UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

GESTÃO PARA O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO, UTILIZANDO UMA ABORDAGEM ÁGIL E ADAPTATIVA NA MICROEMPRESA

DISSERTAÇÃO DE MESTRADO

Jean Carlo Albiero Berni

Santa Maria, RS, Brasil
2010

GESTÃO PARA O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO, UTILIZANDO UMA ABORDAGEM ÁGIL E ADAPTATIVA NA MICROEMPRESA

por

Jean Carlo Albiero Berni

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Engenharia de Produção, Área de Concentração em Gerência da Produção, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Engenharia de Produção

Orientador: Marcos Cordeiro d'Ornellas

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia de Produção

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

GESTÃO PARA O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO, UTILIZANDO UMA ABORDAGEM ÁGIL E ADAPTATIVA NA MICROEMPRESA

elaborada por
Jean Carlo Albiero Berni

como requisito parcial para obtenção do grau de
Mestre em Engenharia de Produção

COMISSÃO EXAMINADORA:

_____
Marcos Cordeiro d'Ornellas, Dr.
(Presidente/Orientador)

_____
Lisandra Manzoni Fontoura, Dra. (UFSM)

_____
Elpidio Oscar Benitez Nara, Dr. (UNISC)

Santa Maria, 22 de março de 2010.

AGRADECIMENTOS

Agradeço à Universidade Federal de Santa Maria, e aos professores e funcionários do Programa de Pós-Graduação em Engenharia de Produção.

Ao Prof. Marcos Cordeiro d'Ornellas, pela orientação desse trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES, pelo apoio financeiro.

Aos professores membros da Banca Examinadora, pela revisão e sugestões.

Aos meus colegas de trabalho e sócios, pelo incentivo.

Aos meus pais Selito e Luci, por me permitirem chegar até aqui.

Ao meu irmão Cristiano, pelo companheirismo ao longo desses anos.

À minha noiva Clarissa, pela apoio incondicional.

Determinação coragem e auto confiança são fatores decisivos para o sucesso. Se estamos possuídos por uma inabalável determinação conseguiremos superá-los. Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho.
(Dalai Lama)

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Engenharia de Produção
Universidade Federal de Santa Maria

GESTÃO PARA O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO, UTILIZANDO UMA ABORDAGEM ÁGIL E ADAPTATIVA NA MICROEMPRESA

AUTOR: JEAN CARLO ALBIERO BERNI
ORIENTADOR: MARCOS CORDEIRO D'ORNELLAS, DR.
Data e Local da Defesa: Santa Maria, 22 de março de 2010.

O desenvolvimento de software científico possui particularidades que, frequentemente, diferem das regras comumente utilizadas na produção de softwares comerciais. Novos requisitos que surgem no decorrer do desenvolvimento, necessidades de comunicação constante com os *stakeholders* (pesquisadores) e adaptação nos processos fazem da produção desse tipo software uma atividade mais interativa do que o convencional. Metodologias ágeis para desenvolvimento de software surgem como alternativas aos métodos planejados ou prescritivos e, como analisado, atendem às carências decorrentes da elaboração de softwares em ambientes dinâmicos. Neste trabalho, é proposto um método de gestão para o processo de desenvolvimento de software científico. Na definição do método, combinaram-se as metodologias ágeis *Scrum* e *Extreme Programming* levando-se em consideração as características do processo científico. O método foi aplicado em projetos reais de uma microempresa de base tecnológica para sua validação e, através de uma análise qualitativa, pode-se verificar a melhoria dos processos de desenvolvimento da empresa.

Palavras-chave: Metodologias ágeis, software científico, processos de software, Engenharia de Software, microempresa.

ABSTRACT

Master's Dissertation
Program of Post-Graduation in Production Engineering
Federal University of Santa Maria

MANAGEMENT FOR THE SCIENTIFIC SOFTWARE DEVELOPMENT PROCESS, USING A AGILE AND ADAPTIVE APPROACH IN SMALL BUSINESS

AUTHOR: JEAN CARLO ALBIERO BERNI
TUTOR: MARCOS CORDEIRO D'ORNELLAS, DR.
Date and Place of Defense: Santa Maria, March 22st, 2010.

The scientific software development has particularities that, frequently, differ from usual rules in commercial software production. The emerging requirements that appear while the development process is running, needs for often communication between stakeholders (researchers) and developers, and necessary ways to customize the process, make this kind of software building activity more iterative than conventional. Agile methodologies for software development arise as alternatives for prescriptive methods and, as analyzed in this research, have the answer for how to build software in dynamics environments. In this work is proposed a management method for the scientific software development process. The agile methodologies Scrum and Extreme Programming were tailored to reach the peculiarity of scientific process, and the method definition. The model was applied in real projects in a small business company for validation and, through a qualitative analysis, the improvement of the development process could be checked.

Keywords: Agile methodologies, scientific software, software process, Software Engineering, small business.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CMMI – Capability Maturity Model Integration

ERP – Enterprise Resource Planning

LaCA – Laboratório de Computação Aplicada

ITSM – Incubadora Tecnológica de Santa Maria

ROI – Retorno Sobre Investimento

RUP – Rational Unified Process

SEBRAE – Serviço Brasileiro de Apoio às Micro e Pequenas Empresas

SPI – Software Process Improvement

TDD – Test Driven Development

XP – Extreme Programming

SUMÁRIO

1. INTRODUÇÃO

Devido às pressões competitivas, cada vez mais presentes na indústria de software, a relação cliente-empresa tem influenciado consideravelmente a forma como os serviços são providos. Empresas de software são obrigadas a fornecer soluções de forma inovadora levando em consideração fatores determinantes para o sucesso, como por exemplo, a otimização do *time-to-market*, que é o tempo que um produto leva desde sua concepção até estar pronto para venda. Pesquisas recentes têm identificado questões relacionadas ao desenvolvimento de software que definem o modo como processo produtivo de software é organizado (O'CONNOR & COLEMAN, 2007; KIRK, 2004; ANGKASAPUTRA & PFAHL, 2004).

Dessa maneira, empresas de software, especialmente *startups*, precisam ser flexíveis, criativas, dinâmicas e capazes de entregar produtos rapidamente por motivos de sobrevivência (O'CONNOR & COLEMAN, 2007). Assim, centralizam os esforços no produto e temem que o melhoramento de processos acarrete a perda de foco, aumento dos custos e perda de flexibilidade. As empresas que se propõem ao desenvolvimento de software científico, encontram um ambiente extremamente mutável, com novas tecnologias surgindo a todo momento. Além dos aspectos de inovação tecnológicas atrelados, existem questões de pesquisa e incertezas que, muitas vezes, não podem ser previstas no planejamento de execução de projetos dessa natureza.

Os produtos inovadores estão inseridos em ambientes de negócios dinâmicos caracterizados pela dificuldade em prever o futuro, incertezas e grandes desafios. Nesse contexto, o modelo tradicional de gestão de projetos tem sido questionado quanto a sua eficácia, e novas competências vem sendo desenvolvidas (SUIKKI et al., 2006 apud CONFORTO & AMARAL, 2007). Assim, surgem novos paradigmas como as metodologias ágeis, que propõem uma forma alternativa de estruturar o desenvolvimento de projetos com as características supracitadas. Esses métodos têm por objetivo orientar o processo de produção para responder de forma mais eficiente à dinamicidade do ambiente, que é caracterizado pelo desenvolvimento de software com geração de valor e inovação.

A forma como o ambiente influencia o desenvolvimento de software é razão pela qual essas empresas adotam metodologias ágeis ou decidem por metodologias tradicionais (ZIEMER, 2007). As variáveis impostas pelo ambiente competitivo irão definir as estratégias que devem ser tomadas pela organização de modo que possa honrar seus acordos comerciais, cumprindo cronogramas e desenvolvendo produtos com a qualidade desejável. Em micro e pequenas empresas, observa-se que adoção de mecanismos para a padronização do processo produtivo acabam burocratizando a empresa em excesso, restringindo sua flexibilidade. Segundo O'Connor (2007), quanto menor a empresa, maior a aversão à documentação e burocracia.

Como consequência, essas empresas acabam utilizando processos de desenvolvimento de software adaptados, baseados em metodologias padrões da indústria como XP (*Extreme Programming*), RUP (*Rational Unified Process*), *Scrum*, dentre outros. Em outras palavras, a organização traça seu próprio método de gestão para o processo de desenvolvimento de software - um subprocesso - fazendo com que sejam definidas somente atividades ou tarefas que permitam a fluência do processo produtivo. A adaptação ou *Tailoring* de processos de software vem sendo estudado ao longo das últimas duas décadas, existe um grande número de autores que tratam dessa abordagem (BASILI & ROMBACH, 1999; CURTIS et al., 1992; DRAPPA & LUDEWIG, 1999; RAFFO & HARRISON, 2000; LAKEY, 2003; STORRLE, 2003).

Microempresas e/ou *startups* têm necessidades diferentes de grandes organizações produtoras de software. O excesso de burocracia ou demasiada especificação de processos pode reduzir a capacidade da empresa produzir com agilidade. Conforme mencionado, a perda de foco no desenvolvimento e aperfeiçoamento do produto pode representar o fracasso no andamento de qualquer projeto. A utilização de uma metodologia ou processo de produção adaptado às realidades enfrentadas pela pequena organização é de fundamental importância para sua sobrevivência e crescimento (BERNI et al., 2009).

Dentro do universo das empresas nascentes ou de pequeno porte, neste trabalho, restringe-se o escopo de análise sobre as organizações que se propõem ao desenvolvimento de softwares científicos e que possuem forte interação com universidades. A produção de software científico apresenta características que os diferenciam do desenvolvimento de software comerciais comuns, como são aqueles destinados a aplicações de gestão (ERPs), controle e planejamento de empresas e demais organizações.

Para Segal & Morris (2008) o processo de criação de softwares científicos é fundamentalmente diferente do desenvolvimento de softwares convencionais. Fatores de

incerteza elevam consideravelmente os riscos por trás de projetos desta natureza, devendo-se assumir que mudanças são inevitáveis e que todos os aspectos do projeto, em seu início, não são conhecidos. Assim, se faz necessária a aplicação de uma abordagem diferenciada capaz de minimizar e controlar possíveis modificações no cenário estabelecido para o processo de desenvolvimento do software.

Nesse cenário, percebe-se que as atividades de gestão devem garantir agilidade no processo produtivo, diminuir o tempo de produção (*time-to-market*), reduzir custos e otimizar a utilização de recursos disponíveis. Fundamentando-se nessa asserção surge o questionamento: É possível criar um método de gestão para o processo de desenvolvimento de software baseado em metodologias ágeis que seja adaptável à realidade de microempresas que desenvolvem software científico?

O objetivo geral desse trabalho é desenvolver um método de gestão para o processo de desenvolvimento de software científico, utilizando metodologias ágeis como base. Esse método será estabelecido fazendo-se uma adaptação das melhores práticas de gestão e de desenvolvimento de software, descritas pelas metodologias ágeis *Scrum* e XP, que podem ser aplicadas às particularidades do desenvolvimento do software científico. O método de gestão proposto será implantado e avaliado no ambiente real de uma microempresa.

Os objetivos específicos deste projeto são:

a) Identificar as características do desenvolvimento de software científico na microempresa;

b) Analisar as metodologias ágeis com a finalidade de encontrar as melhores práticas que possam ser empregadas dentro de uma microempresa que desenvolve software científico;

c) Implantar e avaliar o método de gestão para o processo desenvolvimento de software científico proposto em projetos reais de uma microempresa.

Para o desenvolvimento deste trabalho utilizar-se-á uma pesquisa de natureza exploratório-explicativa. A pesquisa exploratória tem como objetivo o aprimoramento de ideias ou a descoberta de intuições e, através de um delineamento bibliográfico, efetuar o levantamento das informações pertinentes ao problema para averiguação do que já foi analisado em relação ao tema (SELLTIZ et al., 1975). A pesquisa explicativa, que têm como preocupação identificar os fatores que determinam ou que contribuem para a ocorrência dos

fenômenos, será utilizada para avaliar as hipóteses verificadas por meio de um delineamento experimental (GIL, 1988). Desse modo, a presente pesquisa visa a identificação dos fatores que influenciam o ambiente estudado, a definição de um método de gestão que será aplicado em um ambiente real e, posteriormente, a validação desse método através de uma análise qualitativa.

Para a pesquisa experimental foram levantadas as seguintes hipóteses:

a) HIPÓTESE 1: A adaptação (*Tailoring*) de metodologias ágeis para utilização em microempresas é plausível;

b) HIPÓTESE 2: É possível utilizar-se de metodologias ágeis adaptadas para o desenvolvimento de softwares científicos.

Para a realização dessa pesquisa organizou-se a estrutura do trabalho da seguinte forma: no Capítulo 2 são descritas as metodologias ágeis, seus fundamentos e valores. As metodologias *Scrum* e *Extreme Programming* são analisadas de acordo com o objetivo do trabalho; no Capítulo 3 abordam-se as características do desenvolvimento de software direcionado a resolução de problemas científicos; o Capítulo 4 apresenta o método de gestão para o processo de desenvolvimento de software científico proposto juntamente com as delimitações do escopo da pesquisa; no Capítulo 5 são discutidos os resultados obtidos através da análise de três estudos de caso, e; no Capítulo 6 expõe-se a conclusão do trabalho.

2. METODOLOGIAS ÁGEIS

Pequenas empresas de software não estão adotando metodologias prescritivas justamente pelo atrelamento à burocracia pertinente aos modelos que são propostos. Em um ambiente em constante modificação, onde velocidade de resposta e entrega de produtos é o diferencial competitivo, um modelo de processo adaptável e flexível é o intento das organizações.

O estilo antecipatório ou prescritivo para o desenvolvimento de software, representado pelas metodologias planejadas ou tradicionais como o modelo de desenvolvimento em cascata, desenvolvimento evolucionário, RUP (RATIONAL, 2001), dentre outras, pregam a definição dos requisitos de software nas etapas iniciais do processo de desenvolvimento. Ou seja, devem-se prever com antecedência todas as prováveis iterações que irão ocorrer ao longo da execução do projeto. Essas técnicas são muito importantes para definição de custos, riscos e demais contingências que possam influenciar o andamento do projeto, mas raramente são capazes de prever com precisão todas as questões relacionadas ao processo de software.

Ao se tentar prever, no início do projeto, todos os fatores que influenciarão as atividades de desenvolvimento e, a partir disso, planejar sua execução, acaba engessando sua evolução. Conforme Brooks (1987) especificar totalmente um software antes do início de sua implementação é impossível. Com base nestas justificativas, surgem novas metodologias para o desenvolvimento de software que pregam que a elaboração de software é um processo criativo e experimental, e não deve estar associado de forma rígida a cronogramas, prazos e escopo de desenvolvimento.

Em 2001, especialistas em processos de desenvolvimento de software, representando os métodos *Extreme Programing* - XP (BECK, 1999), *Scrum* (SCHWABER,1985), dentre outros, publicaram o Manifesto Ágil (BECK et al, 2001), que estabelecia um conjunto de valores e princípios que deveriam ser aplicadas aos processos de produção de software. Métodos ágeis são processos que suportam a filosofia ágil. Estes métodos consistem de elementos individuais ou práticas, dentre as quais pode-se citar definição de requisitos/histórias, programação em pares, reuniões diárias, demostrações semanais às partes

interessadas e desenvolvimento em ciclos. A filosofia ágil prega a entrega constante de valor para o cliente, onde o foco principal deve ser a identificação e desenvolvimento das funcionalidades mais importantes para o negócio do cliente.

Para Shore e Warden (2008), o desenvolvimento de software baseado em metodologias ágeis tem-se tornado cada vez mais popular e vem sendo adotado de forma gradual mesmo por grandes empresas, historicamente mais burocratizadas, como Yahoo, Microsoft, Google e Symantec. Afirmam, também que os métodos ágeis auxiliam no sucesso organizacional concentrando-se em agregar valor e em diminuir custos. Isto se traduz diretamente em um aumentado retorno sobre os investimentos (ROI).

O Standish Group (1994) apud Shore e Warden (2008) fornece algumas definições clássicas de sucesso organizacional para o desenvolvimento de software:

a) Bem sucedido: "completo a tempo, dentro do orçamento, com todos os recursos e funções como especificados originalmente";

b) Desafiado: "completo e operando, mas fora do orçamento, ou fora do prazo estimado e com menos recursos e funções do que especificado originalmente."

c) Defeituoso: "Cancelado em algum ponto durante o ciclo de desenvolvimento"

"Apesar de sua popularidade, existe algo errado nessas definições. Um projeto pode ser bem sucedido mesmo que nunca gere um centavo e pode ser desafiado, mesmo que gere milhões de lucro" (SHORE & WARDEN, 2008). Nota-se que o propósito final do software não é observado: gerar valor agregado para o cliente ou para a empresa que o desenvolveu. O foco reside na implementação das funcionalidades previstas em contrato e não levam em consideração a dinamicidade desse tipo de ambiente.

As metodologias ágeis vem ganhando popularidade junto as empresas de desenvolvimento e seus clientes justamente por atingirem de imediato esses pontos. A geração de valor para o cliente o mais rápido possível, com entregas de software incrementais que visam atender os objetivos do negócio.

