por parte do observador.

Os post-its que representam as tarefas devem seguir o modelo presente na Figura 4.5. O campo “RESPONSÁVEL” abriga o nome do membro da equipe de desenvolvimento que ficou encarregado do desenvolvimento da tarefa em questão.

O campo “ID” abriga um número que não deve se repetir entre tarefas para representá-la. O campo “NOME” abriga o nome que descreve a tarefa e que foi definido no Product Backlog.

Um exemplo de post-it de tarefa pode ser visto na Figura 4.6.

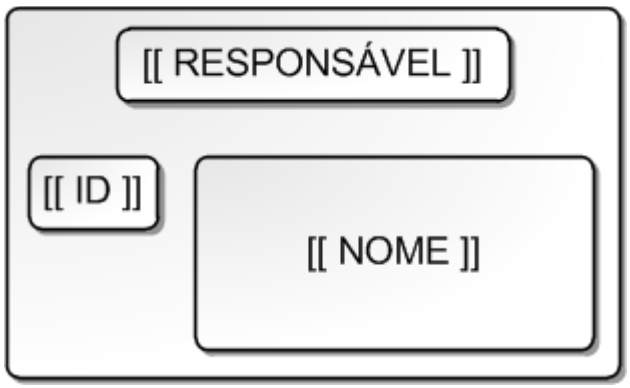

Figura 4.5 - Modelo de post-it de tarefa

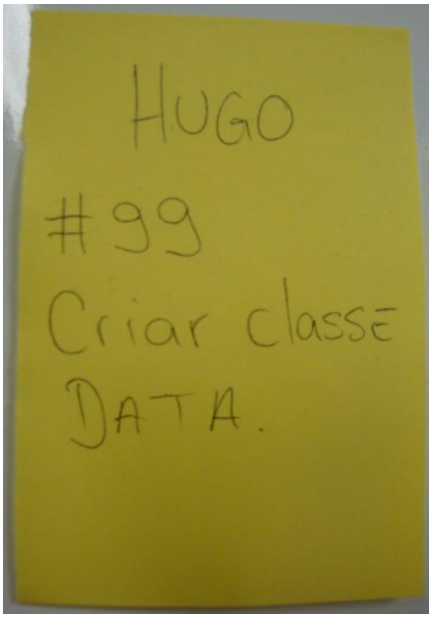

Figura 4.6 - Exemplo de post-it de tarefa

A atualização do estado tarefas no quadro é de responsabilidade dos próprios membros da equipe, sendo que cabe ao Scrum Master validar se uma tarefa realmente foi concluída conforme as definições combinadas nas reuniões anteriores.

4.5.2 **Sprint Burndown**

O Sprint Burndown é um gráfico que apresenta o progresso do desenvolvimento de um sprint.

Neste gráfico o eixo das ordenadas representa o número de tarefas que ainda devem ser finalizadas enquanto que o eixo das abscissas é um eixo temporal que tem como limites as datas de início e término do sprint.

Um exemplo de Sprint Burndown de um sprint hipotético realizado entre 1/1 e 22/1 pode ser visto na Figura 4.7.

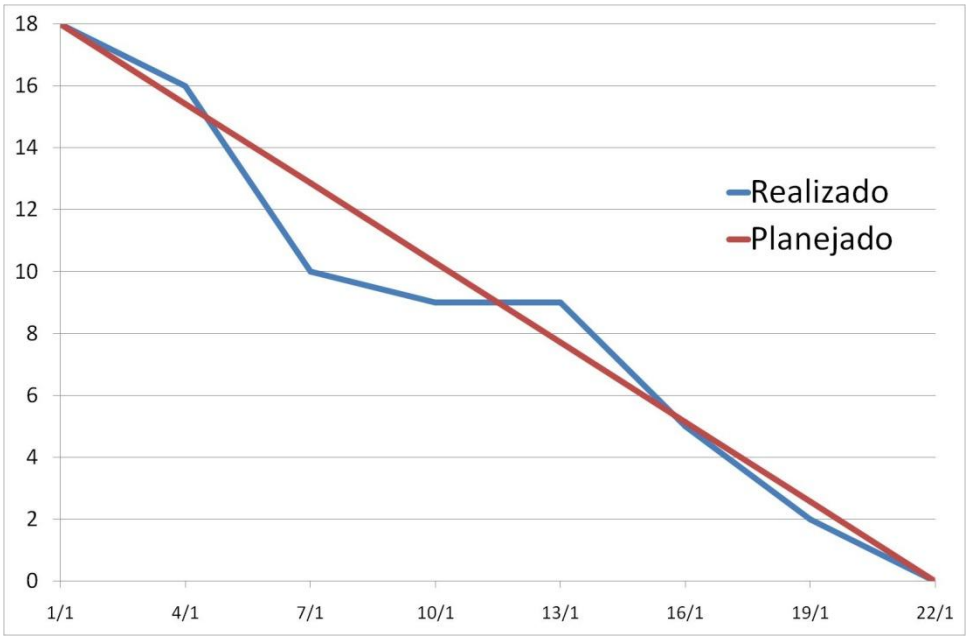

Figura 4.7 - Exemplo de Sprint Burndown

Esta ferramenta pode auxiliar na detecção de falhas na previsão e estimação de tempo das tarefas do sprint e de obstáculos que limitam o ritmo de desenvolvimento da equipe.

Um gráfico de burndown semelhante ao da Figura 4.8 ilustra o caso em que a equipe utilizou estimativas muito pessimistas para as estórias que compõem o Sprint Backlog em questão, o que é naturalmente corrigido nos próximos sprint conforme a equipe ganha mais experiência e habilidade de realizar estimativas.