2.1. O manifesto ágil

Conforme mencionado, em 2001 um grupo de entusiastas das metodologias ágeis lançaram o Manifesto Ágil, onde foram publicados um conjunto de valores e princípios que regem o pensamento ágil para o desenvolvimento de software (BECK et al, 2001). Para estes, o ágil é mais do que processos leves, é uma maneira de pensar sobre desenvolvimento do software. A descrição canônica desta maneira de pensar é o Manifesto Ágil, um conjunto de quatro valores e doze princípios, descrito a seguir:

Os quatro valores:

a) Indivíduos e interações são mais importante que processos e ferramentas;

b) Software funcionando é mais importante que uma documentação extensa;

c) A colaboração do cliente é mais importante que a negociação do contrato;

d) Responder às mudanças é mais importante que seguir o planejamento;

Os doze princípios:

a) Nossa maior prioridade é satisfazer o cliente por meio de entregas antecipadas e contínuas de software com alto valor.

b) Solicitações de mudanças são bem-vindas mesmo quando o desenvolvimento está adiantado. As rédeas dos processos ágeis mudam para a vantagem competitiva do cliente.

c) Entregar software funcionando com frequência, de alguma semanas a alguns meses, com preferência a uma escala de tempo mais curta.

d) Pessoas de negócios e os desenvolvedores devem trabalhar juntos diariamente ao longo do projeto.

e) Construir projetos com pessoas motivadas. Fornecer-lhes o ambiente e apoio necessários e confiar que farão o trabalho.

f) O método mais eficiente e efetivo de transmitir informações para e dentro de uma equipe de desenvolvimento é a conversa frente a frente.

g) Software funcionando é a principal medida de progresso.

h) Os processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de mantes um ritmo constante por tempo indefinido.

i) Atenção contínua, excelência técnica e um bom design aumentam a agilidade.

j) Simplicidade, a arte de maximizar a quantidade de trabalho não feito, é essencial.

l) As melhores arquiteturas, requisitos e designs surgem de equipes que se auto-organizam.

m) Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, ajustando seu comportamento de acordo.

2.2. Ágil x clássico

Para Martins (2007), as metodologias clássicas são abrangentes e apresentam resultados eficientes em diversas situações que são baseadas em processos já conhecidos. As metodologias ágeis, por sua vez, introduziram novas abordagens para tratar de projetos únicos elaborados com base no conhecimento e criatividade. Empresas inseridas em ambientes de grande incerteza e baixa previsibilidade vão apresentar resultados muito melhores quando utilizando das metodologias ágeis.

Diversos processos, que são considerados iguais, podem ser executados de formas

semelhantes, como ocorre na construção civil ou na montagem e manutenção de equipamentos. Eventuais problemas que surgem no decorrer do projeto são facilmente identificados e corrigidos por equipes mais experientes. A incerteza interna desse tipo de projeto é muito pequena. O desenvolvimento de software, por outro lado, que é geralmente associado a descobertas científicas, é totalmente diferente. Esses projetos possuem muitos elementos desconhecidos que podem alterar totalmente o planejamento inicial (MARTINS, 2007).

Os métodos clássicos têm foco maior no planejamento e controle e qualquer alteração no planejamento é considerada uma anomalia que precisa ser corrigida. Os métodos ágeis, por outro lado, centralizam-se na execução e adaptação e, como apontam Shore e Wardem (2008), quando o negócio precisa de uma mudança ou novas informações são descobertas, as equipes ágeis mudam de direção para se adequarem. As Figuras 1 e 2 sumarizam a evolução de um processo clássico e um ágil respectivamente.

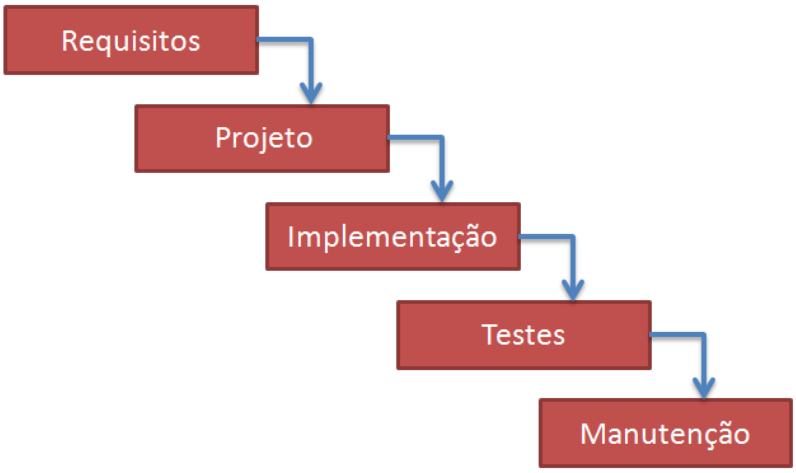

Figura 1: Modelo Clássico - Processo Cascata

Figura 2: Processo Ágil

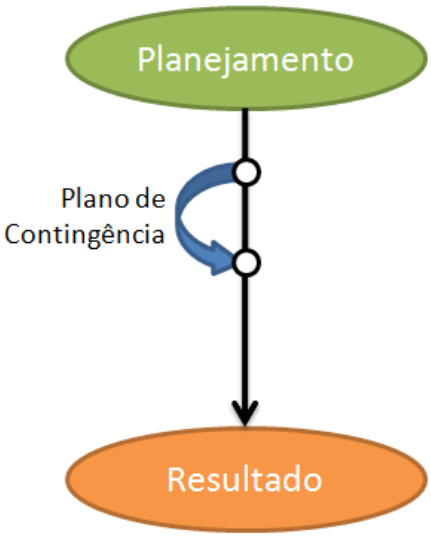

Figura 3: Abordagem Clássica

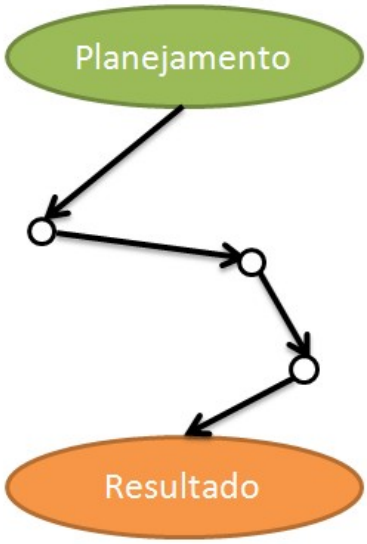

Figura 4: Abordagem Ágil

Na abordagem ágil, a ocorrência de alterações no processo são consideradas normais e providências para adaptação e adequação as novas realidades são executadas (Figura 4). Por outro lado, no processo Cascata, pode-se observar que a natureza linear do processo é seu maior problema. Dentro dessa abordagem as premissas para contornarem resultados não esperados nos processos intermediários são considerados planos de contingência, que não deveriam ocorrer, conforme observado na Figura 3. Para amenizar essa questão surgiram os modelos de processos de software evolucionários que combinam o desenvolvimento sequencial linear com a filosofia iterativa da prototipagem e definição de requisitos. Dentre os modelos evolucionários pode-se destacar o Incremental e o Espiral.

a) Modelo incremental: o sistema é dividido para ser desenvolvido em incrementos e cada incremento é disposto para desenvolvimento em uma sequência linear, como apresenta a Figura 5. Um exemplo de software desenvolvido segundo esse modelo pode ser um editor de textos, onde no primeiro incremento seriam elaboradas as questões de manipular arquivos como abertura e visualização, no segundo incremento a capacidade de edição do arquivo, no terceiro a verificação sintática e gramatical e, no quarto incremento, métodos avançados de visualização. Percebe-se que em cada incremento pode-se efetuar o seu planejamento técnico e assim definir a arquitetura do componente levando as informações que estão disponíveis (PRESSMAN, 2006).

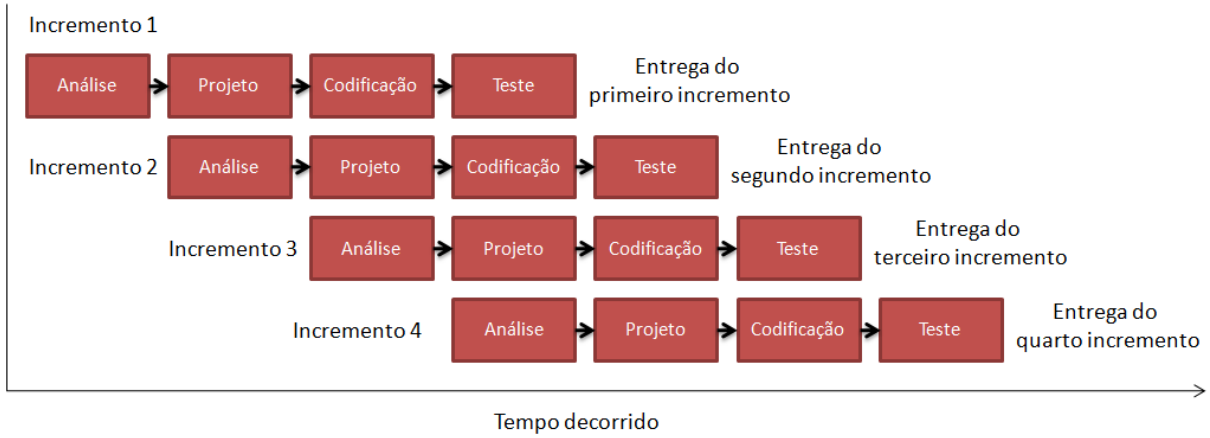

Figura 5: Modelo Incremental

b) Modelo Espiral: foi introduzido por Boehm (1985), apresentado na Figura 6. Nessa metodologia o processo de desenvolvimento segue a linha da espiral e, em cada ponto onde são esperados protótipos, o usuário define se a versão está finalizada ou deve voltar para as camadas iniciais para novas definições. Conforme a evolução do projeto ocorre sobre a espiral o software vai recebendo novas funcionalidades até completar seu planejamento.

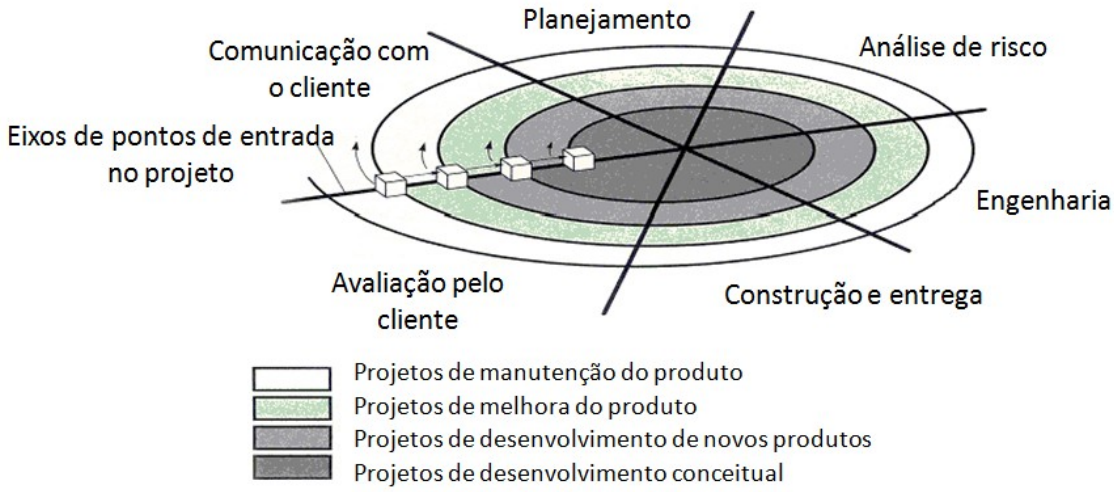

Figura 6: Modelo Espiral

Embora os modelos evolucionários apresentem uma abordagem para tratar das contingências que podem interferir no projeto técnico (gestão de riscos), o processo continua sendo executado de forma linear, onde os requisitos são definidos na fase de definição de requisitos, o projeto técnico na fase de projeto e assim sucessivamente (SCHWABER, 1985). Também, o principal ponto de controvérsia e discussão permanece, o desenvolvimento preditivo. A rigidez imposta por estes métodos, muitas vezes, reprime a flexibilidade que é

necessária no desenvolvimento de software em ambientes complexos.

Camara (2007) expõe alguns pontos em que os método ágeis são diferentes das propostas tradicionais para o desenvolvimento de software. Comparando com as melhores práticas para gestão de projetos - para a abordagem clássica - apontadas pelo Project Management Institute - PMI, verifica-se a orientação para aplicação dos recursos da empresa como apresenta a Figura 7. Por outro lado, a orientação ágil é mais próxima a representação da Figura 8.

Figura 7: Orientação PMI

Figura 8: Orientação Ágil

Nas figuras 7 e 8, as setas da esquerda mostram uma posição constante dos recursos, as setas intermediárias semi-variáveis e as setas da direita são as mais variáveis. Nota-se que enquanto os processos tradicionais tem maior preocupação em desenvolver o que foi definido como requisito para o sistema, a abordagem ágil procura, no tempo disponível, entregar os recursos que são mais importantes para o cliente, podendo variar o escopo de desenvolvimento para melhor adequação do processo.

Nas seções a seguir são descritas as duas metodologias ágeis utilizadas como base para a definição do método proposto nesse trabalho. O *Scrum*, que contempla atividades voltadas a gestão do processo de desenvolvimento e o *Extreme Programming*, mais direcionado às práticas de Engenharia de Software, fornecendo técnicas aplicáveis ao projeto e programação de sistemas.

2.3. *Scrum*

O *Scrum* é um processo leve para gerenciar e controlar projetos de desenvolvimento de software que segue a filosofia iterativa e incremental. Foi formalizado em 1995 por Ken Schwaber e Jeff Sutherland (SCHWABER, 2005). O termo *Scrum* tem origem no jogo de Rugby, é o nome usado para a reunião em círculo de jogadores para planejar a próxima jogada. Em uma analogia, o projeto é planejado etapa por etapa e executado em ciclos, mas com uma visão de longo prazo que é ganhar o jogo (Martins 2007). Para Schwaber (1995) a principal diferença entre a abordagem prescritiva (Cascata, Espiral e Iterativa) e a empírica (*Scrum*) é que esta última presume que a análise, projeto e o processo de desenvolvimento são imprevisíveis.

O metodologia *Scrum* é baseada no controle empírico de processos e seus fundamentos tem origem na indústria japonesa de manufatura, tendo como principais contribuintes os modelos de Produção Enxuta (*Lean Manufacturing*). Essa abordagem é oposta ao modelo cascata, onde inicia-se a análise logo que algumas definições já estejam disponíveis. O projeto é desenvolvido em etapas pequenas, ditas iterações, e para cada iteração são definidos os requisitos, projeto, programação e testes, resultando em ciclos iterativos com várias entregas de software ao longo do processo.

A Figura 9 mostra o processo *Scrum*. O projeto inicia-se com uma visão do produto, que pode ser vaga no início, uma pequena descrição técnica ou do ponto de vista comercial. A partir dessa visão é definido o *Product Backlog*, que é uma lista de itens e funcionalidades que o sistema deve possuir para que a visão seja concretizada. Estas funcionalidades também são chamadas histórias, ou seja, semelhante a descrição de um caso de uso para o item em questão. A ordem ou prioridade dessa lista de funcionalidades é definida pelo *Product Owner*, que é a pessoa ou empresa a quem o produto é destinado, o cliente. A prioridade define o que a funcionalidade gera de valor para o negócio do cliente. Após a definição da visão e dos itens iniciais, é escolhido um conjunto de funcionalidades, com base na sua prioridade, que irá ser desenvolvido dentro da primeira iteração. Tenta-se desenvolver as funcionalidades de maior importância para o cliente na primeira iteração e, desse forma, entregar valor o mais rápido possível. Geralmente as funcionalidades são agrupadas de forma a criar uma *release* ou software funcional ao final de cada ciclo de desenvolvimento. Os ciclos de desenvolvimento ou iterações são chamados *Sprints* e a lista de funcionalidades definidas para cada *Sprint* de *Backlog* do *Sprint*. Os *Sprints* de desenvolvimento possuem tamanhos definidos pela pela

equipe, porém recomenda-se a duração de trinta dias ou menos.

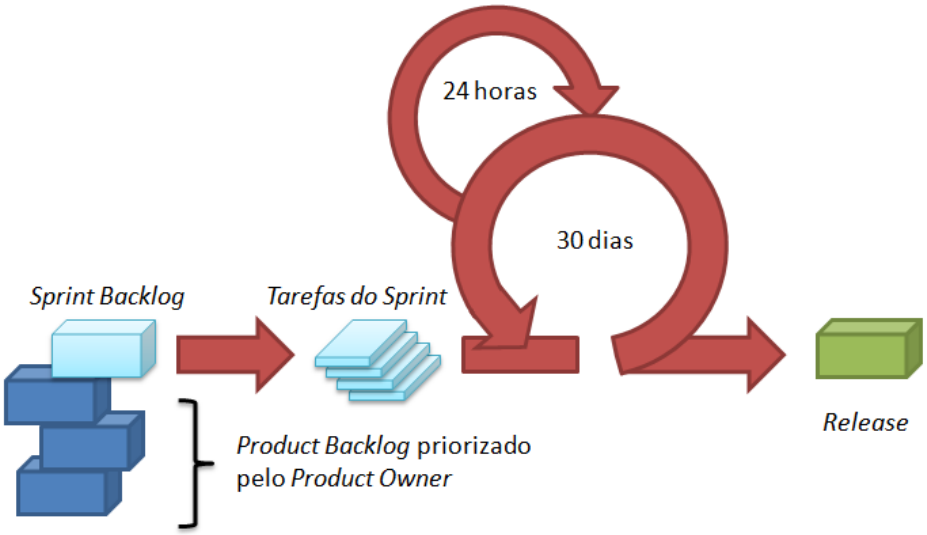

Figura 9: Processo do *Scrum*

Ao final da cada *Sprint*, o que foi desenvolvido é apresentado para o cliente e aos *stakeholders*, que vão avaliar a versão do produto, adicionar, modificar ou remover funcionalidades no *Backlog* do Produto e confirmar os itens que serão desenvolvidos no próximo *Sprint*.