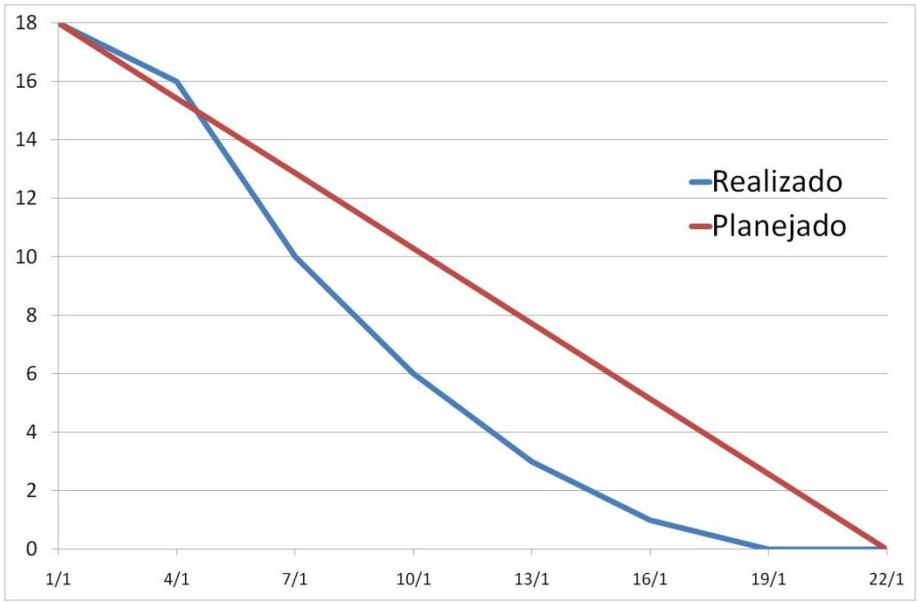

Figura 4.8 - Sprint Burndown com estimativas baixas

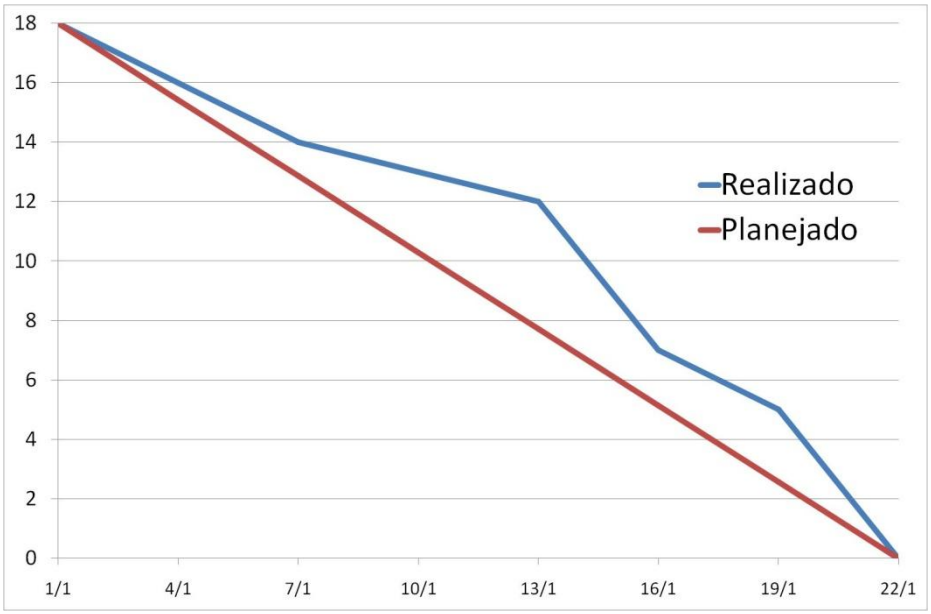

Figura 4.9 - Sprint Burndown com estimativas altas

No caso do Sprint Burndown se assemelhar com o gráfico da Figura 4.9 a equipe pode estar realizando estimativas muito otimistas para as tarefas, o que será corrigido de maneira análoga ao caso de estimativas pessimistas.

Outra possível causa desta anomalia é a ocorrência de eventos que diminuam o ritmo de desenvolvimento da equipe. Tais eventos podem ser projetos paralelos que ocupam tempo de membros da equipe, ausência do Product Owner para esclarecer dúvidas da equipe ou outros eventos específicos. O Scrum Master deve ficar atento para tal questão, sempre com o objetivo de manter o ritmo da equipe o mais próximo possível do ritmo linear presente no gráfico. Também é dele a responsabilidade de atualizar o gráfico com uma freqüência suficiente para acompanhar as alterações dos estados das tarefas da equipe de desenvolvimento.

4.5.3 **Scrum diário**

O Scrum diário é uma reunião que deve ocorrer todos os dias no mesmo local e no mesmo horário.

As presenças do Scrum Master e da equipe de desenvolvimento são obrigatórias, mas a do Product Owner é apenas opcional, sendo que não há espaço para ele falar devido a curta duração deste tipo de reunião.

O objetivo da reunião é atualizar a equipe com relação ao progresso do sprint e tem duração máxima de 15 minutos.

É recomendável que a reunião seja feita com todos os participantes em pé, para assegurar a brevidade da mesma.

Durante a reunião, que deve ter sua ocorrência sempre assegurada pelo Scrum Master, cada membro deve relatar o que fez desde a última reunião, o que planeja fazer hoje e fatos que estejam atrapalhando seu ritmo de desenvolvimento.

Esta reunião aumenta a comunicação entre membros da equipe de desenvolvimento, ilustra problemas que devem ser solucionados pelo Scrum Master e reduz a necessidade de se realizar outras reuniões que poderiam diminuir o ritmo da equipe.

4.6 **Fim do ciclo de desenvolvimento**

4.6.1 **Apresentação do Sprint**

A apresentação do Sprint é uma reunião que é realizada após o término do período de tempo do Sprint e que tem como objetivo mostrar para o Product Owner a implementação das tarefas que constavam no Sprint Backlog.

Kinberg (2007) cita os seguintes motivos que tornam esta reunião necessária:

a) A equipe de desenvolvimento obtém reconhecimento pelo trabalho realizado.
b) Outras pessoas da empresa têm a oportunidade de aprender o que a equipe esta desenvolvimento
c) A demonstração encoraja feedbacks dos stakeholders com relação ao sistema
d) As demonstrações podem ser eventos em que times diferentes podem interagir e discutir seu trabalho.
e) A demonstração faz com que o time realmente finaliza a parte do sistema em questão, ao invés de se acumular várias funcionalidades que não foram terminadas conforme nas metodologias tradicionais.