2.3.1. Papéis no *Scrum*

O *Scrum* define basicamente três papéis principais o *Product Owner*, o *Scrum Master* e o *Scrum Team*.

a) *Product Owner*: representa dono do projeto, é ele quem define o cronograma e as prioridades de desenvolvimento. Também é responsável por assegurar que o mais importante para o negócio seja desenvolvido primeiro através de suas escolhas. Esse cargo é representado, provavelmente, pelo cliente, ou por um gerente de projetos, ou um membro da equipe de marketing ou um cliente interno.

b) *Scrum Master*: é o responsável por estabelecer e ensinar as práticas e regras do

Scrum e fazer com que as pessoas envolvidas as sigam, blindar a equipe contra interferências externas e remover os obstáculos ao desenvolvimento do projeto. Deve possuir habilidades de desenvolvimento e de engenharia de software. Suas responsabilidades podem ser resumidas em:

- Remover os impedimentos entre o *Project Owner* o *Scrum Team*, de modo que o primeiro possa orientar o desenvolvimento selecionando o que é mais importante. Para tanto, deve afastar ambos das atividades burocráticas e demais processos que atrasem ou dificultem sua interação;

- Manter as informações sobre a evolução do projeto atualizadas e disponível a todos os participantes do projeto;

- Auxiliar o *Product Owner* nas suas decisões por meio do conhecimento técnico.

c) *Scrum Team*: é o grupo de pessoas responsável por desenvolver as funcionalidades do produto. A equipe deve ser auto-organizada e multifuncional, com pessoas com diferentes conhecimentos técnicos. O *Scrum Team* pode ser composto por programadores e participantes externos, como por exemplo, pessoas de marketing e clientes, sem cargos específicos. O número recomendado é de sete colaboradores, tendo o mínimo de dois participantes.

2.3.2. Fases e práticas

O *Scrum* possui 3 fases: *Pregame*, *Game* e *Postgame*. No *Pregame* e *Postgame* os processos são bem definidos e os conhecimentos de como realizar essas etapas são explícitos. Na fase de *Game* os processos são empíricos e tratados com uma caixa preta. Dentro dessa fase intermediária ocorrem os *Sprints* onde o produto é desenvolvido. A seguir são descritas com maiores detalhes as três fases.

a) *Pregame*: tem duas partes, o Planejamento e a Arquitetura. No Planejamento são definidos os componentes e funcionalidades que a *release* a ser desenvolvida irá possuir. Com base nas informações disponíveis até o momento é formado um *Backlog* com estimativas de

prazo e custo. Podem-se destacar como atividades principais dessa fase:

- Desenvolvimento de uma lista com as funcionalidades ou histórias que o sistema deve contemplar;
- Definição do número de *releases* e suas datas de entrega;
- Definição de equipe do projeto;
- Estimativas de custos e tempo de desenvolvimento para cada história, e consequente definição das funcionalidades que irão integrar os *Sprints* e *releases*.
- Definição do que será desenvolvido primeiro.

Na Arquitetura são definidos os detalhes de implementação das funcionalidades que compõem a *release*. As principais atividades dessa etapa são:

- Definição do contexto de desenvolvimento e seus requisitos;
- Rever itens do *Backlog* e efetuar mudanças necessárias em vistas da análise técnica;
- Definir o projeto de implementação das funcionalidades do *Backlog*.

O *Pregame* deve gerar informações suficientes para o início do primeiro *Sprint*. Com base no itens do *backlog* e suas estimativas de custos e tempos é possível organizar a execução do primeiro ciclo. O planejamento não deve ser muito aprofundado, pois com a evolução do desenvolvimento podem surgir novos requisitos que influenciarão nas definições dos próximos *Sprints* e *releases*.

b) *Game*: essa fase é formada pelos *Sprints* de desenvolvimento das funcionalidades que irão compor as *releases* planejadas. Respeitando as questões de prazo, custo e qualidade, dentro dessa fase, além dos *Sprints*, são realizadas reuniões com a equipe para rever o planejamento do projeto. Dentro dos ciclos ou *Sprints* é que se dá a elaboração do sistema pelas etapas de desenvolvimento, empacotamento, revisão e ajuste. O *Sprint* possui um

objetivo e um tempo definido, chamado *time box*, que normalmente varia de uma a quatro semanas. As práticas mais comuns dento do ciclo são:

- *Sprint Planning Meeting*: de posse da lista de *Backlog* priorizado pelo *Product Owner* com as estimativas, é feita uma revisão do que deve ser desenvolvido no *Sprint* atual, bem como o que irá integrar a próxima *release*. Os itens do *Backlog* são atribuídos aos integrantes da equipe e da-se início ao ciclo de desenvolvimento.

- *Daily Scrum Meeting*: são reuniões diárias de quinze minutos no mesmo horário e local com todos os integrantes da equipe. Tem objetivo de verificar o andamento da execução das tarefas e identificar dificuldades para que sejam tomadas ações corretivas. Nessa reunião cada integrante faz um breve relato das atividades realizadas no dia anterior e o que possui de tarefas para desenvolver.

- *End-of-Sprint Revie*w: é a reunião de fechamento do *Sprint*, onde toda equipe deve estar presente. Também podem participar pessoas de marketing, vendas e clientes. São demonstradas as funcionalidades desenvolvidas e explicadas as alterações necessárias para atender aos itens definidos no *Backlog*. Nessa etapa podem ser inseridos novos itens no *Backlog* e modificadas as direções do que foi planejado anteriormente, como as ordens de prioridade e estimativas de custos e tempo. Ainda nessa reunião, de forma conjunta é definido o objetivo do próximo *Sprint*.

Se a equipe perceber que pode adicionar novos itens ao *Sprint*, deve consultar o *Product Owner* para verificar quais outras funcionalidades poderiam ser incluídas no ciclo. Caso seja constado que as funcionalidades não serão todas entregues no prazo estipulado, o ciclo pode ser interrompido e uma reunião é marcada para o planejamento do próximo *Sprint* e atualização do *Product Backlog*.

c) *Postgame*: essa é a fase de fechamento, que ocorre após o *Product Owner* decidir que a *release* final está entregue e respeitando suas aspirações para o sistema. Nessa etapa o produto é preparado para distribuição, incluindo integração, testes adicionais, documentação de usuário, material para treinamento e marketing, dentre outros. Nessa fase ainda ocorrem reuniões de retrospectivas para avaliar como o processo foi conduzido, tendo como finalidade o aprendizado para a elaboração de projetos futuros.

Além das fases e práticas, o *Scrum* propõe uma ferramenta para o controle da evolução das tarefas, o *Scrum Board*. O *Scrum Board* é um quadro onde as histórias ficam dispostas e pode-se verificar visualmente o seu estágio de implementação. Na Figura 10 pode-se visualizar a organização desse quadro. Um de seus componentes, importante para visualização do andamento do ciclo é o gráfico *Burndown*, que permite verificar a quantidade de funcionalidades que ainda restam ser desenvolvidas ao longo do processo.

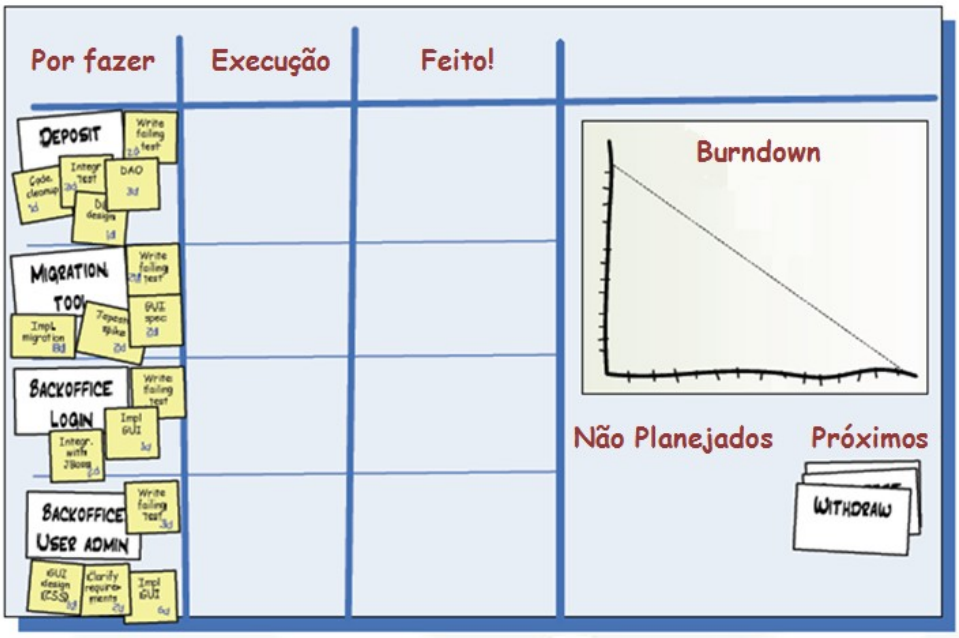

Figura 10: *Scrum Board*

O *Scrum*, apresenta um conjunto de práticas leves dinâmicas para gestão de projetos de desenvolvimento de software. Através das práticas que foram descritas, é possível atingir o nível 2 de certificação CMMI, e parcialmente o nível 3 (SIQUEIRA, 2007). Porém percebe-se que a intenção da metodologia é agilizar o processo, minimizando questões burocráticas e concentrando as atividades no desenvolvimento do software.

2.4. *Extreme Programming* - XP

O E*xtreme Programming* - XP é uma metodologia ágil para o desenvolvimento de software que se direciona, principalmente, a solucionar as limitações do processo (BECK, 1999). Esse método não aborda profundamente questões de planejamento e controle como o

processo de gerenciamento, análise financeira ou vendas. O XP é um estilo de desenvolvimento de software cujo foco está na busca pela excelência na aplicação de técnicas de Engenharia de Software (MARTINS, 2007).

O *Extreme Programming* parte do princípio que no desenvolvimento de software tudo muda, os requisitos, o projeto do sistema, o negócio, a tecnologia, a equipe. Sendo assim, a metodologia XP procura amenizar os impactos dessas variações que ocorrem ao longo do processo de desenvolvimento. Para Beck (1999) apud Martins (2007) o XP distingue-se das outras técnicas para o desenvolvimento de software pelas seguintes características:

a) Trabalha com iterações de curta duração, o que resulta em *feedbacks* rápidos e contínuos;

b) Utiliza uma abordagem incremental de planejamento que pode ser modificado conforme a evolução do projeto;

c) Usa mecanismos de testes automatizados, que são escritos pelos desenvolvedores, clientes ou pessoal da qualidade, permitindo com os que os erros possam ser identificados no momento que novas funcionalidades sejam adicionadas ao sistema;

d) Confia na comunicação oral, nos testes e no código-fonte para comunicar a estrutura do sistema e seus objetivos;

e) Confia no trabalho colaborativo entre os membros da equipe, tendo esses, diferentes habilidades técnicas.

2.4.1. Fases

O XP trabalha com o processo de desenvolvimento em ciclos que duram, geralmente, uma semana. A ideia é entregar software funcional para o cliente ao final de cada ciclo. Dentro do ciclo a equipe desempenha, de forma simultânea, quase todas as atividades de desenvolvimento. A análise, projeto, codificação e testes ocorrem de maneiras muito rápidas.

O desenvolvimento é orientado a cenários ou histórias, funcionalidades do sistema que tem valor para o cliente. Dentro de um ciclo de uma semana são desenvolvidos de cinco a dez cenários. Ao longo do ciclo, cada cenário passa por todas as fases de desenvolvimento que são: planejamento, análise, projeto e codificação, testes e implantação, como apresenta a Figura 11 (SHORE & WARDEN, 2008).

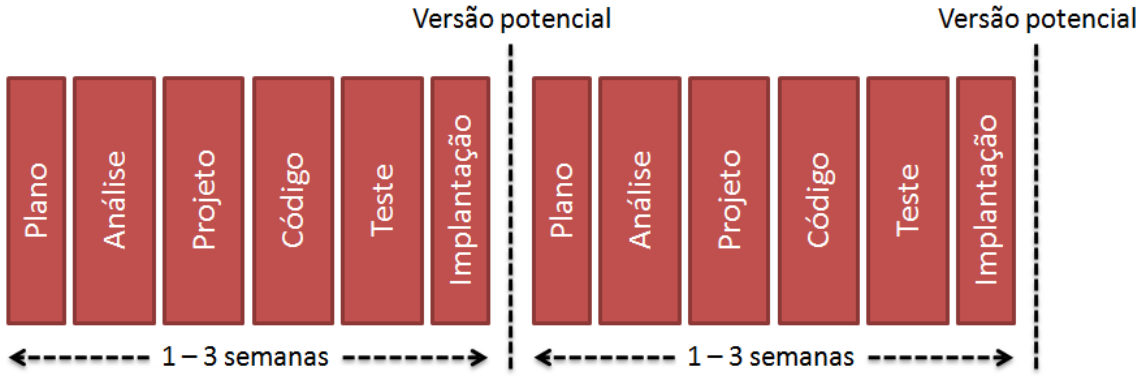

Figura 11: Processo do XP

a) Planejamento: a equipe do XP possui especialistas em negócios, clientes *on-site*, que são responsáveis pela visão do produto, separação do projeto em cenários, priorização dos itens para desenvolvimento e gerenciamento dos riscos. A equipe de desenvolvimento fornece as estimativas levando em consideração as características técnicas do projeto. Nas primeiras semanas a equipe se concentra em desenvolver uma visão geral do projeto e planeja as entregas de versões. No início de cada ciclo a equipe cria um plano de execução e traça as metas do ciclo, além de revisar a visão geral para modificar ou corrigir o plano. Diariamente são realizadas reuniões rápidas no ambiente de trabalho para manter todos os envolvidos sobre o andamento do processo.

b) Análise: ao invés de efetuar a análise no início do projeto, ou somente no início do ciclo, no XP, a equipe de clientes *on-site* passam bastante tempo com a equipe de desenvolvimento. Esses especialistas em negócios tem como função organizar os requisitos e repassar o que deve ser desenvolvido aos programadores quando forem solicitados, ou seja, quando os desenvolvedores precisam de informações, simplesmente as pedem.

c) Projeto e codificação: o desenvolvimento no XP utiliza a abordagem incremental, onde pequenas partes do projeto são programadas e integradas continuamente. Os programadores são responsáveis por organizar seu ambiente de desenvolvimento, decidem

sobre o controle de versões e formas para manter o processo de desenvolvimento automatizado.

d) Testes: no XP o desenvolvimento, comumente, segue a prática do *Test Driven Development* - TDD. Nessa técnica os casos de uso do sistema são escritos na forma de testes e as funcionalidades são desenvolvidas para executar com sucesso nos testes. Dessa forma, o sistema tende a apresentar poucos *bugs*, pois a cada novo incremento é executada a bateria de testes que vai se ampliando no decorrer do projeto. Os clientes *on-site* também participam dos testes, tentando identificar *bugs* de interfaces ou demais problemas que são difíceis de serem capturados pelos testes automatizados.

e) Implementação: ao final de cada iteração ou ciclo, o software deve estar pronto para implantação. O sistemas é implantado internamente ao final de cada ciclo para demostração, e a implantação no cliente real é agendada de acordo com as possibilidades do negócio do comprador. Dependendo da organização da empresa, a equipe de desenvolvimento pode continuar mantendo o software ou repassar para outra equipe responsável pelo suporte. Na conclusão do projeto, nas últimas semanas, são efetuados a redação dos manuais, a implantação e treinamento finais.

2.4.2. Práticas

Como apresentou Martins (2007), qualquer prática pode ou não ser incluída dentro do processo, o que determina sua utilização é o escopo de atuação do projeto e suas características perante o ambiente de desenvolvimento. Mesmo práticas que não foram descritas por Beck (1999) podem ser adicionas no processo se estas contribuírem para sua melhoria e agilidade. Ainda, para Shore e Warden (2008) toda equipe que utiliza XP terá sua própria maneira de praticá-la.

As práticas básicas do XP são:

a) Cenários ou histórias: como ocorre no *Scrum*, as histórias representam funcionalidades da forma como são percebidas pelo usuário. Por exemplo: "como administrador financeiro da empresa vou inserir as despesas geradas no sistema". São textos

simples e objetivos que procuram orientar a equipe no desenvolvimento das funcionalidades do sistema. Normalmente, os cenários são descritos pelos próprios clientes e anotados em cartões de papel, que posteriormente são afixados numa parede para permitir a visualização frequente por parte da equipe. Juntos das histórias podem ser anexados outros documentos que facilitem a sua implementação como algoritmos e gráficos. Também são descritos critérios de aceitação que permitem identificar se a funcionalidade foi desenvolvida corretamente como testes ou o resultado esperado para a execução do sistema.