A reunião deve se iniciar com a apresentação do objetivo do Sprint, e com uma descrição básica do sistema caso alguma pessoas que não conheça o produto esteja presente na reunião.

A apresentação deve possuir um ritmo constante e ser altamente objetiva, ilustrando apenas as funcionalidades do sistema.

As descrições das tarefas realizada no Sprint devem possuir um nível de negócio, sendo que não é recomendável detalhas aspectos técnicos de cada atividade, já que isso pode não ser de interesse do Product Owner.

Ao término da reunião é fundamental que o Product Owner dê uma opinião clara para a equipe de desenvolvimento com relação ao quanto de suas expectativas com relação ao Sprint foram atendidas.

4.6.2 **Revisão das tarefas**

Dado o pequeno tamanho da equipe de desenvolvimento na empresa em questão, é proposta uma reunião ao fim de cada Sprint que não faz parte das metodologias originais que serviram de base para este trabalho.

O objetivo desta reunião é compartilhar todos os aspectos técnicos que não foram citados na apresentação do Sprint por não fazerem parte do escopo daquela reunião.

A reunião deve ser guiada pelo Scrum Master, que deve passar por cada tarefa integrante do Sprint Backlog questionando seu responsável sobre como ela foi implementada.

O resto da equipe deve discutir sobre a maneira escolhida para implementar a tarefa e levantar questões sobre as estratégias de design utilizadas e a padronização do código na empresa.

Ao término da reunião o Scrum Master deve oficializar o término do Sprint com a equipe para que se iniciem as atividades de preparação para o próximo ciclo de desenvolvimento.

4.6.3 **Retrospectiva do Sprint**

A reunião de retrospectiva do Sprint tem como objetivo garantir a melhoria contínua da equipe de desenvolvimento e da empresa com relação a aplicação de metodologias ágeis para a gestão de projetos na empresa.

Nesta reunião, que tem o Scrum Master e a equipe de desenvolvimento como participantes, serão retomadas quais tarefas faziam parte do Sprint Backlog, quais decisões foram tomadas durante o desenvolvimento e quais obstáculos atrapalharam o ritmo da equipe.

Cada membro deve relatar:

a) Quais práticas foram boas para o Sprint e que devem continuar sendo executadas da mesma maneira.
b) Quais práticas poderiam ter sido executadas de maneira diferente de forma a melhorar o desempenho da equipe.
c) Quais questões devem ser levadas em conta por parte de todos no próximo Sprint para que a produtividade e a satisfação dos membros da equipe de desenvolvimento sejam maiores.

Todas as informações e tópicos debatidos devem ser registrados pelo Scrum Master e disponibilizados para o resto da equipe. O Scrum Master também deve ficar atento no próximo Sprint para que a equipe não cometa os mesmos erros e não se esqueça dos possíveis pontos de melhoria que foram levantados.

5 ANÁLISE DOS RESULTADOS INICIAIS

5.1 Considerações iniciais

Nesta parte do trabalho será descrito o início da implementação da proposta apresentada no capítulo anterior na empresa em questão.

Além da apresentação da proposta para os diretores da empresa e para a equipe de desenvolvimento de software, serão apresentados detalhes da utilização da rotina descrita neste trabalho no estágio de planejamento de um projeto interno bem como no primeiro ciclo de desenvolvimento do mesmo.

5.2 Apresentação da proposta

A proposta apresentada neste trabalho não obteve grandes resistências de implantação na empresa em questão.

Foram apresentados os benefícios advindos de metodologias ágeis como o desenvolvimento modular do sistema e a reação rápida com relação a mudanças de escopo, sendo que o primeiro gerou grandes expectativas por parte dos sócios da empresa.

No período de apresentação da proposta estava sendo iniciado o planejamento de um grande sistema contendo funcionalidades exploradas por poucos softwares no mundo.

Dado o caráter experimental do software devido a falta de sistemas semelhantes no mercado, ele seria desenvolvido internamente e apresentado posteriormente para possíveis clientes como prova de conceito, tornando-o uma maneira de prospectar projetos de venda deste sistema com customizações que seriam requisitadas pelos clientes.

Por ser um projeto interno, sem grandes pressões de clientes e prazos, ele foi escolhido para ser desenvolvido de acordo com a metodologia apresentada neste trabalho.

5.3 Início do Projeto

O autor deste trabalho ficou encarregado do papel de Scrum Master do projeto, bem como fez parte da equipe de desenvolvimento juntamente com o membro efetivo da área de software. O diretor de P&D, por conhecer a modelagem matemática e financeira contida no sistema e por ter sido aquele que trouxe a idéia do desenvolvimento deste software para e empresa ficou com o papel de Product Owner.

O Scrum Master criou as páginas específicas sobre este projeto no sistema de gestão que a empresa utiliza e requisitou os materiais que seriam utilizados no decorrer do projeto conforme a proposta presente neste trabalho, como um quadro branco, post-its e etc.

Houve então uma reunião inicial (nos moldes apresentados no item 4.3.1) em que os diretores apresentaram o projeto para o restante da empresa, descrevendo brevemente o mercado em que ele estaria inserido e quais funcionalidades gerais deveriam ser desenvolvidas.

Nos próximos dias a equipe de desenvolvimento, um especialista da área de modelagem e o Scrum Master se encarregaram de definir uma versão inicial do Product Backlog conforme descrito no item 4.3.2.

A presença do Product Owner, mesmo sendo recomendada pela proposta deste trabalho, não foi garantida em todas as reuniões. Este papel recaiu, e deve recair da mesma maneira em projetos futuros, sobre um membro que detêm conhecimentos técnicos vitais para o funcionamento da empresa, que por ser pequena exige que este membro participe em múltiplos projetos. Este fato faz com que o Product Owner possua pouco tempo disponível para as reuniões de início do projeto, que normalmente demandam muito tempo.