As estimativas para cada história podem ser calculadas de forma semelhante ao *Scrum*. O tempo de duração para a implementação de cada cenário e calculado em dias. Por exemplo, a funcionalidade "carregar imagens no sistema" custa três dias de trabalho ininterrupto de oito horas ou possui três *story points*. Então, dentro de um ciclo, com base no número de programadores e das horas totais disponíveis, pode-se calcular quantos *story points* podem ser desenvolvidos na iteração. Assim selecionam-se a quantidade de histórias suficientes para completar as horas de desenvolvimento disponíveis no ciclo.

b) Pequenas *releases* e iterações de uma semana: no XP as *releases* devem ser entregues ao cliente com as funcionalidades mínimas que tenham valor para o negócio e que permitam um rápido *feedback*. As *releases* devem ser desenvolvidas em poucos ciclos. Os ciclos ou iterações devem ser planejadas semanalmente, conforme descrito anteriormente.

c) Reuniões s*tand-up*: nessa prática são realizadas reuniões diárias em pé de, no máximo, quinze minutos com a finalidade de verificar o andamento do ciclo. Nessa reunião os membros da equipe descrevem as atividades realizadas e o que tem por fazer até a próxima reunião. A forma como a reunião é realizada justifica-se pela agilidade que o processo deve ter, pois as pessoas normalmente não conseguem ficar muito tempo discutido e planejando em pé.

d) Projeto técnico incremental: a equipe deve trabalhar no projeto técnico diariamente. As questões de *design* devem atender as necessidades do momento. Ao não se planejar todas as questões técnicas do projeto juntamente com sua visão, evita-se a perda de tempo com modificações ou aplicação de novas abordagens. O XP parte do princípio que o conhecimento de como resolver os problemas vai aumentando com a evolução do desenvolvimento, portanto o projeto técnico deve ser realizado em pequenas partes atendendo somente aos requisitos de cada etapa.

e) Teste antecipado: no XP, antes do código começar a ser escrito, testes automatizados devem ser programados para permitir com que as funcionalidades a serem desenvolvidas possam ser verificadas. A intenção dessa prática é corrigir os erros no momento em que eles acontecem, pois a cada nova funcionalidades adicionada, o conjunto de testes é executado para verificar inconsistências no sistema. O software precisa passar por todos os testes de unidade e de aceitação, que são criados ao longo do desenvolvimento, para ser entregue ao cliente.

f) *Refactoring* de código: essa prática prega que o código, mesmo já desenvolvido e testado, pode sofrer evolução. O *refactoring* tem por objetivos melhorar a qualidade do código, tornar a solução mais simples ou é utilizado para refazer a programação de histórias com comportamentos indesejados.

g) Programação em pares: esse á a prática da Programação Extrema que causa maior polêmica e contradições entre os Gerentes de Projetos. Alguns apontam que a programação em pares pode ter benefícios como melhorar a qualidade do código, evitar o retrabalho, melhorar a comunicação da equipe, ou simplesmente aumentar a produtividade por ter uma pessoa sempre ao lado cuidando o que é feito em frente ao computador. Os programadores trabalham juntos verificando os mínimos detalhes de implementação. Por outro lado, existem Gerentes que afirmam que a técnica é um desperdício de recursos e que não é prático e eficiente duas pessoas trabalhando no mesmo código. De qualquer forma, essa prática é muito popular e utilizada entre os seguidores do XP.

h) Código compartilhado: no XP todo o código desenvolvido é de autoria global. Em qualquer momento, qualquer programador pode efetuar melhorias no código independentemente da autoria original. A prática de programação em pares auxilia na consciência do código coletivo. A equipe deve adquirir um senso de responsabilidade comum sobre a implementação e buscar aumentar a qualidade do produto.

i) Integração contínua: sempre que uma funcionalidade é terminada e passa pelos testes, deve ser integrada a versão corrente do software. Os programadores integram seu código continuamente, o que permite a equipe lançar *releases* sempre que houver sentido para o negócio do cliente.

j) Envolvimento real dos clientes: ter o cliente em contato direto com a equipe de desenvolvimento facilita a compreensão dos requisitos e acelera o *feedback*. Essa relação sem intermediários evita o desperdício de tempo e facilita a negociação de que funcionalidades devem ser desenvolvidas na sequência do processo.

l) Padrão de codificação: no processo descrito pelo XP quanto mais as pessoas trabalharem de forma coletiva maior será a padronização da forma como o código é produzido. A forma mais comum de padronização é o padrão de nomenclatura, que define a forma como as classes, interfaces, propriedades, métodos e variáveis do sistema são nomeadas. Essa prática facilita o entendimento do código pelos programadores que não o desenvolveram originalmente, o que auxilia nas atividades de *refactoring* e correção de *bugs*.

Pode-se perceber que o XP é uma metodologia para desenvolvimento que foca em práticas de Engenharia de Software que auxiliam nas mudanças que surgem constantemente no decorrer dos processos de software. Os testes constantes do sistema são trabalhados para minimizar o surgimento de defeitos e as integrações contínuas permitem que o cliente tenha software funcionando com as novas funcionalidades tão logo sejam desenvolvidas.

O capítulo a seguir apresenta as particularidades da produção de software e direciona o escopo delimitado por esse trabalho, o desenvolvimento de software para aplicações científicas.

3. DESENVOLVIMENTO DE SOFTWARE

Conforme Sommerville (1995) o processo de software é um conjunto de atividades e resultados associados que produzem um produto de software. Para Schwartz (1975) as principais fases de um processo de software são: Especificação de requisitos; projeto de sistema; programação (codificação); verificação e integração. Em cada fase de um processo de software são executadas atividades inerentes para que os objetivos propostos possam ser atingidos. Segundo Pressman (2006), estas atividades constituem um conjunto mínimo para se obter um produto de software. Ainda existem questões de alocação de recursos e pessoal que precisam ser previstos quando do início de cada projeto.

O desenvolvimento de software não combina somente técnicas e ferramentas, é predominantemente uma atividade humana (ANGKASAPUTRA & PFAHL, 2004). Em uma empresa de software, os principais recursos para o desenvolvimento das atividades são providos pelas interações humano-computador. A necessidade de planejamento do processo produtivo torna-se indispensável para que haja um melhor aproveitamento desses recursos.

Alguns autores afirmam que o desenvolvimento de software é um processo empírico. Para Ziv (1997) a incerteza é inerente e inevitável ao desenvolvimento de software. Conforme Wegner (2005) não é possível especificar completamente um sistema interativo. Nota-se que o processo de produção de software apresenta alguma características que o fazem diferenciar-se de outras áreas da produção industrial. Um dos pontos principais para essa conclusão é a não linearidade desse tipo de processo.

Para DeMarco (2009) não se pode controlar o que não se pode medir. O desenvolvimento de software com componentes de inovação, com os quais se busca um maior retorno sobre investimento (ROI) pela diferenciação de mercado, são ainda mais arriscados e possuem processos de desenvolvimento mais imprevisíveis. Métricas de controle não funcionam muito bem nesse tipo de ambiente. DeMarco (2009) também afirma que a melhor forma para gerenciar o processo de desenvolvimento é organizar os colaborares e controlar somente as variáveis tempo e dinheiro.

No presente trabalho focou-se na produção de softwares científicos cujo processo de

desenvolvimento, de acordo com a análise de Segal & Morris (2009), apresenta diferenças que incluem a grande complexidade de algumas áreas de pesquisa, a impossibilidade de especificar todos os requisitos no início do projeto e a dificuldade de efetuar testes nos ambientes de pesquisa. Ainda acrescentam que as práticas utilizadas no desenvolvimento de softwares comerciais não podem ser aplicadas da mesma forma no desenvolvimento de softwares científicos.

Dessa forma, faz-se necessário a busca por um meio de sistematizar o processo produtivo de empresas desenvolvedoras de software que se enquadram nas características mencionadas. As metodologias ágeis fornecem possibilidades de organização e planejamento para a produção de software, permitindo que o processo seja simplificado e adaptável. A seguir serão apresentadas as particularidades do desenvolvimento de softwares científicos e, na sequência, a justificativa para a utilização das metodologias ágeis na elaboração do método de gestão proposto.

3.1. Software científico

Para Pressman (2006) as aplicações de softwares científicos e de engenharia, que são utilizadas em áreas do conhecimento que vão da astronomia à vulcanologia, da análise automotiva de tensões à dinâmica orbital do ônibus espacial, e da biologia molecular à manufatura automatizada, eram caracterizadas inicialmente por algoritmos "*number crunching*" (que processam números). Todavia, as aplicações modernas nessa áreas estão se afastando dos algoritmos numéricos convencionais. Projetos apoiados por computadores, simulação de sistemas e outras aplicações interativas começaram a adquirir características de tempo real e a serem utilizados nas mais diversas aplicações industriais.

O desenvolvimento de software científico foi tema da revista IEEE Software de junho/julho de 2008 (SEGAL & MORRIS, 2008) e janeiro/fevereiro de 2009 (SEGAL & MORRIS, 2009). O objetivo dessas edições foi apresentar a grande variedade de softwares científicos e suas aplicações, dentre as quais destacam-se sistemas que rodam em computadores de alto desempenho e sistemas embarcados em diversos instrumentos, para manipular, analisar ou visualizar informações. As revistas também expõem questões para a melhoria do processo de desenvolvimento de softwares científicos, abordando as relações entre usuários, desenvolvedores e os sistemas.

O utilização cada vez maior de softwares decorrentes de pesquisas no meio industrial e em aplicações comerciais vem expondo algumas questões pouco exploradas. Historicamente, pesquisadores costumavam desenvolver seus próprios programas para testarem seus estudos. Como analisou Wilson (2006 apud BAXTER et al, 2006), diversos cientistas e engenheiros passavam muito tempo de suas pesquisas escrevendo código, corrigindo *bugs*, e mantendo software, e não se preocupavam em fazer essas tarefas de uma maneira efetiva. Como resultado, grande parte do tempo dedicado aos seus estudos era consumido por tarefas de programação e não na investigação de suas hipóteses.

A crescente demanda comercial por esses softwares e a consequente busca por soluções profissionais capazes de fornecer um nível maior de personalização, implantação e suporte, tem expandido as possibilidades de negócios para empresas de base tecnológica que buscam desenvolver sistemas em parceria com laboratórios e grupos de pesquisas. Para Sanders & Kelly (2008), a complexidade de desenvolver softwares científicos que acompanhem os modelos e teorias propostos pelos cientistas é muito alta. Assim, os clientes para essas aplicações escolhem desenvolvedores que melhor gerenciam as constantes mudanças que ocorrem nesses ambiente dinâmicos.

Conforme Letondal & Zdun (2003), a Engenharia de Software para o desenvolvimento de sistemas decorrentes de pesquisas científicas, deve ser focada na rápida evolução do meio e adaptável às diferenças entre os envolvidos no processo e suas atribuições dentro do projeto. O processo de desenvolvimento de software em ambientes de pesquisa ocorre, de forma geral, como o modelo apresentado por Sander & Kelly (2008), demostrado na Figura 12.

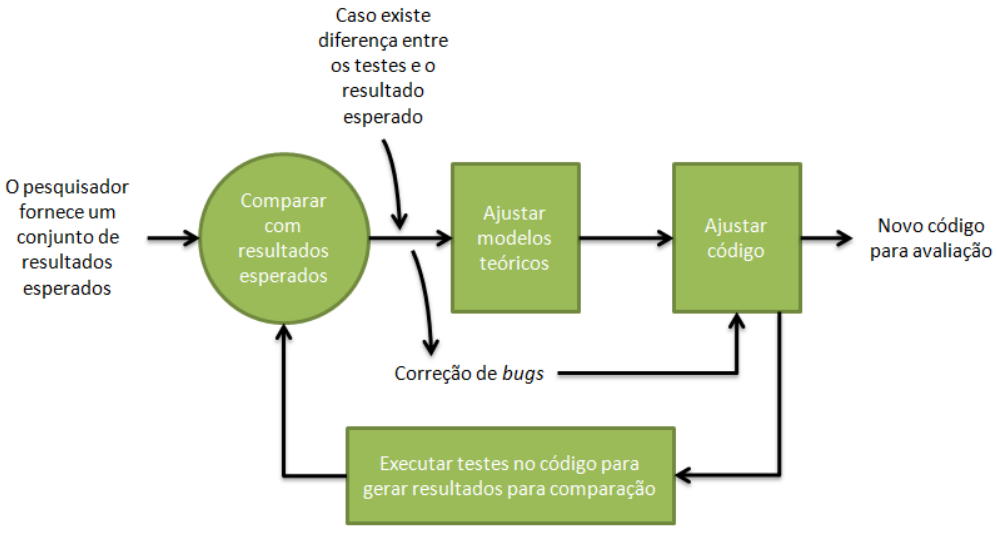

Figura 12: Processo de desenvolvimento em ambientes de pesquisa

A produção de software científico apresenta alguns problemas de fácil caracterização, como por exemplo os requisitos que emergem durante o processo de desenvolvimento. Os pesquisadores estão habituados a desenvolver software em seus laboratórios de uma forma bastante incremental, onde a descoberta de novos meios são aplicadas diretamente em suas pesquisas. A definição completa dos requisitos de software no início da pesquisa, como sugerem as metodologias tradicionais de desenvolvimento, é muito difícil de ser realizada. Outra dificuldade é o problema da comunicação. A documentação (especificação de requisitos) juntamente com as reuniões formais contratuais não são suficientes para o compartilhamento do conhecimento entre os cientistas e engenheiros de software.

Segal (2005) afirma que utilização de metodologias ágeis em projetos científicos combate diretamente o problema dos requisitos que surgem durante o progresso do desenvolvimento e a dificuldade de comunicação entre os envolvidos no processo. Além disso, outros fatores que envolvem o processo de produção de softwares científicos são beneficiados pela utilização das metodologias ágeis. A seguir, são descritas, de forma mais detalhada, as vantagens dessa abordagem.

3.2. Agilidade na pesquisa científica

O desenvolvimento de software científico, normalmente, é uma atividade multidisciplinar, onde diversas áreas do conhecimento podem estar envolvidas na elaboração de determinada aplicação. A gestão de projetos de desenvolvimento de software nessas condições exige certo grau de flexibilidade e adaptabilidade. A aplicação de metodologias ágeis para tanto, e como consequência, a redução de burocracias e aumento da comunicação entre as parte envolvidas, pode elevar as chances de sucesso do projeto.

Vários estudos recentes reportam o sucesso da aplicação de metodologias ágeis, principalmente o *Extreme Programming,* em projetos de desenvolvimento de software científico (ACKROYD, 2008; BAXTER et al., 2006; CRABTREE, C. et al, 2009; FRUHLING, 2005; KANE, 2006). Essas investigações mostram que os métodos ágeis se ajustam bem ao caráter exploratório, iterativo e de natureza colaborativa das pesquisas científicas. Os autores apontam que as práticas ágeis como testes automatizados, integração contínua, refatoração de código e comunicação constante favorecem o processo de

desenvolvimento de software baseado em pesquisas científicas.

Outras práticas das metodologias ágeis também se adaptam de forma satisfatória as particularidades da produção desse tipo de software. O fato da não especificação detalhada de requisitos no início de cada projeto é vantajoso para o processo, pois a produção de resultados é imprevisível e a adaptação e experimentação guiam o processo de desenvolvimento. Utilizar a prática ágil de desenvolvimento em ciclos, onde os requisitos são definidos com base nos resultados obtidos na iteração anterior, parece ser uma abordagem mais plausível e condizente com as práticas de exploração que ocorrem no desenvolvimento dessas aplicações.

Como apontou Segal (2005), o conhecimento e a solução dos problemas que surgem no decorrer do projeto podem aflorar de diversas fontes de conhecimento disponíveis, pois a multidisciplinaridade das equipes colabora para que isso ocorra. Um software que evolui de acordo com as novas descobertas é mais robusto e preparado do que aquele que segue uma descrição rígida pré-concebida. A possibilidade do ajuste da descrição dos itens que estão na lista de espera para produção, levando em consideração as novas informações que surgem no decorrer do projeto, contribui para a criação de um processo com riscos mais controlados.

Componentes de incerteza no planejamento é outro aspecto considerado na adoção de metodologias ágeis em processos de desenvolvimento de softwares científicos. A pesquisa por trás do desenvolvimento possui muitos pontos variáveis, que trazem consigo riscos que podem sensibilizar a execução do processo de produção. Essas variáveis podem alterar o planejamento do processo e a entrega das versões agendadas. No caso de uma hipótese levantada não ser passível de validação, pode ocorrer de todo o esforço da iteração de desenvolvimento ser perdido.

Outra prática ágil avaliada diz respeito à relação direta e frequente entre os desenvolvedores e seus clientes/pesquisadores. Ao invés da comunicação entre os participantes ser realizada apenas na definição inicial de requisitos, os métodos ágeis definem reuniões frequentes entre os *stakeholders* do projeto. Para o desenvolvimento científico, essa técnica faz com que a falta de conhecimento por parte dos executores e suas dúvidas sejam sanadas pelo relacionamento constante entre as partes envolvidas. Ou seja, um dos principais problemas levantados pelos desenvolvedores desse tipo de software, a comunicação, é minimizado. Conforme Sanders e kelly (2008), se um software apresenta um resultado não esperado, ambos cientistas e desenvolvedores devem ajustar a teoria e o respectivo código. Devem preparar novamente os dados de entrada e verificar os resultados obtidos de forma iterativa.

A seguir são relacionados alguns princípios apresentados no manifesto ágil (BECK et

al, 2001) que, conforme analisou-se, aprimoram o processo de desenvolvimento de software científico. Baseadas nesses princípios se fundamentaram as mais importantes metodologias ágeis para desenvolvimento de software utilizadas no mercado.

a) Entregas rápidas e frequentes de software com valor agregado. Permitem a avaliação constante da evolução do projeto e a entrega do que é mais importante primeiro, foca o processo em atividades com maior grau de relevância.

b) Mudanças nos requisitos do software são bem-vindas. Na pesquisa científica mudanças nos rumos das investigações podem ocorrer com relativa frequência, dependendo das descobertas que surgem ao longo do processo.

c) *Stakeholders* e desenvolvedores devem trabalhar juntos durante o projeto. A comunicação direta e constante facilita a resolução de problemas que possam ocorrer durante o processo de desenvolvimento, além de melhorar o entendimento dos requisitos do software.

d) Software funcionando é a primeira medida de progresso do projeto. Quão cedo os pesquisadores tenham o sistema para testar e validar, antes poderão ser efetuadas as modificações no projeto e no processo.

3.3. Validação e confiabilidade

Falhas de software são relativamente comuns. Para sistemas críticos, cujas falhas podem significar em perdas econômicas significativas, danos físicos ou ameaças a vida humana, medidas específicas devem ser tomadas para garantir confiabilidade, disponibilidade e segurança (SOMMERVILLE, 2007). Para tanto utilizam-se técnicas de Engenharia de Software, principalmente relacionadas a testes de validação, para assegurar a qualidade dos sistemas.

Em softwares não críticos, conforme analisou Segal (2005), onde o *time-to-market* é a prioridade mais alta, a realização de procedimentos para testes completos pode atrasar o lançamento dos produtos no mercado, e como consequência abrir espaços para concorrência. Empresas consolidadas no mercado tem o hábito de lançar suas versões de testes, com a

indicação *beta*, e dessa forma receberem o *feedback* dos usuários antes do lançamento das versão oficial. Essa prática para validação e correção funciona muito bem para produtos de massa, pois além de representar uma forma de divulgação, permite que a empresa possa testar seu produto em um ambiente real e receber sugestões de melhorias. Outra vantagem para a empresa decorre da sua não responsabilização por eventuais defeitos, visto que os usuários concordam com os termos da versão que, de fato, deveria ser utilizada somente para testes.

Dentro do contexto da validação e confiabilidade para softwares decorrentes de pesquisas científicas, alguns detalhes devem ser observados. Para Baxter et al. (2006), no desenvolvimento de softwares científicos, a capacidade de reprodução dos resultados é um dos pontos fundamentais. A competência de gerar resultados de forma consistente é utilizada como forma de medir a validade de um programa de computador e para tanto é necessário: certificar que o programa executa como deveria (através de testes), saber exatamente o que é utilizado para produzir os resultados, e reconhecer a rastrear os *bugs*.

Nesse tipo de software, frequentemente, os dados de saída são extensos e complexos. A precisão desses dados (ou um subconjunto desses) é dependente de uma grande quantidade de fatores e da interação dinâmica desses fatores. Na computação científica, o desenvolvimento do software se dá pela tradução dos modelos científicos propostos pelos pesquisadores em código fonte e, em uma equipe de desenvolvimento multidisciplinar, a responsabilidade pelas informações de entrada e a verificação dos dados de saída deve ser feito pelos cientistas envolvidos (KELLY & SANDERS, 2008).

Existem softwares decorrentes de pesquisas que além do âmbito dos estudos de laboratório, são utilizados no meio industrial ou em aplicações médicas. Esse tipo de aplicação deve observar normas e legislação correspondente antes da sua utilização comercial. No caso de aplicações médicas, onde softwares são utilizados para o auxílio a diagnóstico ou para o diagnóstico automatizado, esses sistemas podem ser considerados críticos pois atuam em questões relacionadas à vida humana.

As metodologias ágeis se enquadram muito bem no desenvolvimento de softwares onde a prioridade é o *time-to-market*. Porém softwares que possuem componentes críticos, no processo de desenvolvimento devem ser adicionados procedimentos para conferir a confiabilidade exigida por órgãos regulamentadores. Os processos ágeis apresentam uma filosofia leve para a produção de software, mas não excluem a possibilidade de inserção de procedimentos no processo de produção, mesmo que esses possam tornar a gestão do processo mais burocratizada. Dessa forma, pode-se adaptar o processo para que este atenda normas e padrões de qualidade desejados.

4. PROPOSTA DE MÉTODO DE GESTÃO

Shore (2008) afirma que cada projeto e situação são exclusivos, portanto, definir um processo ágil adaptado para cada caso é uma boa decisão. Os principais autores que debatem sobre metodologias ágeis, incentivam as equipes de desenvolvimento modelarem os processos ágeis de acordo com suas necessidades específicas. A ideia base das metodologias ágeis, além de permitir a adaptação, é garantir que haja sempre a busca pela melhoria da gestão do processo de desenvolvimento.

No presente trabalho, é proposto um método de gestão para o processo de produção direcionado ao desenvolvimento de softwares científicos, através da combinação das metodologias ágeis *Scrum* e *Extreme Programming*. O método tem por objetivo resolver a problemática do desenvolvimento de softwares de cunho científico para aplicações industriais e médicas. Na literatura das metodologias ágeis, muitas das práticas não são descritas em detalhes. Essas lacunas, muitas vezes, são propositais e tem a intenção de flexibilizar a utilização das práticas. Dessa forma, o método proposto busca preencher essas lacunas adaptando o processo às particularidades do desenvolvimento de softwares científicos.

Nas próximas seções são descritas as vantagens do processo de *Tailoring* de metodologias ágeis, é caracterizado o ambiente de desenvolvimento na microempresa, os processos de negócios que ocorrem e, por fim, apresenta-se o método de gestão para o processo de desenvolvimento.

4.1. *Tailoring* Scrum/XP

A personalização e combinação de metologias para o desenvolvimento de software é descrito na literatura de Engenharia de Software como processos de *Tailoring*. Como aponta Segal (2005), o processo de *Tailoring* é utilizado para ajustar a metodologia de desenvolvimento a um projeto em particular, levando o seu contexto em consideração.

Keenan (2004) afirma que projetos de software variam muito entre si, devido, principalmente, a sua escala, escopo e desafio técnico. A abordagem ágil, reconhece esse problema e apresenta alternativas que facilitam o processo de adaptação para as distintas situações.

Um número crescente de empresas vêm adotando subprocessos adaptados de metodologias ágeis. Para Mohamed (2009), o desenvolvimento tradicional de software falhou ao tentar gerenciar projetos onde surgiam vários problemas de mudanças de requisitos, enquanto as metodologias ágeis foram criadas exatamente para focar essa questão. Mohamed (2009) também afirma, que a experiência empírica mostrou que dificilmente uma metodologia ágil trabalha sozinha com os conceitos publicados originalmente. O que geralmente ocorre é a sua personalização para o domínio da aplicação ou uma adaptação de várias práticas de metodologias distintas.

O objetivo do *Tailoring* de processos de software é adaptar um processo definido na literatura para atender as necessidades específicas de uma organização ou projeto (PEDREIRA, 2007). Para Ginsberg & Quinn (1995 apud PEDREIRA, 2007) o *Tailoring* de processos de software pode ser feito em dois níveis diferentes dentro da empresa, no organizacional e no nível de desenvolvimento/produção. Para tanto, atividades não necessárias são removidas do processo e alguns elementos são adicionados levando em consideração o tipo de software que a empresa desenvolve. Essas medidas tem por objetivo evitar o desperdício de tempo e recursos.

No nível organizacional, dentro dos processos relacionados com a gestão de produção, definiu-se como metodologia base o *Scrum* que, conforme descrito na seção 2.3, possui práticas que se direcionam as atividades de planejamento e controle, focadas na gestão do projeto. No nível de desenvolvimento, onde as técnicas de Engenharia de Softwares são empregadas diretamente, buscou-se no *Extreme Programming* um grupo de práticas que melhor se enquadram no processo de desenvolvimento de softwares científicos. Essas escolhas tem como finalidade melhorar a qualidade do que é produzido e garantir agilidade no processo para responder às contingências do ambiente.

Conforme apontam alguns autores as metodologias ágeis *Scrum* e *Extreme Programming* são complementares (KNIBERG, H, 2007). Para Kane (2006) o Scrum não especifica práticas de engenharia a serem utilizadas, como definições para testes de software e padrões de codificação e, por tanto, a utilização do *Extreme Programming* para adicionar essas práticas ao processo é justificável. Conforme observa-se na Figura 13 existe uma pequena interseção entre Scrum e Xp, que diz respeito ao planejamento dos ciclos. Porém unindo as práticas de ambas as metodologias tem-se um processo mais completo que

contempla tanto questões organizacionais, como de produção. Dentro do ciclo de desenvolvimento o processo fica organizado como demostra a Figura 14.

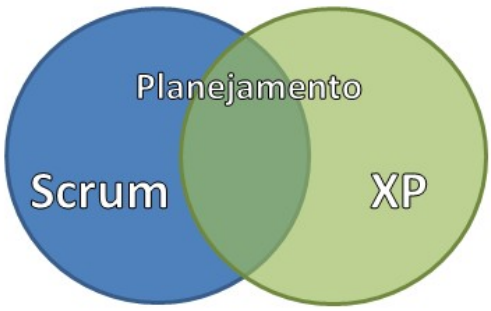

Figura 13: Interseção *Scrum*/XP

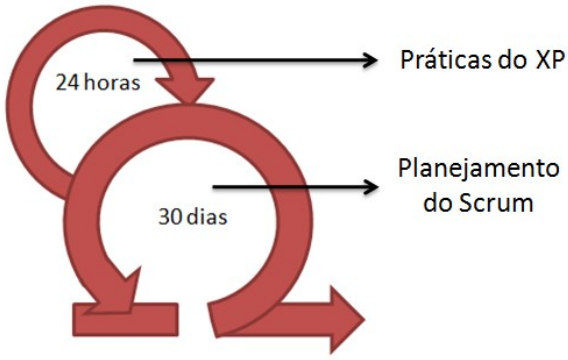

Figura 14: *Tailoring Scrum*/XP

Na sequencia é proposto um método de gestão para o processo de desenvolvimento que se baseia na combinação *Scrum*/XP e tem por objetivo atender as principais funções de planejamento, controle e execução de projetos de desenvolvimento de softwares científicos em uma empresa de base tecnológica.

4.2. Caracterização da microempresa

As organizações desenvolvedoras de software brasileiras, em sua maioria, são micro e pequenas empresas (MCT, 2005). Segundo informações do SEBRAE (2004), a mortalidade de 49,4% das micro e pequenas empresas brasileiras ocorre nos dois primeiros anos, 56,4% em três e 59,9% em até quatro anos de existência. Conforme a pesquisa, os principais fatores que contribuem para esses dados estão a forte concorrência, a inexperiência gerencial e de marketing dos proprietários, a alta carga tributária, a falta de capital de giro e a ausência de políticas internas voltadas para a garantia da qualidade dos produtos.

Segundo o SEBRAE, microempresas (MEs) são empreendimentos que empregam até nove pessoas no caso do comércio e serviços, ou até 19, no caso dos setores industrial ou de construção, refletindo a realidade de pequenas empresas de software, como as que estão inseridas em incubadoras de base tecnológica.

Conforme Habra et al. (2008), faltam recursos nas microempresas, tanto de pessoal como financeiros, atrelados somente ao processo de produção. Estas empresas tem, por definição, pequenas equipes e as pessoas envolvidas são pressionadas por prazos apertados e muitas por tarefas de produção associadas. A maioria das pequenas empresas de software não tem utilizado padrões para desenvolvimento como o CMMI ou ISO 9000 (O'CONNOR & COLEMAN, 2007), pois iniciativas de SPI (*Software Process Improvement*) incorrem em custos adicionais significantes. A adoção de mecanismos que padronizam o processo produtivo de software acaba burocratizando a empresa em excesso, restringindo sua flexibilidade.

Dessa maneira, no desenvolvimento de software são esperados, baixos custos, velocidade de produção, taxas de falhas aceitáveis e flexibilidade para adaptações às exigências do ambiente (ANGKASAPUTRA & PFAHL, 2004). A utilização de padrões de desenvolvimento significa o reuso de conhecimentos para soluções de problemas que foram bem sucedidas em projeto anteriores. A análise das atividades anteriores serve como parâmetros de alimentação e comparação para a implementação de novos projetos.

Em busca de flexibilidade, essas empresas acabam utilizando processos de desenvolvimento de software adaptados, baseado em métodos ágeis como XP (*Extreme Programming*), FDD (*Feature Driven Development*), *Scrum*, dentre outros. A organização define seu próprio método de gestão, fazendo com que sejam utilizadas somente atividades ou tarefas que permitam a fluência do processo produtivo, conforme abordado na seção anterior.

Levando-se em consideração os objetivos econômicos desse tipo de empresa, somente práticas ágeis, flexíveis e que incorram em redução de custos podem garantir o crescimento e evolução da organização. Para MacCormack (2001 apud FRANCO, 2006), o grau de sucesso de empreendimentos que utilizam a abordagem ágil para desenvolvimento de software está fortemente associado a sua capacidade de gerar:

a) Um ciclo de vida iterativo com lançamento do produto em evolução para a revisão e comentários dos *stakeholders*;

b) Incorporação diária de novos códigos com informações das mudanças de projeto;

c) Uma equipe com ampla experiência em concluir e entregar múltiplos projetos;

d) Atenção desde o início do projeto na arquitetura de componentes modulares e baixo acoplamento (módulos mais independentes).

Por outro lado, para Taylor et al. (2007), existe um falso julgamento de que adaptação e adoção de metodologias ágeis provocará melhoras de imediato nos produtos e na satisfação dos clientes. Principalmente empresas de pequeno porte são atraídas pelas promessas de simplificação e flexibilização de processos. Entretanto, pela natureza empírica dos métodos ágeis, quando da sua utilização, devem ser tomados cuidados que permitam avaliar a real eficiência das práticas adotadas. Microempresas, que são mais suscetíveis, podem não resistir ao impacto de uma experiência de implantação falha.

Para a elaboração e validação do método de gestão para o processo de desenvolvimento de software científico proposto nesse trabalho, são aplicadas as práticas discutidas na microempresa de base tecnológica Animati Computação Aplicada. A Animati está situada na Incubadora Tecnológica de Santa Maria - ITSM, surgiu como *spin-off* de um laboratório de pesquisa do curso de Ciência da Computação/UFSM, e desenvolve softwares de base científica para aplicações médicas, industriais e de pesquisa. A empresa possui em seu quadro funcional, mestres, mestrandos e alunos de graduação e, frequentemente, executa projetos de desenvolvimento de softwares científicos em parceria com o Laboratório de Computação Aplicada - LaCA/UFSM.

4.3. Abrangência dos processos

O foco da empresa Animati é o desenvolvimento de soluções direcionadas a aplicações de processamento e análise de imagens. Para desenvolver os sistemas, a empresa conta com profissionais em nível de graduação e mestrado na área. Além de seus pesquisadores, a empresa também trabalha em parceria com grupos de pesquisa do curso de Ciência da Computação/UFSM. A interação com professores e alunos do curso, facilita a absorção de novas tecnologias que são empregadas na elaboração dos projetos, bem como oportuniza temas de pesquisa para os alunos de graduação e mestrado.

A empresa surgiu com o objetivo de fazer a ligação entre a tecnologia que era produzida na forma de artigos, trabalhos de graduação e dissertações de mestrado e o potencial mercado consumidor para essas tecnologias em seu estado da arte. Essa relação

empresa-universidade traz benefícios para ambos envolvidos, tendo como resultado o fluxo de conhecimento e transferência de tecnologia para o mercado consumidor final.

Atualmente, os serviços prestados pela empresa Animati Computação Aplicada variam de acordo com o nível de interesse dos pesquisadores/cliente e o desejo de aplicação de suas pesquisas. De forma geral, os clientes da empresa podem ser subdivididos em dois grupos, os pesquisadores e os clientes de aplicações comerciais.

a) Pesquisadores: são os clientes potenciais para os softwares científicos produzidos pela empresa. Dentro do escopo dos sistemas de processamento e análise de imagens, surgem as mais diversas aplicações e áreas de pesquisas, por exemplo, sistema para a análise e caracterização de minérios através de imagens de microscopia, imagens de satélite para geoprocessamento, imagens de microscópio para análise de patologias celulares, imagens de radiologia para diagnóstico médico.

O conjunto de métodos para o processamento e análise de imagens podem ser utilizados em todas as áreas de pesquisa mencionadas, onde o objeto comum de análise são as informações extraídas das imagens digitais. A forma como essas informações são processadas e interpretadas variam de acordo com cada objetivo de pesquisa, onde existe uma grande possibilidade de evolução e novas descobertas. O desenvolvimento de novos métodos para o processamento e análise de imagens vêm ocorrendo através de pesquisas em laboratórios nacionais e internacionais. Alguns dos métodos já consolidados também são aperfeiçoados para serem executados de forma mais eficiente em novos hardwares.

De forma geral, os software decorrentes de pesquisas que são desenvolvidos dentro da Animati, se enquadram em uma das linhas descritas a seguir. São problemas de pesquisa que para sua resolução utilizam-se:

- Métodos cujos resultados de suas combinações são conhecidos. Dado um problema de pesquisa, a equipe consegue facilmente identificar os métodos de processamento e análise de imagens necessários para sua resolução. São métodos triviais, como segmentação das imagens por operações simples, medições de elementos na imagem, operadores para visualização e demonstração de propriedades das imagens. No planejamento do projeto, esse tipo de problema é o que contém riscos menores e maior precisão nas estimativas para o tempo de execução das tarefas de produção. Consideram-se projetos de risco leve.

- Métodos conhecidos e o resultados das suas combinações desconhecidos. Nesse caso, a equipe tem ideias de métodos que podem ser utilizados no processo, porém não conhece por completo os resultados que podem surgir das suas combinações dentro do domínio da pesquisa. No planejamento desse tipo de software deve ser considerada a possibilidade dos métodos não produzirem os resultados esperados, e sua consequente substituição. O risco é intermediário, pois a redefinição de tarefas e métodos de processamento pode alterar o planejamento inicial das histórias.

- Métodos e suas combinações desconhecidos. Nesse contexto tem-se o maior risco para o planejamento inicial. O processo deve ser muito dinâmico e flexível, permitindo ajustes constantes. Os pesquisadores devem estar envolvidos diretamente no processo, acompanhando a evolução dos resultados produzidos e sugerindo modificações nos métodos utilizados. São projetos de risco elevado.

Após a conclusão do desenvolvimento e validação dos sistemas, os softwares resultantes podem ser direcionados ao segundo grupo, os clientes de aplicações comerciais.

b) Clientes de aplicações comerciais: esse tipo de usuário utiliza o software que foi desenvolvido através de pesquisa como ferramenta de auxílio as suas atividades. Os sistemas são utilizados na indústria, laboratórios, clínicas médicas, dentre outros. A Animati, como modelo de negócios, faz a personalização, implantação, treinamento e suporte para as ferramentas que desenvolve.

Na Animati os processos de negócios ocorrem de duas formas principais: um pesquisador ou laboratório procura a empresa para o desenvolvimento do software de cunho científico, ou uma demanda de mercado é identificada pela empresa e esta, por sua vez, busca o conhecimento necessário para o desenvolvimento do produto/serviço, muitas vezes em parceria com outros pesquisadores. No Capítulo 5 serão descritos com maiores detalhes alguns sistemas desenvolvidos pela empresa.

4.4. Método de gestão proposto

O objetivo principal desse trabalho é a criação de um método de gestão para o processo de desenvolvimento de software científico dentro da microempresa, utilizando a abordagem proposta pelas metodologias ágeis. Para tanto, os métodos *Scrum* e *Extreme Programming* foram estudados e, através do processo de *Tailoring*, adaptados a realidade e ao contexto de uma empresa de base tecnológica que desenvolve softwares decorrentes de pesquisas científicas.