Esta questão foi contornada tornando o especialista em modelagem um representante do Product Owner. Ele detinha dos mesmos poderes de decisão do Product Owner e mantinha contatos periódicos com ele para sanar dúvidas da equipe de desenvolvimento.

No decorrer das reuniões foi escolhida uma parte do sistema que possuía questões matemáticas e financeiras conhecidas previamente para ser detalhada e incluída na primeira versão do Product Backlog.

5.4 **Primeiro ciclo de desenvolvimento**

5.4.1 **Início do Sprint**

Após algumas reuniões que geraram a primeira versão do Product Backlog, verificou-se através do campo “Estimativa” do documento que havia trabalho suficiente para um Sprint de três semanas.

De acordo com o item 4.4, foram realizadas então as reuniões para a determinação do Sprint Backlog deste ciclo de desenvolvimento.

Durante as reuniões, realizados conforme descrito no item 4.4.1, cada estória presente no Product Backlog foi detalhada, sub-dividida em tarefas menores, e delegada para algum membro da equipe de desenvolvimento.

Com todas as tarefas do Sprint determinadas, foram marcadas as datas de início e término do mesmo, além de que o Scrum Master preparou o quadro branco reservado para este projeto com as tarefas que haviam sido designadas para este ciclo de desenvolvimento. Uma imagem do quadro branco durante os primeiros dias do Sprint pode ser vista na Figura 5.1.

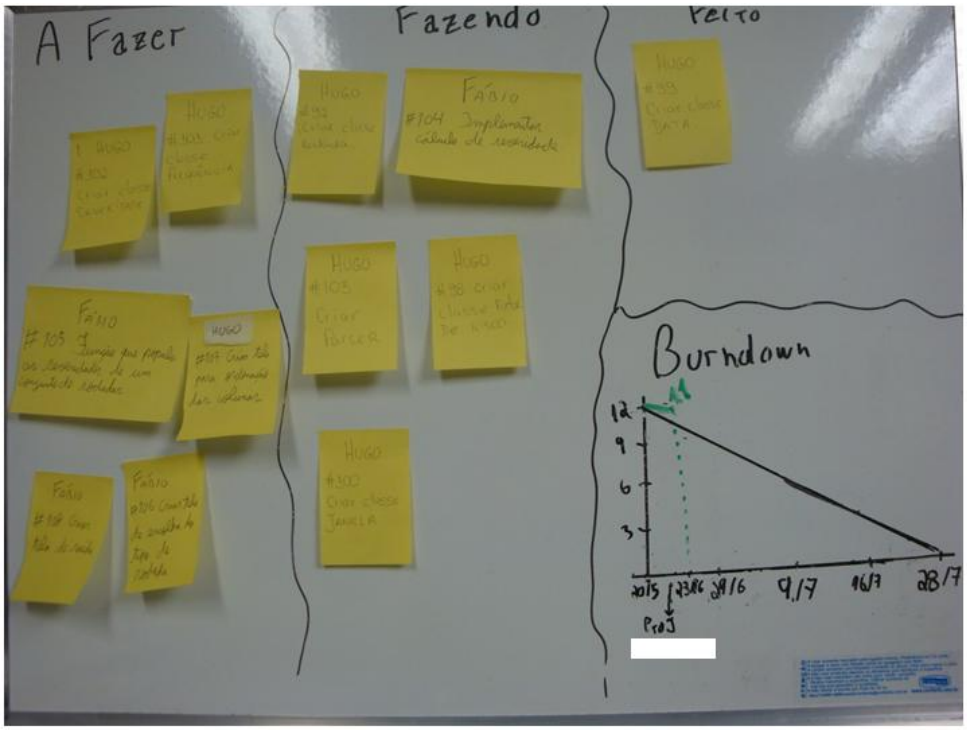

Figura 5.1 - Quadro do Sprint

5.4.2 **Integração com o sistema de gestão da empresa**

Além do controle do progresso das atividades através do quadro branco, foi utilizada exaustivamente por parte do Scrum Master e da equipe de desenvolvimento a ferramenta de gestão de projetos chamada Redmine[1].

A ferramenta permite que seja realizada a gestão dos projetos através do cadastro de tarefas e membros.

Cada membro possui um cadastro de acesso ao sistema e um cargo, que reflete suas permissões no sistema como cadastro ou alteração de tarefas.

As tarefas, chamadas tickets no sistema, possuem campos para determinar seu prazo, seu estado, o membro responsável, sua importância, sua categoria, a versão do sistema com a qual ela se relaciona além de um espaço para registro de horas gastas no desenvolvimento da tarefa.

A categoria de cada ticket foi utilizada como forma de determinar a estória a qual ela faz parte, facilitando assim a criação de filtros para visualizar o estado de todas as tarefas que compõe determinada estória.

De maneira semelhante, o campo versão dos tickets foi utilizado para determinar o Sprint relacionado àquela tarefa

A funcionalidade de versão de software desta ferramenta de gestão permite que sejam acompanhadas todas as tarefas daquele Sprint, bem como sua porcentagem de término e gráficos semelhantes ao Sprint Burndown apresentado no item 4.5.2 gerados de forma automática.

Quando um ticket é criado e delegado para um membro da equipe de desenvolvimento ele recebe um e-mail com a notificação do ocorrido e deve atualizar o estado da tarefa de "Novo" para "Atribuído".

Conforme a tarefa é implementada ele pode atualizar sua porcentagem de término, bem como registrar notas relevantes sobre o desenvolvimento como o número da revisão do sistema de versionamento do projeto ou decisões de design que foram tomadas. Outro elemento que deve ser atualizado freqüentemente pelo responsável da tarefa é o gasto de horas no desenvolvimento da mesma.