Com base na análise das metodologias ágeis *Scrum* e XP, pode-se verificar que ambas funcionam bem sendo combinadas para a criação de um processo ágil e adaptável. Enquanto o *Scrum* endereça questões referentes a organização do processo de produção, com práticas para o planejamento, execução e controle, o *Extreme Programming* é mais voltado para as técnicas de Engenharia de Software que são aplicadas diretamente no desenvolvimento das aplicações.

O método proposto foi dividido em dois níveis, o organizacional, baseando-se nas técnicas para planejamento, execução e controle propostas pela metodologia ágil *Scrum*, e no nível de produção/desenvolvimento onde algumas práticas do XP foram adaptadas e utilizadas.

4.4.1. Gestão do processo – nível organizacional

A gestão do processo no nível organizacional dá-se pela definição dos cargos/papeis ocupados pelas pessoas envolvidas e pela definição das fases do projeto com suas atividades respectivas. Os cargos básicos definidos são: Pesquisador, Gerente do Projeto e Equipe de Desenvolvimento.

a) Pesquisador: equivalente ao *Product Owner* do *Scrum*, é o responsável pela descrição do software a ser desenvolvido. O pesquisador traça os prazos e negocia com a empresa as questões referentes a valores e escopo de desenvolvimento. Esse cargo é muito importante para o projeto, pois além de fornecer as informações necessárias, modelos matemáticos, dados para validação, o pesquisador irá decidir as prioridades de desenvolvimento para as funcionalidades do sistema.

No caso do pesquisador ter dificuldades de deslocamento e participação nas reuniões, um membro da equipe o representa e se encarrega de repassar e receber as informações necessárias à execução do projeto. A figura do pesquisador também pode ser executada por um membro interno da equipe em projetos que tenham origem dentro da empresa, desde que possua as qualificações necessárias. Nota-se que um dos requisitos principais para o desempenho desse cargo é a qualificação técnica e domínio sobre o assunto de pesquisa que resultará no software.

b) Gerente do Projeto: tem função semelhante ao *Scrum Master*. Possui como responsabilidade organizar o processo, agendar e controlar as reuniões, os prazos, verificar o andamento das atividades e das práticas utilizadas. Deve conhecer questões relacionadas a Engenharia de Software e ter experiência no desenvolvimento de sistemas. De forma geral, o Gerente do Projeto deve remover os impedimentos do projeto e realizar as atividades burocráticas, permitindo que o processo tenha o fluxo desejado.

c) Equipe de Desenvolvimento: são os programadores e demais pessoas envolvidas com o desenvolvimento do sistema. É desejável que a equipe tenha condições de se auto-organizar, verificar o andamento do processo e sugerir melhorias para o que se está desenvolvendo. Uma equipe multidisciplinar é desejável visto que, na atividade de exploração científica, conhecimentos diversos colaboram para a resolução dos problemas.

As fases que organizam o processo de produção são: Planejamento Inicial, Execução e Fechamento do Projeto.

a) Planejamento Inicial: na fase de Planejamento Inicial as pessoas envolvidas com o processo se reúnem para definir uma visão geral do sistema. São definidas uma lista de funcionalidades que o software deve contemplar ordenadas pela sua prioridade, porém não são descritas com muitos detalhes. Cada item dessa lista deve conter, pelo menos, sua descrição, um caso de uso para sua validação e uma pré-estimativa de custo para seu desenvolvimento. Nessa etapa inicial definem-se também o tamanho dos ciclos de desenvolvimento (*Sprints*), as possíveis datas para a entrega de versões do sistema e as funções dos participantes do projeto. A reunião do Planejamento Inicial deve gerar informações e material suficientes para a definição do primeiro ciclo de desenvolvimento e seus objetivos. Os ciclos de desenvolvimento ocorrem dentro da fase de Execução.

b) Execução: nessa fase ocorre o processo iterativo de desenvolvimento através dos ciclos. A duração e o número de ciclos é definido no Planejamento Inicial e leva em consideração os prazos e os agendamentos para entrega de versões ou demostração do sistema. Geralmente, o final da cada ciclo coincide com a entrega de uma versão para validação. Cada ciclo possui três atividades principais: Planejamento do Ciclo, Reuniões Diárias e Fechamento do Ciclo.

- Planejamento de Ciclo: é realizada uma reunião para o planejamento do ciclo em seu primeiro dia de execução. Nessa reunião é discutido o objetivo do ciclo e com base na sua duração são calculados quantos itens da lista de funcionalidades podem integrar o ciclo em questão. As estimativas dos itens dessa lista são revistos e de acordo com a equipe disponível é calculado o número de dias/homem disponível. As funcionalidades tem estimativas individuais e calculadas em dias de trabalho. Então são selecionadas as funcionalidades de acordo com o número de dias/homem disponíveis. Ainda, é feita uma multiplicação do número de dias/homem disponíveis por uma variável chamada fator de foco, que tem por objetivo deixar a estimativa mais próxima do real. Por exemplo: se em um determinado ciclo tem duração de 10 dias e o número de programadores em tempo integral é 3, então teoricamente dispõe-se de trinta dias/homem (10 dias multiplicado por 3 programadores). Se for utilizado como fator de foco a variável 0,7, o número de dias homem cai para 21 (30 multiplicado por 0,7). Esse ajuste é feito pois os desenvolvedores não passam o dia inteiro de trabalho escrevendo código, existem intervalos para descanso, estudos e busca por soluções, correções de problemas, dentre outras atividades que não relacionam diretamente com a programação das funcionalidades do sistema.

- Reuniões Diárias: as reuniões diárias tem como finalidade o acompanhamento da evolução do processo. Nessas reuniões cada integrante descreve resumidamente no que trabalhou o dia anterior e o que possui para desenvolver ainda. O Gerente do Projeto pode identificar a necessidade da presença do pesquisador para solucionar detalhes de implementação das funcionalidades que não foram bem definidas, ou ainda avaliar problemas no andamento do processo como atrasos ou novas demandas. Se forem identificadas novas funcionalidades para serem desenvolvidas durante o ciclo em questão, esses itens são inseridos numa lista de espera chamada "itens não planejados". Se, dentro do ciclo houver tempo hábil para seu desenvolvimento, o

Gerente do Projeto pode adicioná-los a lista de "itens por fazer", caso contrário, dependendo de sua prioridade, são considerados no planejamento do próximo ciclo. Nessa reunião também são identificadas as funcionalidades que estão concluídas e o quadro de tarefas (*Scrum Boar*d) é atualizado. As histórias são posicionadas de acordo com sua situação no quadro e gráfico *Burndown* é traçado de acordo com as funcionalidades que ainda restam ser desenvolvidas.

- Fechamento do Ciclo: nessa reunião todas as pessoas envolvidas no processo devem estar presentes. São demostradas as funcionalidades que foram implementadas e são justificadas as alterações no planejamento devido a problemas eventuais que podem surgir no decorrer do processo. Nessa etapa, o pesquisador avalia a implementação do sistema e testa o conjunto de informações necessárias para validar o software. Após a validação ou não, o pesquisador pode sugerir novas funcionalidades ou solicitar modificações. Os novos itens devem seguir o mesmo processo de definição do planejamento inicial, com sua descrição, estimativa de custo e nível de prioridade, que serão considerados no planejamento do próximo ciclo. Essa reunião também é utilizada para se fazer uma retrospectiva do processo, com a finalidade de explorar pontos que podem ser melhorados em ciclos futuros.

c) Fechamento do Projeto: é a fase de finalização do projeto, ocorre dias após ou juntamente com a reunião de fechamento do ultimo ciclo de desenvolvimento. Toda equipe deve estar reunida para a demostração final de todas as funcionalidades do sistema. Após validação e aceite da finalização por parte do pesquisador, a equipe de desenvolvimento juntamente com o Gerente do Projeto definem as estratégias para realização de testes adicionais e preparação do software para distribuição, redação de manuais e implantação do sistema nos domínios do pesquisador. Também é feita uma reunião de retrospectiva para avaliar o processo, onde são identificados pontos facos e fortes que servirão de base para melhorias em projetos futuros. A Figura 15 mostra como funciona a organização do processo no nível organizacional.

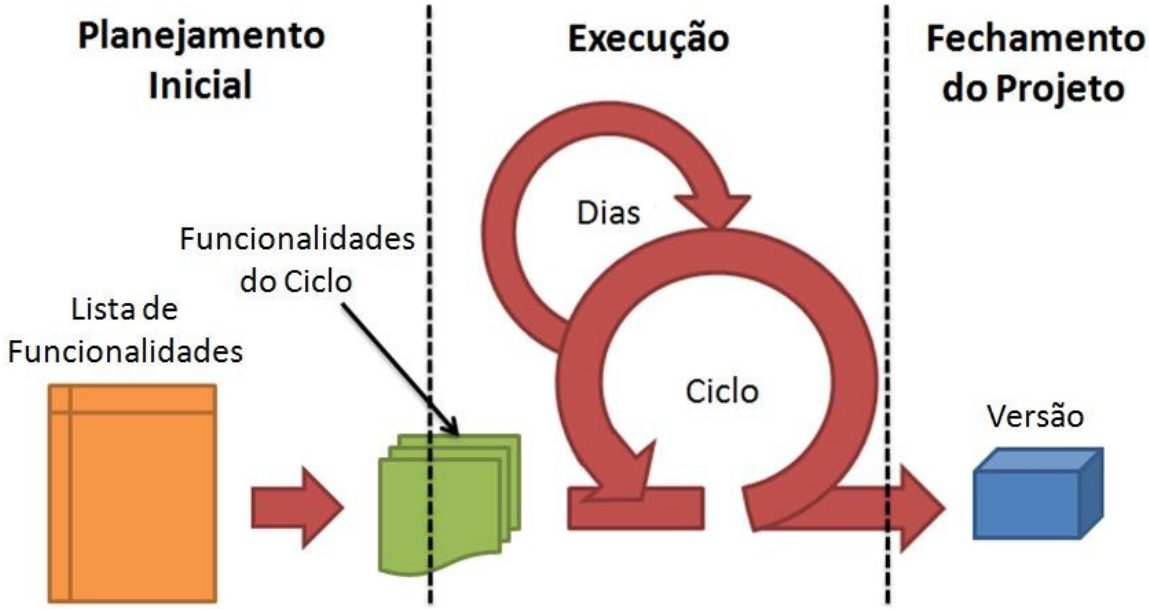

Figura 15: Gestão do processo – nível organizacional

4.4.2. Produção - nível de desenvolvimento

Existem pesquisas que apontam que o uso do *Extreme Programming* traz um acréscimo de produtividade ao desenvolvimento de softwares científicos, conforme descrito na seção 3.2. A maioria das práticas do XP se direcionam ao processo de codificação dos sistemas, englobando técnicas de Engenharia de Software. Dentre as práticas citadas na seção 2.4.2, a seguir são descritas aquelas que se adequam e contribuem para o desenvolvimento de softwares decorrentes de pesquisas científicas

a) Cenários ou histórias: representam os itens da lista de funcionalidades que serão implementadas no sistema. Esses itens são descritos inicialmente pelos pesquisadores e discutidos em equipe para verificação de sua possibilidade de implementação. A equipe, baseada em sua experiência, estima os custos de desenvolvimento em dias para cada funcionalidade. Cada história contém sua descrição, uma estimativa para seu desenvolvimento, sua prioridade, uma identificação única e um caso de uso para sua

validação.

No caso das histórias representarem um modelo matemático, ou método científico passível de testes para sua validação, juntamente da funcionalidade devem estar descritas informações para tal. De acordo com Baxter (2006) bons cientistas não começam a desenvolver seus experimentos sem descrever os materiais e métodos para testar suas hipóteses. De forma similar, antes de começar a programação do método científico, o seu projeto de software deve conter detalhes importantes de implementação. Essa descrição não precisa ser muito grande, mas deve responder as seguintes questões: o que o programa faz e como os resultados produzidos podem ser verificados.

b) Pequenas *releases* e iteração de uma semana: essas práticas estão em consonância com o planejamento de processo proposto pelo *Scrum*. Com versões do sistema entregues em intervalos menores permite que os pesquisadores possam verificar o progresso de desenvolvimento, avaliar os resultados intermediários, e sugerir modificações para o sistema. A avaliação intermediária é muito importante para os pesquisadores, pois de posse dessas informações podem ser avaliadas as possibilidades ou necessidade de reformulação da pesquisa.

c) Projeto técnico incremental e integração contínua: o *design* técnico de cada funcionalidade é discutida durante o seu desenvolvimento e não nos planejamentos iniciais. A modularidade dos componentes do sistema, que devem funcionar de forma independente, é o objetivo principal. Utilizando essa abordagem as reuniões de planejamento são mais dinâmicas e o projeto de cada funcionalidade leva em consideração o conhecimento adquirido, até então, sobre seu domínio da aplicação. Uma vez que as funcionalidades são validadas, são integradas ao sistema permitindo que o software cresça continuamente ao longo do processo.

d) *Refactoring* de código: o objetivo do *refactoring* é melhorar a qualidade do código. Essas melhorias, através da reorganização ou reformulação do código fonte, podem também aumentar o desempenho do sistema. No desenvolvimento de softwares científicos o *refactoring* é quase sempre necessário. No processo de investigação científica as constantes alterações do código fonte, com o objetivo de atingir o resultados esperados, acabam desorganizando a estrutura do código ou o tornam ilegível. A prática de *refactoring* é então utilizada para simplificar a sua estrutura e tornar o código mais simples e eficiente.

e) Padrão de codificação e código compartilhado: a definição de um padrão de codificação auxilia a leitura do código por aqueles que não o desenvolveram, facilitando as atividades de correção de *bugs* e de *refactoring* de código que são muito comuns no desenvolvimento de softwares científicos. A prática de código compartilhado, onde qualquer codificação é considerada de autoria global, diminui as individualidades e facilita a atuação de qualquer programador sobre qualquer parte do código.

f) Envolvimento dos pesquisadores: o envolvimento dos pesquisadores no desenvolvimento do software resulta em diversos benefícios para o processo. As dúvidas de implementação são solucionadas de forma direta sem intermediários, permitindo uma maior compreensão que se está desenvolvendo por ambas as partes, programadores e pesquisadores. Para tanto, a comunicação oral é priorizada para agilizar o processo e reduzir a burocracia documental. Essa interação reduz o desperdício de tempo e facilita o processo de definição de prioridades para o desenvolvimento do software. Para Segal (2005), a substituição da formalização de documentos pela comunicação oral não é um consenso e algumas questões devem ser observadas. Potenciais problemas podem incluir a redução de conhecimento ao longo do tempo e a saída de pessoas da equipe com consequente perda de informações. Para minimizar tais problemas deve-se, portanto, determinar a documentação que é realmente necessária para se evitar a burocracia excessiva.

Abaixo são descritas duas práticas bastante difundidas pelos seguidores do *Extreme Programming*, que para o desenvolvimento de softwares científicos, apresentam limitações e não podem ser aplicadas da forma como foram descritas originalmente. A programação em pares e o teste antecipado.

a) Programação em pares: como descrito na revisão de literatura, essa prática é a que provoca maior discussão entre Gerentes de Projeto. Enquanto existem afirmações que dizem que destinar dois programadores para um mesma tarefa eleva o custo do projeto em demasia, algumas pesquisas colocam que dois programadores fazendo a mesma tarefa gera um pequeno custo extra que se compensa pela qualidade do código que é produzido. Porém, como nas microempresas os recursos são limitados e os prazos de desenvolvimento são menores decorrentes de projetos de escala reduzida, essa prática, muitas vezes, não é viável.

b) Teste antecipado: para a realização dessa prática existem conjuntos de bibliotecas escritas para as várias linguagens de programação, que auxiliam no processo criação de testes automatizados. Essa técnica consiste na programação de um teste automatizado para cada funcionalidade e a programação dessa deve corresponder a codificação mínima para que passe o teste. Para as funcionalidades que dependem interfaces com o usuário os testes automatizados são mais complexos de serem executados. No desenvolvimento dos modelos ou métodos científicos, as características complexas dos dados de saída dos processamentos impendem a criação de um teste que possa validá-los de forma automatizada. Portanto, essa prática é pouco utilizada no desenvolvimento de softwares científicos.

5. APLICAÇÃO E RESULTADOS

O método de gestão para o processo de desenvolvimento de software científico proposto no capítulo anterior foi aplicado na empresa Animati Computação Aplicada para sua avaliação e validação. Para tanto, foram realizados três estudos de caso em projetos de cunho científico desenvolvidos pela empresa. A seguir são descritos os estudos de caso analisados e na sequencia os resultados observados.

5.1. Estudos de caso

Os projetos analisados foram: um sistema para a resolução de problemas de pesquisadores externos a empresa de aplicação industrial e em laboratórios de pesquisa, um projeto interno no qual foi identificada uma demanda de mercado para o produto/serviço e um sistema que atende as necessidades de pesquisadores parceiros da empresa e seus laboratórios. Os três projetos juntos abrangem todas as características para o desenvolvimento de software descritas na seção 4.3. Nas seções que seguem são descritos os projetos, suas características e aplicação das práticas ágeis do método proposto.

5.1.1. Aplicação industrial

Essa pesquisa tem por objetivo o desenvolvimento de um sistema para a classificação de tábuas de madeira de pinus. Está sendo desenvolvida em conjunto com um laboratório de pesquisa em ciências agronômicas. O objetivo do software é analisar imagens digitais das tábuas para classificar sua qualidade. Recebendo as imagens digitais como entrada de dados, o software deve realizar operações para seu processamento e análise e gerar os resultados de

classificação dos objetos imageados. Os métodos de processamento e análise de imagens empregados nesse sistema abrangem segmentações, operações morfológicas, medições de comprimentos e de área, dentre outros.