[1] Mais detalhes da ferramenta, que é gratuita, podem ser encontrados no site http://www.redmine.org

Quando a tarefa é finalizada pelo membro responsável ele deve atualizar o estado de “Atribuído” para “Resolvido”. Algum membro com permissão de finalização de tarefas, que no caso é o Scrum Master, verificará a tarefa e poderá tanto alterar o estado para “Fechado”, finalizando o ticket, ou para “Atribuído”, indicando que algo mais deve ser implementado pelo membro responsável.

A tela de atualização dos tickets pode ser vista na Figura 5.2.

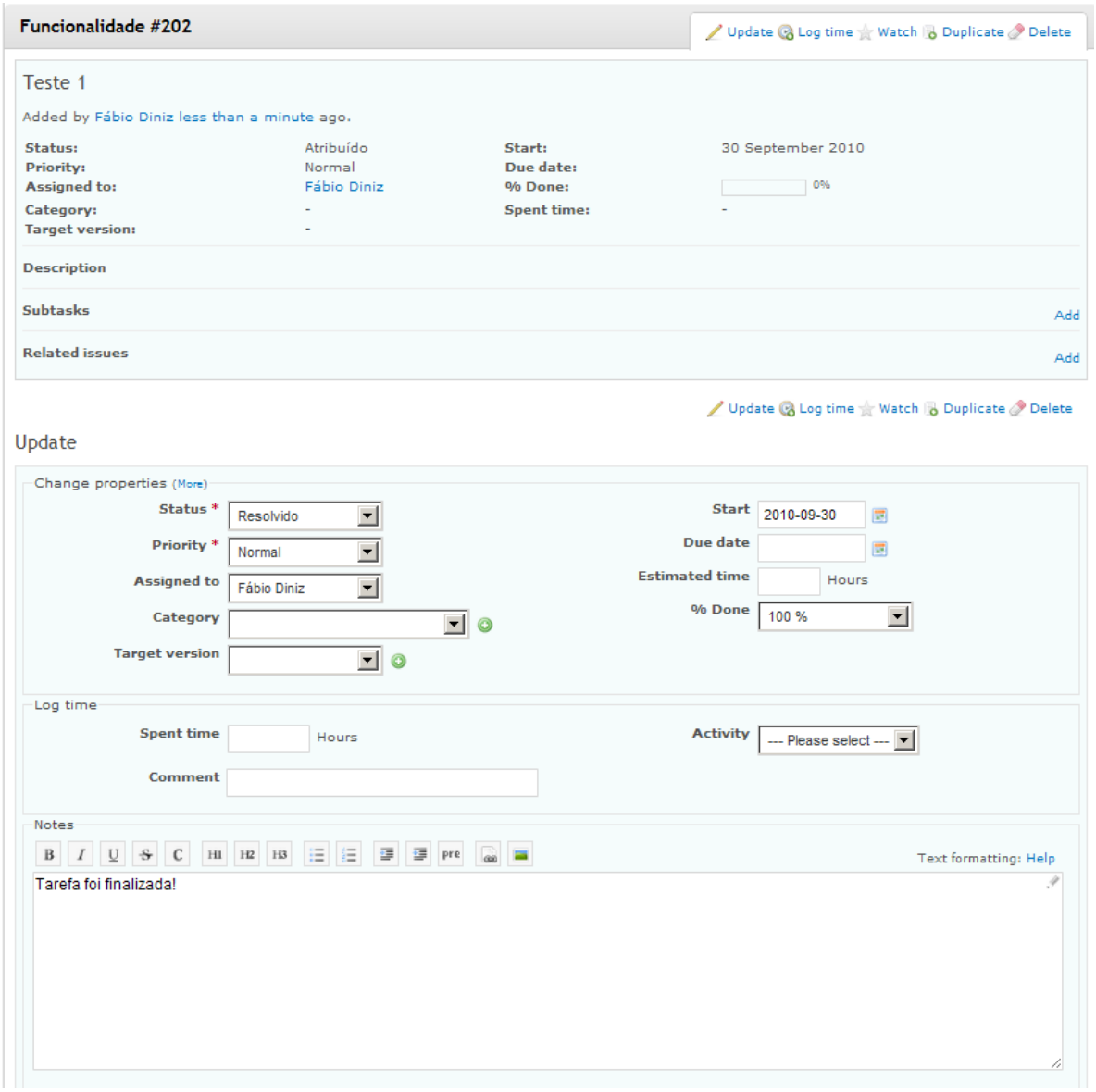

Figura 5.2 - Atualização de ticket

Este sistema de atualização das tarefas permite que esta ferramenta de gestão de projetos se torne uma versão detalhada e com registro permanente do quadro branco do projeto.

Além da atualização das tarefas a ferramenta possui um sistema de páginas Wiki[2] que facilita a documentação da gestão dos projetos da empresa.

Este sistema de páginas Wiki foi utilizado neste projeto para documentar todas as reuniões realizadas, detalhes de cada Sprint como obstáculos e objetivos, e diagramas de design do software. A página contendo links para o registro das reuniões do projeto pode ser vista na Figura 5.3.

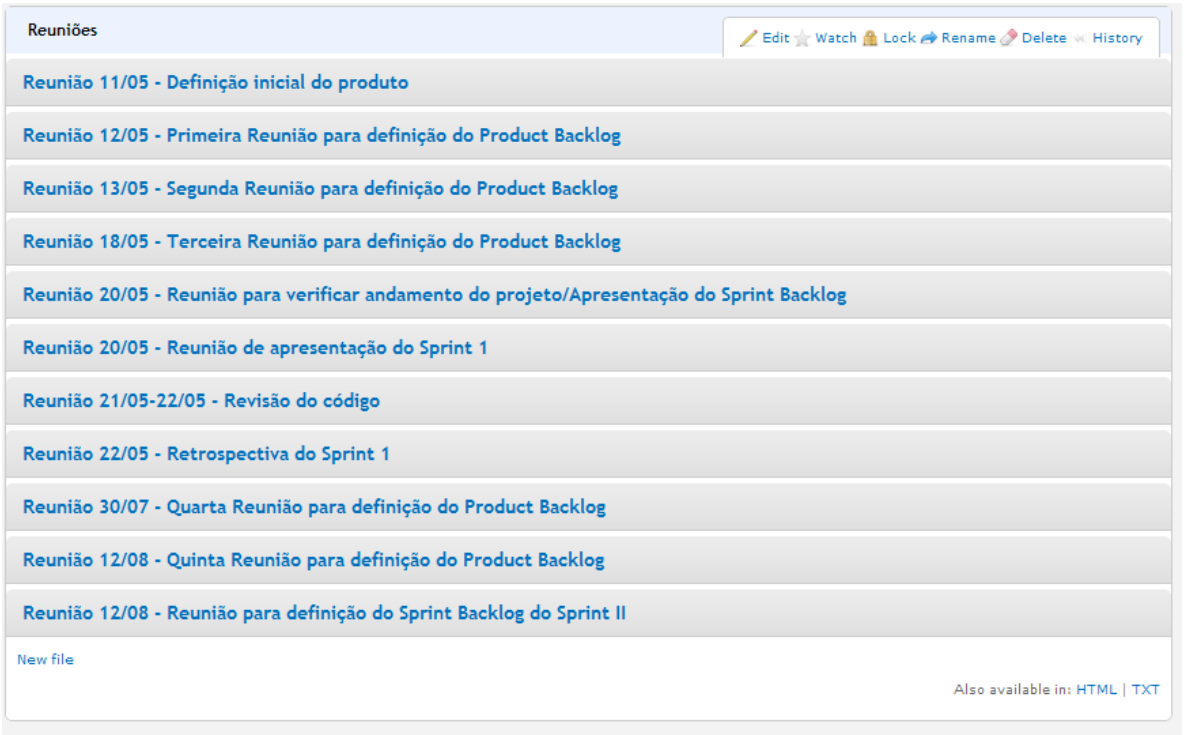

Figura 5.3 - Exemplo de página wiki

5.4.3 **Rotina de desenvolvimento**

Durante o desenvolvimento das tarefas do primeiro Sprint o Scrum Master assegurava a realização do Scrum diário conforme descrito no item 4.5.3.

Este tipo de reunião foi prontamente aceito pelos membros da empresa, já que seus benefícios de evitar reuniões demoradas, facilitar a comunicação entre desenvolvedores e compartilhar o estado das atividades entre todos são simples de serem notados.

[2] Tipo de site Web que permite a criação e a edição de várias páginas Web interligadas de maneira simplificada, como a Wikipedia.

As reuniões eram realizadas após o almoço, um horário em que todos estavam presentes, sempre em frente ao quadro branco, que se encontrava entre as estações de trabalho dos dois membros da equipe de desenvolvimento, com todos os membros de pé, para que a reunião sempre fosse curta e objetiva.

Cada membro era responsável por atualizar o estado de suas atividades tanto no quadro branco como no sistema de gestão Redmine.

O Scrum Master acompanhava estas alterações do estado das tarefas e sempre verificava se uma tarefa indicada como completa realmente deveria possuir este estado através da execução de testes das funcionalidades.

Durante o desenvolvimento das tarefas sempre apareciam dúvidas sobre a arquitetura do sistema e sobre quais aspectos do software deveriam ser priorizados, como velocidade de processamento, robustez, usabilidade, prazo da tarefa e etc. O Scrum Master, mesmo não interferindo na decisão da equipe sobre questões de desenvolvimento, apontava quais valores presentes nas metodologias ágeis poderiam auxiliar na tomada de decisão.

Os princípios mais relevantes para o ambiente em questão foram o segundo e o quarto do manifesto ágil apresentado no anexo 8.1.

O segundo princípio trata da priorização de software funcional, software que pode não conter todas as funcionalidades, mas que consegue executar as que têm, sobre uma documentação completa. Este princípio pode ser utilizado para tratar de questões que envolviam o caráter experimental do sistema que estava sendo desenvolvido, já que sempre se concluía ser mais interessante finalizar determinada funcionalidade para que ela fosse apresentada para o Product Owner do que documentá-la com exatidão. Isto ocorria devido às incertezas naturais presentes no sistema sobre quais funcionalidades ele deveria possuir. Caso este princípio não fosse seguido a maior parte do tempo dedicada à documentação poderia ter sido descartada devido à rápida velocidade de mudança das funcionalidades do sistema.

O quarto princípio do manifesto trata da priorização da capacidade de responder rapidamente às mudanças sobre respeitar o planejamento. Este princípio, assim como ocorrido no princípio anterior, foi de grande importância devido ao caráter experimental do sistema sendo desenvolvido. Durante o ciclo de desenvolvimento a equipe, que no primeiro Sprint estavam cuidando de um módulo do sistema que seria drasticamente estendido em Sprint posteriores, tinha que se preocupar constantemente com a maneira que era feita a arquitetura do sistema para que ele fosse facilmente estendido nos próximos ciclos de desenvolvimento. Isto exigia que tarefas não previstas no Sprint Backlog, como reuniões de design, fossem

feitas para sempre assegurar que o sistema estava pronto para aceitar mudanças posteriores sem que ocorresse retrabalho por parte dos membros da equipe de desenvolvimento.

O manifesto ágil foi uma ferramenta crucial para a transformação da cultura da empresa que possuía raízes de desenvolvimento em cascata, apresentado no item 2.3.1, mas que por nunca ter aplicado-o de maneira formal estava pronta para mudanças que eliminassem os problemas descritos no estudo do contexto da empresa.

Durante o desenvolvimento do Sprint apareceram pequenos projetos que possuíam um cronograma curto e que exigiam a participação dos membros da equipe de desenvolvimento para serem entregues no prazo estipulado.

Dada a natureza experimental do projeto, e o fato de que ele possuía um foco voltado para a pesquisa e o desenvolvimento de produtos internos, ele possuía uma prioridade baixa frente a projetos voltados para clientes da empresa.

Devido a estas circunstancias, na ocorrência do surgimento destes projetos eram feitas pausas no Sprint até que eles, que tinham duração de duas a quatro semanas, fossem completados.