A natureza das imagens, a forma como são obtidas, seu tamanho e a definição das características que devem ser extraídas, formam um conjunto de informações que devem ser combinados para se realizar os processamentos e atingir os resultados esperados. Esse tipo de problema enquadra-se no nível intermediário, no qual empregam-se métodos conhecidos e o resultados das combinações desses métodos pode apresentar resultados inesperados. Para tanto, a presença do pesquisadores é fundamental para avaliar a correção e efetividade do resultados intermediários no decorrer do desenvolvimento.

Uma das dificuldades desse projeto consiste na distância geográfica entre os pesquisadores que patrocinam a pesquisa e a equipe de desenvolvimento, São Paulo - SP e Santa Maria - RS, respectivamente. Para minimizar o efeito desse componente na execução desse projeto, um pesquisador interno a empresa foi designado para representar o cientista patrocinador do projeto. O pesquisador interno, além de ter conhecimento técnico para auxiliar na definição dos métodos utilizados para a resolução do problema, mantém contato direto com o patrocinador, obtendo informação pertinentes ao projeto e apresentando os resultados intermediários.

O andamento do projeto ocorre conforme o método proposto, dentro das fases de Planejamento Inicial, Execução e Fechamento do Projeto

a) Planejamento Inicial: como o pesquisador patrocinador da pesquisa não pode estar presente na reunião de planejamento inicial, organizou-se o processo da seguinte forma. O pesquisador interno designado, antes da primeira reunião com a equipe, entrou em contato com o patrocinador via e-mail e telefone para identificar as necessidades do projeto, ou seja, efetuou uma definição de requisitos do sistema de forma simplificada. De posse dessas informações realizou-se uma reunião de planejamento com toda a equipe presente para melhor definição das história de desenvolvimento e cálculo das estimativas para cada funcionalidade. Após essas definições, o pesquisador interno entrou novamente em contato com o patrocinador para identificar as prioridades para desenvolvimento. Ao final do Planejamento Inicial e equipe possuía as descrições das funcionalidades, as estimativas, a ordem de prioridade, os prazos e plano para entrega de versões do sistema.

b) Execução: com as informações do Planejamento Inicial e o número de pessoas

disponíveis, foi possível planejar o primeiro ciclo de desenvolvimento. Com base no cálculo de homens/dia para o ciclo, selecionaram-se as histórias de acordo com sua prioridade para completar a capacidade de desenvolvimento do ciclo. No momento da realização dessa análise, haviam sido finalizados dois ciclos desenvolvimento. Durante esse período realizaram-se reuniões diárias para verificar o andamento do processo e atualização do quadro de tarefas com as informações sobre a evolução do projeto. Nas Figuras 16, 17 e 18 pode-se observar o quadro de tarefas após a primeira reunião de planejamento, três dias antes do fim do ciclo e ao final do primeiro ciclo de desenvolvimento, respectivamente.

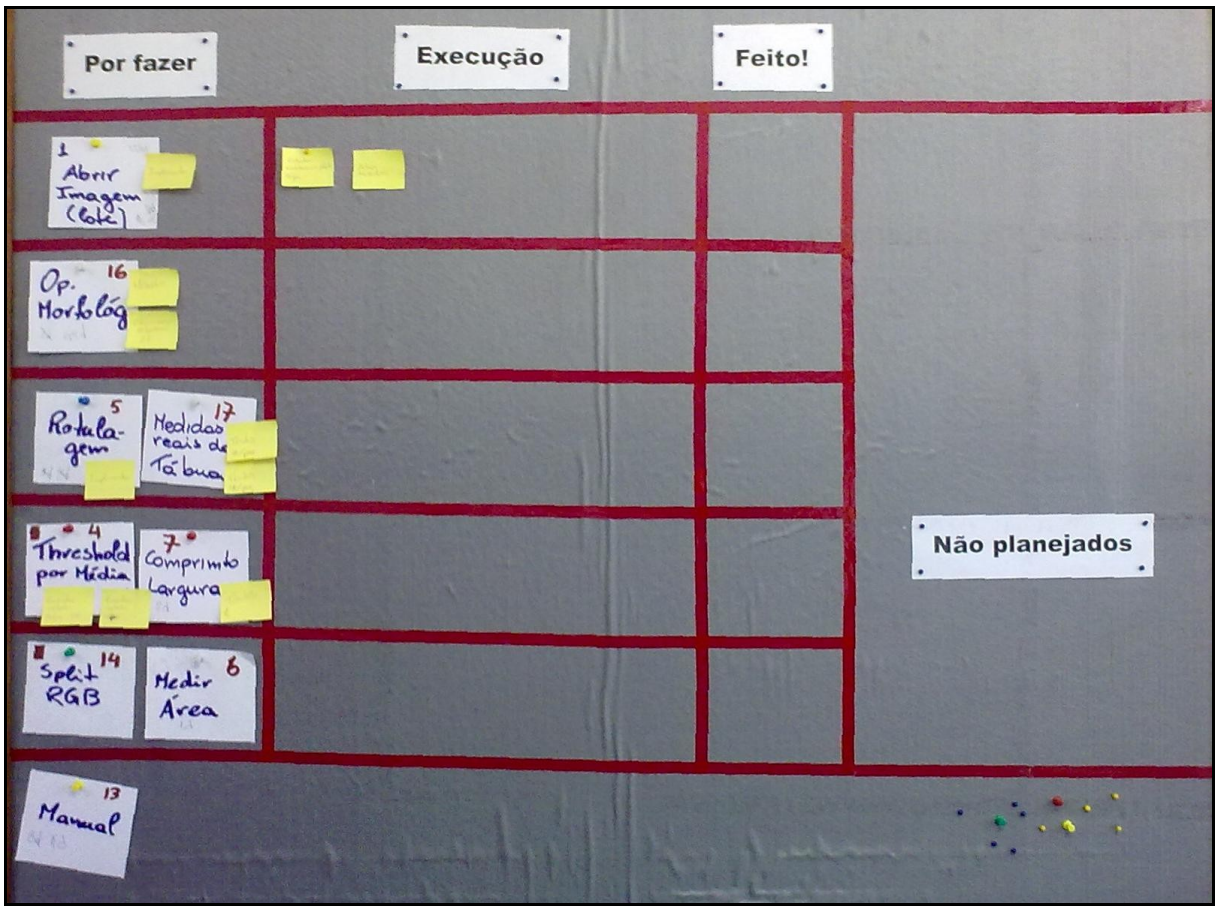

Figura 16: Quadro de Tarefas - primeiro dia de planejamento

Figura 17: Quadro de Tarefas - andamento do processo

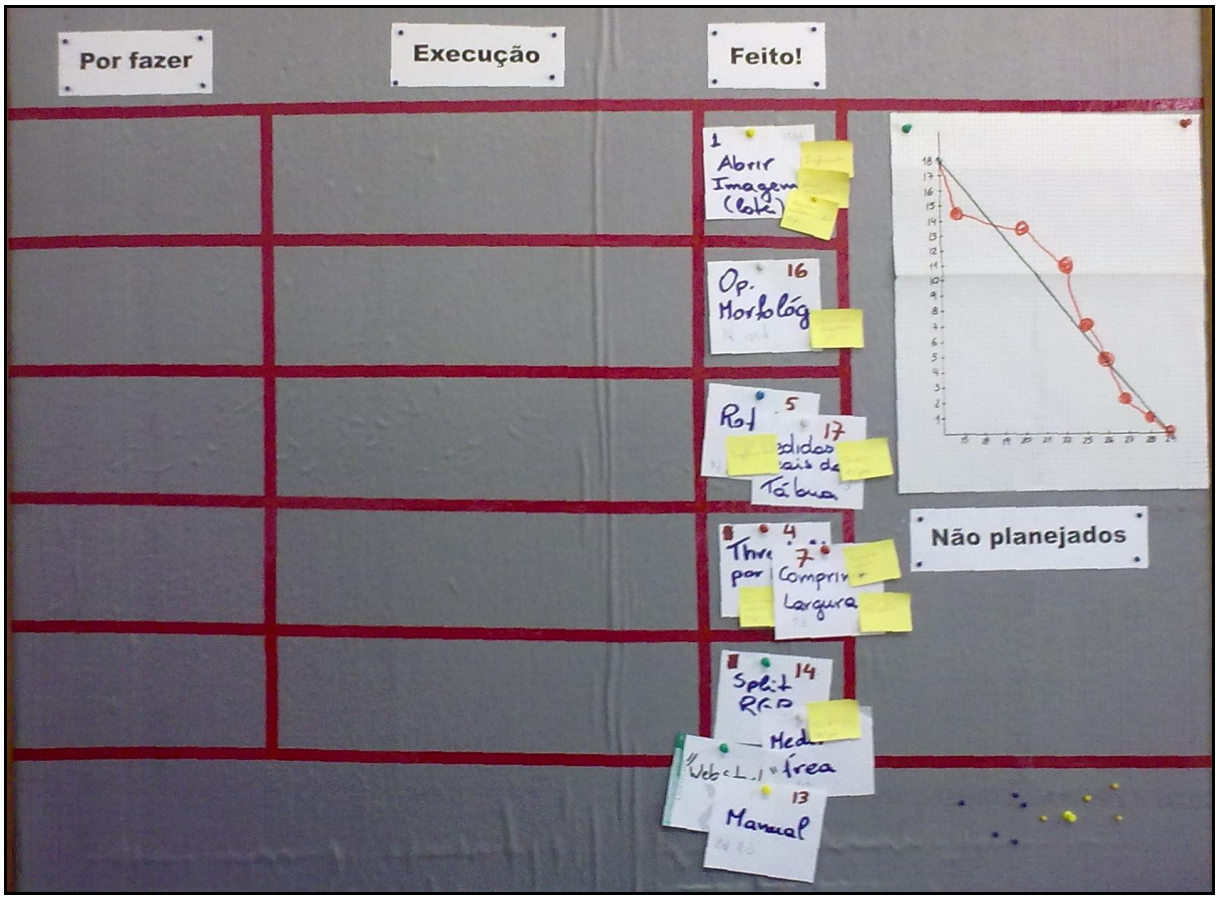

Figura 18: Quadro de Tarefas - fim do ciclo de desenvolvimento

Ao final de cada ciclo, que coincidiu com a entrega de versões do sistema para avaliação do patrocinador pode-se discutir os resultados intermediários obtidos, bem como questões de melhoria, sugestões e alteração de prioridades para o próximo ciclo. A reunião de retrospectiva foi realizada internamente, onde foram levantadas questões para melhorias do processo.

c) Fechamento do Projeto: como a análise do projeto foi efetuada juntamente com sua execução, somente as atividades já realizadas puderam ser analisadas. As atividades de fechamento do projeto não puderam ser verificadas, porém, como seguem, a linha dos fechamentos dos ciclos, acredita-se que sua aplicação será de simples validação.

As práticas no nível de desenvolvimento empregadas nesse projeto de forma mais intensa foram: cenários ou histórias (Figura 19), pequenas *releases*, projeto técnico incremental e integração contínua, *refactoring* de código, padrão de codificação e código compartilhado.

	A	B	C	D	E	F
1	**Lista de Funcionalidades**					
2						
3						
4	**ID**	**Nome**	**Prioridade**	**Estimativa**	**Como demonstrar (validação)**	**Notas**
5	1	Abrir e mostrar imagem (lote)	OK	5	Mostrar a imagem na tela	
6	14	Split RGB	OK	0.5	Mostrar a imagem em tons de cinza	Na verdade para a banda verde do RGB
7	4	Threshold por Média	OK	1	Mostrar imagem binarizada	Fazer sobre a o verde do RGB
8	13	Documentação primeira entrega	OK	2		ver requisitos
9	16	Operações morfológicas (erosão, dilatação) para produzir máscara da tábua	OK	2	Mostrar máscara	
10	5	Rotulagem de imagem binarizada	OK	3	Mostrar resultado em cores ou Outlines e mostrar número de objetos.	Usar método descrito na Monografia
11	6	Medida de área	unido com 8	1	Resultado em tabela	
12	7	Medidas de comprimento e largura	OK	2	Resultado em tabela	Obedecer o sentido da tábua
13	17	Calcular escala e medidas reais da tábua	OK	1	Mostrar em tabela	Dado: comprimento real da tábua é de 2 metros.
14	32	Segmentação dos defeitos (nós)	99	3	segmentar	crescimento de regiões
15	33	Interface: ver mais de uma imagem; zoom; arrumar menus; etc.	95	6		
16	34	Apresentação das medidas dos nós (defeitos sem classificação)	94	1		tabela
17	28	Documentação da segunda entrega	93	2		manual
18	8	Outras medidas morfológicas (shape analyzer) + Área	91	3	Resultado em tabela	Área; diâmetros conforme norma; comprimento conforme norma; circularidade; outros, a definir
19	9	Medidas cromáticas		1	Resultado em tabela	média (de cada banda), mediana, desvio padrão
20	20	Tratamento de erros básico	90			
21	18	Pesquisa para diferenciação entre os tipos de defeitos	89	5		
22	35	Interface: tabelas de resultados	85			
23	22	Internacionalização	60			Inglês / português
24	23	Revisão na documentação do código	60	4		Inglês e português!
25	15	Incluir avisos de licença	50	1		
26	16	Definição e documentação de arquitetura	50			
27	17	Refatoração para arquitetura definitiva	50			Anima2 + plugin

Figura 19: Lista de histórias/funcionalidades

5.1.2. Aplicação para medicina

Esse projeto foi concebido internamente e tem por objetivo o desenvolvimento de um sistema capaz de processar, analisar e armazenar imagens e dados de acordo com os requisitos necessários em clínicas de diagnóstico por imagem, laboratórios e demais instituições em que a análise de imagens é parte do processo produtivo. Analisando o mercado foi verificado o interesse por uma solução que permita a distribuição de laudos com imagens associadas para os médicos solicitantes dos exames via internet. É uma situação comum os pacientes de cidades ao redor do município de Santa Maria se deslocarem a cidade para realizarem seus exames e depois aguardarem pelos correios o resultado. Além disso, funções triviais de um sistemas para gestão das imagens digitais são procuradas por clínicas e hospitais da região.

O projeto foi inscrito no programa Primeira Empresa Inovadora - PRIME do órgão de fomento FINEP (Financiadora de Estudos e Projetos). Esse programa tem por objetivo financiar projetos de inovação tecnológica de empresas nascentes com recursos não reembolsáveis. A Animati recebeu a aprovação em outubro de 2009 e desde então trabalha na organização e execução de sua proposta.

Para tanto, o processo de desenvolvimento foi organizado conforme o método de gestão descrito nesse trabalho e encontra-se em seu primeiro ciclo desenvolvimento. Por se tratar de um projeto interno, cujos pesquisadores da empresa detém boa parte do conhecimento necessário para o desenvolvimento do projeto, a organização do processo produtivo bem como suas atividades relacionadas ficou facilitado. Na Figura 20 pode-se observar o resultado do planejamento do primeiro ciclo de desenvolvimento. Nota-se que as linhas hachuradas representam a funcionalidades que serão desenvolvidas no primeiro ciclo.

	A	B	C	D	E	F
1	**Projeto PRIME – Funcionalidades**					
2						
3	**ID**	**Nome**	**Importância**	**Estimativa**	**Como Demonstrar**	**Notas**
4	1	Pesquisa por exame no banco	100	pronto		
5	2	Cadastro de usuários	90	1	Interface de cadastro, cadastrar e mostrar lista de usuários	
6	3	Autenticação de usuários	85	2	Passar da tela de login para a busca somente se o usuário estiver cadastrado e a senha correta	
7	4	Gerenciamento de permissões	80	1	Dividir usuários em grupos, e poder atribuir permissões diferentes a eles. Executar busca com restrições que se aplicam ao usuário	
8	5	Download de exames	75	3	Baixar e exame para o computador (somente se o usuário tiver permissão)	
9	6	Mostrar tags dicom da primeira imagem	50	2	Exibir todas as tags dicom da primeira imagem do exame em outra aba do navegador	
10	7	Mostrar resultado de pesquisa com imagens	70	3	Interface de resultado de pesquisa com imagens	
11	8	Upload de exames	50	5	Fazer upload de um exame	
12	9	Upload de imagens para integrar exames	50	2	Fazer upload	
13	10	Upload de laudo vinculado a exame	50		Fazer upload e vincular	
14	11	Página Sobre	50			
15	12	Ajuda	50			
16	13	Informação de licença	50			
17	14	Layout bonito	50			layout pela equipe do Portaw, implementação Animati
18	15	Mostrar laudo vinculado ao exame	50		Mostrar o laudo a partir de link no menu do exame, se o usuário estiver autorizado	
19	16	Internacionalizar documentação do código				
20		Internacionalizar interface				
21		Anonimização de exames				

Figura 20: Resultado do planejamento do primeiro ciclo da aplicação para medicina

Devido a alguns detalhes de desenvolvimento e para a identificação das necessidades dos clientes para esse tipo de produto, buscou-se a colaboração de professores da área médica que trabalham diretamente com a área de pesquisa estudada. Também laboratórios, como possíveis clientes, foram procurados para fornecerem informações do ambiente real de aplicação.

Dentro do processo de produção, esses agentes externos atuam diretamente na definição de prioridades das funcionalidades que estão sendo desenvolvidas e na sua validação. Atualmente, no Brasil, não existe regulamentação específica para esse tipo de software médico. No entanto, estão surgindo iniciativas de padronização e regulamentação tramitando dentro dos órgão competentes. Até o momento da realização dessa pesquisa, não foram necessárias alterações no planejamento do processo produtivo para garantir que produto siga os eventuais novos padrões. Porém, como descrito, as modificações no processo são bem vindas, pois possui a flexibilidade para tanto. Se o processo tivesse sido definido conforme metodologias prescritivas, o surgimento de novas regras seriam encaradas como ameaças ao processo, comprometendo o mesmo.

No nível de desenvolvimento, onde são aplicadas as técnicas de Engenharia de Software, exitem outras pesquisas que analisaram a utilização do *Extreme Programming* para o desenvolvimento de softwares médicos (FRUHLING, TYSER & VREEDE, 2005; KANE et

al. 2006; SANDERS & KELLY, 2008). Fruhling, Tyser & Vreede (2005) apontam que o XP é uma metodologia efetiva para desenvolvimento de aplicações para a saúde, pois a rápida prototipação permite aos desenvolvedores e usuários dos sistemas, esclarecerem melhor os requisitos e melhorar a comunicação e o entendimento do que deve ser desenvolvido.