Durante as pausas o Sprint Burndown deixava de ser atualizado e a data final do Sprint era recalculada com base nos dias em que o projeto ficou parado.

A descrição do motivo das pausas era inclusa no gráfico de Burndown conforme ilustra a foto apresentada na Figura 5.4.

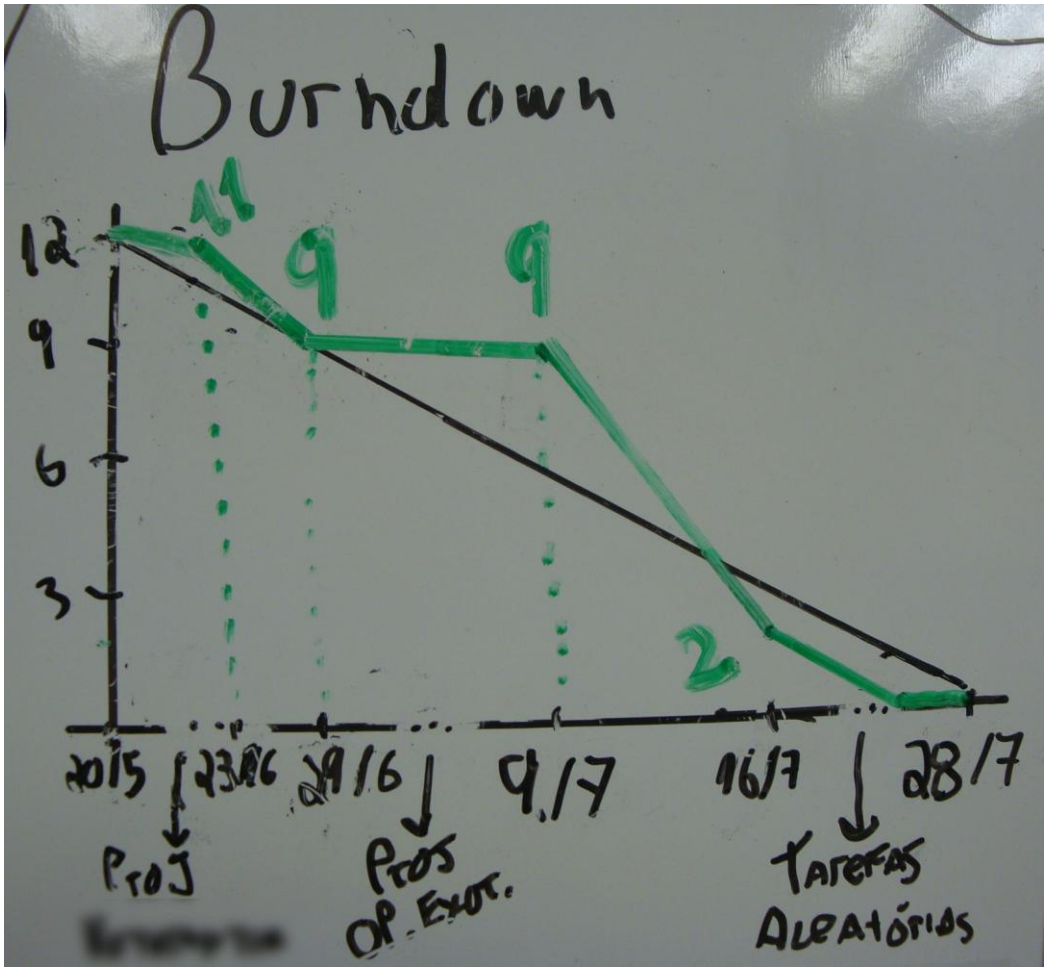

Figura 5.4 - Burndown de Sprint com pausas

A primeira pausa realizada gerou cerca de dois dias de trabalho a mais por parte da equipe de desenvolvimento devido à quebra do ritmo de produção e o esquecimento de decisões da arquitetura do sistema.

Após o término desta pausa foi dado mais ênfase à geração de documentação feita por parte do Scrum Master de todo o processo de desenvolvimento, como reuniões, decisões de design e diagramas, fazendo com que as pausas não acrescentassem tempo extra de desenvolvimento.

5.4.4 **Fim do Primeiro Sprint**

O primeiro Sprint teve 39 dias como período total de desenvolvimento levando em conta as pausas realizadas devido à ocorrência de outros projetos com maior prioridade, feriados e fins de semana.

Alguns dias após o término das atividades foi realizada a reunião de apresentação do Sprint conforme descrito no item 4.6.1. Na reunião o Product Owner pode comprovar que todo o trabalho proposto no Sprint Backlog havia sido realizado, marcando assim o encerramento do Sprint.

Antes de se iniciar o planejamento de um novo Sprint, foram feitas as reuniões descritas nos itens 4.6.2 e 4.6.3.

A reunião de revisão das tarefas apresentou benefícios que comprovaram sua relevância na metodologia. Ao passar por todo aspecto técnico das tarefas do Sprint que havia sido realizado outros membros podiam conhecer as técnicas que foram utilizadas e o conhecimento era compartilhado, algo que foi simples devido ao tamanho reduzido da equipe de desenvolvimento.

Percebeu-se que as reuniões diárias apenas atualizavam o resto da equipe com relação ao estado das tarefas, mas só esta reunião de revisão das tarefas que compartilhava o conhecimento técnico que havia sido conquistado decorrente da experiência daquele Sprint.

A outra reunião que foi realizada foi a de retrospectiva do Sprint. Nesta reunião puderam ser discutidos todos os aspectos bons e ruins que ocorreram neste ciclo de desenvolvimento. Durante a reunião foram colhidas opiniões sobre quais práticas tiveram sucesso e deveriam continuar a serem executadas e quais práticas falharam e devem ser evitadas no próximo Sprint.

No término desta reunião foi percebido que não houve espaço para idéias novas de como melhorar a produtividade da equipe no próximo ciclo de desenvolvimento, e sim apenas para opinar sobre práticas que já estavam sendo executadas. Por este motivo esta terceira opinião que deve ser coletadas dos membros nesta reunião foi colocada na proposta de rotina de desenvolvimento apresentada no item 4.6.3.

5.5 **Primeiros resultados**

Dado o fato de que este é um projeto interno ainda em início de desenvolvimento, não é possível analisar por hora os resultados da aplicação da proposta deste trabalho nos custos dos projetos.

Foi possível verificar melhoras na gestão do escopo, já que havia uma definição clara do escopo do sistema e de cada Sprint por todos os membros do projeto durante esta primeira fase de desenvolvimento, algo que não ocorria em projetos passados.

A priorização do planejamento foi outra melhora observada, sendo que até mesmo os diretores dedicaram tempo exclusivamente para a elaboração do Product Backlog e para as reuniões de definição do Sprint Backlog.

6 CONCLUSÃO

A utilização de técnicas de gestão de projetos em empresas pequenas pode parecer desnecessária em casos em que as equipes são pequenas (como na empresa em questão que possui equipes de no máximo cinco elementos), os projetos possuem um escopo reduzido ou quando não há a ocorrência da execução de projetos de maneira simultânea.

Um profissional com conhecimentos de técnicas de gerenciamento diversas (como os modelos tradicionais do PMBoK ou os modelos de gerenciamento ágil) pode verificar, no entanto, muitos problemas clássicos que podem passar desapercebidos por profissionais de outras áreas.

No caso da empresa alvo deste trabalho os problemas clássicos que foram encontrados e abordados é a fraca gestão de escopo e a falta de planejamento das etapas iniciais do projeto.

Uma gestão de escopo falha em projetos de desenvolvimento de sistemas é um problema que foi um dos pontos chave para a adoção de técnicas de metodologias ágeis na empresa.

Conforme indicado no item 5, que apresentou os resultados iniciais deste trabalho na empresa, tais metodologias podem ser implementadas de maneira rápida em equipes pequenas e possuem resultados que podem ser observados logo nos primeiros ciclos de desenvolvimento.

A gestão de escopo se tornou um dos itens mais críticos nos projetos internos que estão servindo como laboratórios para aplicação da rotina proposta neste trabalho. Nestes projetos há uma preocupação por parte de toda a equipe de que sempre exista um escopo objetivo e claro do projeto e de cada ciclo de desenvolvimento, com a ajuda de uma série de ferramentas de controle físicas e virtuais como mencionado anteriormente. Há uma preocupação extra com relação ao quão válido ainda é o escopo de determinado projeto, sendo que a metodologia propõe que este seja sempre revisto pelo Product Owner para que os próximos ciclos de desenvolvimento possam corrigir quais alterações do plano inicial.

7 REFERÊNCIA BIBLIOGRÁFICA

____. *A Guide to the Project Management Body of Knowledge* (PMBoK). 3. Ed. Project Management Institute, 2004.

ABRAHAMSSON, P.; Salo, O.; RONKAINEN, J.; WARSTA, J. Agile Software Development Methods: Review and Analysis. Espoo: VTT Publications 478, 2002.

Agile Manifesto, http://agilemanifesto.org/, acessado em 22 de Junho de 2010.

BECK, Kent. Programação extrema (XP) explicada: acolha as mudanças. Porto Alegre: Bookman, 2004.

BOEHM, B. W.: A Spiral Model of Software Development, The Institute of Electrical and Electronics Engineers Inc. (IEEE), 1988.

CARAYANNIS E.; KWAK, Y.; ANBARI, F. The Story of Managing Projects: an interdisciplinary approach. Quorum Books, 2003.

CARVALHO, M.M; RABCHINI, R. Construindo competências para gerenciar projetos – teoria e casos. São Paulo: Ed. Atlas, 2006

COCKBURN, A. Agile software development. Boston: Addison-Wesley, 2002

COHEN, D.; LINDVALL, M.; COSTA, P. An introduction to agile methods. In Advances in Computers. New York: Elsevier Science, 2004. P. 1-66

HUNTER, M.; STICKNEY, F. Overview of project management applications. In CLEVELAND, D.I. e KING, W.R. Project management handbook. New York: Van Nostrand Reinhold, 1983. P. 644-668.

IEEE Std 610.12. IEEE Standard Glossary of Software Engineering Terminology, 1990.

ISO – INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 10006: Quality management – Guidelines to quality in Project management. 1997.

KEIL, M. e CARMEL, E. Customer-Developer Links in Software Development. Communications of the ACM, Nova Iorque, v. 38, n. 5, p. 33-44, maio 1995

KERZNER, H. Gestão de Projetos: As Melhores Práticas. Porto Alegre: Bookman, 2002.

KINBERG, H. Scrum and XP from the Trenches, C4Media, Stockholm, 2007

TURRIONI, J.; MELLO, C. Pesquisa-ação na Engenharia de Produção. In MIGUEL, P. (Org) Metodologia de Pesquisa em Engenharia de Produção e Gestão de Operações. São Paulo: Elsevier, 2010. P. 145-163.

PRESSMAN, R. Software Engineering – A Practitioner's Approach. Columbus: McGraw-Hill, 2003

RISING, L. e JANOFF, N. The Scrum Software Development Process for Small Teams. IEEE Software, v. 17, n. 4, p. 11–13, 2000.

ROYCE, W. Managing The Development Of Large Software Systems. In: Proceedings Of IEEE. Wescon: 1970. P. 1–9.

SCHWABER, K. Agile Project Management with Scrum. Redmond: Microsoft Press, 2004

SUTHERLAND, J.; SCHWABER, K. The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. 2007.

8 ANEXOS

8.1 Manifesto Ágil

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Kent Beck Mike Beedle Arie van Bennekum
Alistair Cockburn Ward Cunningham Martin Fowler
James Grenning Jim Highsmith Andrew Hunt
Ron Jeffries Jon Kern Brian Marick
Robert C. Martin Steve Mellor Ken Schwaber
Jeff Sutherland Dave Thomas