Na aplicação para a área médica investigada na pesquisa de Sanders e Kelly (2008), verificou-se que existe uma distinção entre a versão de pesquisa e a versão distribuída para aplicação no mundo real. O desenvolvimento ocorre em duas etapas. Os pesquisadores exploram as teorias e desenvolvem sem um processo bem formalizado e definido. Um vez que o que o software é considerado pronto uma equipe profissional o reescreve. Na Animati, vem ocorrendo um processo semelhante, assim que um protótipo é apresentado para o possível cliente e validado, esse passa pela prática de *refactoring* de código para que melhorias em sua estrutura de codificação sejam aplicadas.

5.1.3. Aplicação científica/pesquisa

Essa aplicação, chamada Anima, vem sendo desenvolvida em parceria com o Laboratório de Computação Aplicada - LaCA do curso de Ciência de Computação - UFSM e tem como objetivo o desenvolvimento de um ambiente para o processamento e análise de imagens, onde alunos de graduação e mestrado possam desenvolver suas pesquisas. O sistema também é empregado em um laboratório de física onde são analisadas imagens de microscopia para caracterização de minerais.

O desenvolvimento do sistema iniciou em 2005 dentro do LaCA, e seguia uma abordagem mais prescritiva para seu desenvolvimento. Ou seja, conforme práticas das metodologias tradicionais, através das fases de levantamento de requisitos, projeto, implementação, sem levar em consideração as mudanças no ambiente e o surgimento de novos requisitos ao longo do desenvolvimento do sistema. Recentemente, devido ao projeto descrito na seção 5.1.1 o Anima precisou ser reformulado para atender padrões de mercado. Para essa reestruturação as práticas ágeis foram empregadas.

No Planejamento Inicial foram reunidos os envolvidos, e definidas as questões básicas para dar início ao processo de desenvolvimento, os ciclos. O planejamento dos ciclos são efetuados internamente a empresa pois existe o conhecimento e capacidade para definição das prioridades de desenvolvimento. A participação dos pesquisadores parceiros ocorre

novamente nas reuniões de fechamento dos ciclos para validação das funcionalidades desenvolvidas e sugestões para modificações ou novos itens para serem integrados ao sistema.

Esse projeto se diferencia dos demais por não possuir prazo de entrega final, ou seja o sistema está em constante evolução e atualização. Sempre que surgem pontos que devem ser melhorados no software, a equipe se reúne com os pesquisadores colaboradores e define o planejamento para uma ciclo de desenvolvimento que culmina em uma nova versão para o sistema. Os prazos de entrega, e o número de ciclos para o desenvolvimento variam de acordo com a quantidade de funcionalidades que devem ser implementadas.

Além dos métodos ágeis descritos nos estudos de caso anteriores, pode-se destacar a utilização da Programação em Pares. Essa prática foi fundamental para que a transferência de conhecimento fosse realizada de forma mais eficiente. Um membro da equipe que havia participado dos antigos processos de desenvolvimento do sistema, e por consequência conhecia os detalhes de implementação, trabalhou em conjunto com outro programador para sua reformulação. O conhecimento adquirido nos desenvolvimentos anteriores permitiu que a nova versão fosse projetada para evitar erros cometidos e favoreceu a troca de conhecimentos entre os programadores.

5.2. Resultados observados

Conforme analisado no seção 4.1 as metodologias ágeis *Scrum* e *Extreme Programming* funcionam bem quando combinadas. Enquanto o *Scrum* endereça questões relacionadas à organização do processo produtivo, o XP sugere práticas que são empregadas diretamente nas atividades de produção. A natureza complementar das duas metodologias permitiu a definição de um método de gestão mais completo e abrangente sem comprometer a agilidade e adaptabilidade dos processos ágeis.

Na organização e controle do processo, onde as fases descritas na seção 4.4.1 foram utilizadas, pode-se verificar um conjunto de características que permitem sugerir que o método proposto é adequado e atente a maior parte das necessidades do tipo de ambiente estudado. No Quadro 1 são apresentados os prós e contras identificados dentro das fases definidas para a organização do processo de produção.

Fases	Prós	Contras
Planejamento Inicial	• Definição simples da visão do produto; • Reuniões curtas com objetivos bem definidos.	• Geração de prazos sub ou superestimados; • Baixa documentação para o desenvolvimento de sistemas críticos.
Execução	• A equipe aprende ao longo do processo buscando sua melhoria; • A equipe torna-se autônoma e prima pela excelência do processo; • A liderança do processo é distribuída; • O ambiente aberto encoraja o *feedback* constante; • Redução da burocracia; • Permite a inserção de novas práticas ao processo de desenvolvimento; • Adaptatividade e agilidade.	• Parte do princípio que a equipe estará comprometida com o processo; • A delegação de tarefas é dificultada pela liderança distribuída; • Necessidade de envolvimento de todos os participantes no processo; • Funciona bem com produtos novos, mas necessita de adaptações para o processo de manutenção dos já existentes; • Depende da experiência da equipe.
Fechamento do Projeto	• Permite a equipe aprender com os obstáculos vencidos; • Facilita a avaliação do esforço do time com base no seu desempenho;	• Depende da validação do cliente/usuário final.

Quadro 1: Prós e contras das fases definidas para a organização do processo

Práticas	Prós	Contras
Cenários ou histórias	Definição direta do objetivo das funcionalidades do sistema	Omissão de detalhes completos de implementação
Pequenas *releases* e iteração de uma semana	Permite a avaliação constante do progresso do desenvolvimento	Sub ou superestimação dos prazos
Projeto técnico incremental e integração contínua	Questões importantes definidas somente quando necessárias, levando em consideração as informações obtidas ao longo do processo	Detalhes de implementação significativos não previstos podem influenciar o andamento do processo
***Refactoring* de código**	Melhoria contínua da qualidade do código fonte	Custos associados
Padrão de codificação e código compartilhado	Legibilidade do código fonte	Tempo de aprendizagem para a padronização
Envolvimento dos pesquisadores	Facilidade de comunicação e resolução de problemas	Dificuldade de reunião de todos participantes (principalmente externos)
Programação em pares	Combinação de conhecimentos e maior qualidade do código	Custos associados
Teste antecipado	Permite a descoberta de *bugs* no momento em que são produzidos	Não podem ser utilizados em qualquer situação

Quadro 2: Prós e contras das práticas do nível de produção

Assim como no nível organizacional do processo, para o nível de produção onde foram aplicadas as práticas descritas na seção 4.4.2 foram identificadas propriedades que permitiram avaliar suas contribuições para o processo produtivo analisado. No Quadro 2 são apresentados prós e contras dessas práticas dentro do escopo avaliado.

Pode-se perceber que a utilização das duas metodologias de forma isolada, sem a sua combinação, faria com que o processo ficasse incompleto pela carência de definições para todas as atividades de produção inerentes. Se fossem utilizadas somente as atividades definidas pelo *Scrum*, no processo de desenvolvimento faltariam meios para organizar as formas de codificação, testes e controle de qualidade. Por outro lado, se somente as práticas do *Extreme Programming* fossem utilizadas, mesmo que a metodologia aponte alguns meios para gestão do processo, as questões de planejamento, execução e controle seriam comprometidas devido às necessidades do ambiente analisado.

Dessa forma, na aplicação do método observaram-se uma série de pontos que sugerem os benefícios da utilização das metodologias ágeis, tanto para o nível de organização do processo como para o nível de produção. No Quadro 3 esses pontos são relacionados levando-se em consideração os contextos do software científico e da microempresa.

Nível	Contexto	Benefícios
Processo	**Software científico**	• Requisitos emergentes são bem-vindos; • Possibilidades para alteração no planejamento do desenvolvimento; • Aumento da comunicação entre os envolvidos.
	Microempresa	• Redução da burocracia processual; • Agilidade para suportar as mudanças no ambiente decorrentes dos processos de pesquisa; • Ferramentas simplificadas para acompanhamento do processo.
Produção	**Software científico**	• Interação frequente entre pesquisadores e desenvolvedores; • Entregas constantes de versões de software; • Agilidade para efetuar correções ou mudanças nos modelos científicos programados.
	Microempresa	• Melhor qualidade do código decorrente das práticas de *refactoring*, testes e padrões; • Atividades de produção simplificadas; • Foco na criação do software e na entrega de valor para o cliente/pesquisador.

Quadro 3: Benefícios do método proposto em relação aos níveis da organização

Através da delimitação dessa pesquisa também foi possível identificar outros dados qualitativos que argumentam a favor da utilização dos métodos ágeis para a produção de software científica dentro da microempresa. A abordagem proposta demonstrou melhorias nos processos de desenvolvimento da empresa e na interação entre os clientes/pesquisadores e desenvolvedores. As vantagens que destacam-se para os pesquisadores, para a empresa e para o processo de produção são:

a) Vantagens para o cliente/pesquisador:

- Contato frequente com os desenvolvedores. O contato direto e frequente com os programadores do sistema permite com que dúvidas referentes à parte científica sejam solucionadas de forma mais rápida e eficiente.

- Descoberta de novas abordagens. Com os pesquisadores em constante colaboração no processo de desenvolvimento percebe-se um aumento do número de sugestões para a solução de problemas, e consequentemente, surgem novas possibilidades e abordagens.

- Resolução de problemas. A participação dos cientistas ajuda na identificação de possíveis problemas ao longo do desenvolvimento. Um erro corrigido de forma antecipada diminui consideravelmente o retrabalho no sistema em relação ao que seria necessário em uma condição avançada de desenvolvimento.

- Pesquisa e incertezas. De posse das informações parciais que são obtidas por meios de testes ao longo do projeto, os pesquisadores tem a possibilidade de rever cálculos ou métodos científicos que estão sendo utilizados, e dessa forma, maximizar as chances de sucesso na elaboração do sistema.

b) Vantagens para a empresa:

- Contato frequente com os clientes. O contato direto com os clientes permite com que erros de programação ou nos objetivos do produto sejam identificados de forma mais eficiente e rápida, evitando sérios problemas de retrabalho.

- Flexibilidade. A discussão constante entre cliente e desenvolvedores permite com que funcionalidades sejam negociadas, fazendo com que o mais importante seja desenvolvido primeiro.

- Criação de inovação e valor. O processo de criação é beneficiado pela atividade de busca por novas soluções. À medida que novos problemas surgem, novas possibilidades devem ser buscadas para resolver problemas específicos, assim, potencializando a inovação dos produtos.

- Redução da burocracia. Um ponto crucial na adoção de metodologias ágeis diz respeito à redução da burocracia dos processos. Geralmente, o atrelamento a padrões predefinidos em metodologias muito planejadas reduz o escopo e a gama de alternativas a serem buscadas pelos desenvolvedores. A diminuição da burocracia nos processos libera tempo aos desenvolvedores e lhes permite criar alternativas com maior flexibilidade.

c) Vantagens para o processo:

- Qualidade do produto. Ao final do processo, ao se evitarem problemas por falta de comunicação e por surgirem alternativas de forma criativa, encontra-se um produto de maior valor para o cliente. Os testes de validação que ocorrem com a presença do cliente ao final de cada ciclo iterativo de desenvolvimento garantem maior qualidade no produto final.

- Redução de riscos. A negociação e definição das funcionalidades do sistema que serão desenvolvidas em cada ciclo fazem com que os ricos inerentes sejam minimizados. O risco de se ter que refazer parte de um ciclo tem menor impacto do que se tivesse que alterar parte do sistema ao final de todo o processo de desenvolvimento.

- Prazos adaptáveis. A negociação para definição das funcionalidades que serão desenvolvidas em cada ciclo permite que sejam dadas prioridades às tarefas, assim, o que não pode ser desenvolvido em determinado ciclo pode retornar como opção para o próximo (muitas vezes com prioridade maior).

- Flexibilidade. O processo torna-se flexível pelo alto grau de interação entre as partes, a negociação de funcionalidades e a discussão de soluções possíveis garantem elaboração do sistema com as características mais próximas possíveis da visão (que pode modificar-se ao longo do processo) do projeto final.

- Velocidade de resposta aos estímulos externos. Quando surgem imprevistos no decorrer do processo de desenvolvimento, a abordagem ágil e adaptativa tem maiores condições de responder às contingências e fazer com que o processo de produção se reorganize. Com metodologias prescritivas, devida a elevada burocratização, os custos de mudança são muito elevados.

6. CONCLUSÃO

Neste trabalho investigou-se a utilização de metodologias ágeis para o desenvolvimento de software científico e a aplicação destas técnicas em projetos reais para verificação de sua eficiência. Pode-se observar que no contexto delimitado pelo universo da pesquisa, a abordagem ágil adapta-se de forma satisfatória à realidade do desenvolvimento de software científico por microempresas.

A abordagem ágil possui características diferentes das metodologias planejadas ou prescritivas. Questões como comunicação constante com os *stakeholders*, redução da burocracia processual, flexibilidade e adaptação no desenvolvimento, fazem com que processo de desenvolvimento de software científico seja ainda mais experimental e criativo. Favorecendo, dessa maneira, a inovação tecnológica e geração de valor para os clientes.

Pequenas empresas ou *startups* têm necessidades diferentes de grandes organizações produtoras de software. O excesso de burocracia ou demasiada especificação de processos pode reduzir a capacidade da empresa produzir com agilidade. Conforme analisado, a perda de foco no desenvolvimento e aperfeiçoamento do produto pode representar o fracasso no andamento de qualquer projeto. A utilização de uma metodologia ou processo de produção adaptado às realidades enfrentadas pela pequena organização é de fundamental importância para sua sobrevivência e crescimento.

Com a combinação destes três componentes, software científico, desenvolvimento ágil e microempresa, foi presumido um enfoque de pesquisa, ainda pouco abordado pela literatura. Com base nessas questões surgiram as hipóteses:

a) HIPÓTESE 1: A adaptação (*Tailoring*) de metodologias ágeis para utilização em microempresas é plausível;

b) HIPÓTESE 2: É possível utilizar-se de metodologias ágeis adaptadas para o desenvolvimento de softwares científicos.

Para a confirmação dessas hipóteses foi proposto um método de gestão para o processo de desenvolvimento de software adaptado através das metodologias ágeis *Scrum* e XP, e direcionado para a produção de aplicações em ambientes de pesquisa científica dentro de microempresa. O método proposto foi aplicado dentro da empresa Animati Computação Aplicada em projetos reais da organização para sua validação. Como resultado, obtiveram-se dados qualitativos que apontaram a melhoria dos processos de desenvolvimento de software da empresa, e como consequência a confirmação positiva das hipóteses. Assim, conclui-se que o método de gestão apresentado pode ser utilizado com sucesso no processo de desenvolvimento de software científico, trazendo benefícios significativos para o mesmo.

Porém, devido ao delineamento qualitativo do estudo, não se pode sugerir que as afirmações apresentadas sejam completamente conclusivas. Algumas considerações poderiam ser mais exploradas e mesmo conclusões adicionais poderiam ser feitas dentro de uma maior amostragem de projetos analisados. Também deve-se ressaltar, que não se pode generalizar o método proposto para aplicação em qualquer tipo de microempresa que desenvolve software. O foco no desenvolvimento de software científico e as características do tipo de empresa analisada reduziu sensivelmente o universo de pesquisa desse trabalho.

Como trabalhos futuros pretende-se analisar a compatibilidade de modelos de maturidade e melhores práticas para o desenvolvimento de software, definidos por modelos como CMMI, PMBOK, ITIL, MPS.BR, com a utilização de metodologias ágeis para desenvolvimento de software científico. Tendo como objetivo a certificação da empresa por modelos de qualidade. Exitem estudos que se direcionam a esse objetivo, como por exemplo, a utilização das práticas do PMBOK juntamente com metodologias ágeis (SLIGER, 2006; GRIFFITHS, 2004) e a utilização de metodologias ágeis para atingir níveis de qualidade para para produção de software MPS.BR (OLIVEIRA, GUIMARÃES & ARAÚJO, 2007).

Outra possibilidade é a análise de ferramentas computacionais para o acompanhamento das atividades do processo de desenvolvimento. Embora as metodologias preguem a desburocratização, e apresentam alternativas simples como o quadro de tarefas, existem questões que precisam ser melhor analisadas. Um exemplo é a possibilidade de acompanhamento do andamento do processo por pessoas externas a empresa, ou esquipes separadas geograficamente.

REFERÊNCIAS

ACKROYD, K. et al. **Scientific software development at a research facility.** IEEE Software, 2008

ANGKASAPUTRA, N.; PFAHL, D. **Making software process simulation modeling agile and pattern-based.** The 5th International Workshop on Software Process Simulation and Modeling, 2004.

ARMOUR, P. **The laws of software process: a new model for the production and management of software.** Auerbach, 2003.

BASILI, V.; ROMBACH, H. **Tailoring the software process to projects and enviroments.** ICSE, 1999.

BAXTER, S. et al. **Scientific software development is not an oxymoron.** PLoS Computational Biology, 2006.

BECK, K. et al. **Manifesto for agile software development.** 2001. Disponível em http://www.agilemanifesto.org, acessado em abril de 2009.

BECK, K. **Programação extrema explicada.** Bookman, 1999.

BERNI, J. C. A.; DORNELLAS, M. C.; FERREIRA, T. K. **Produção de software científico através de metodologias ágeis na microempresa.** Anais do XXIX Encontro Nacional de Engenharia de Produção - ENEGEP, 2009.

BOEHM, B. **A spiral model of software development and enhancement.** Proceedings of International Workshop on Software Process and Software Environments, 1985.

BROOKS, F. **No silver bullet: essence and accidents of software engineering.** IEEE Computer, Apr. 1987.

CAMARA, F. **Um cardápio de metodologias ágeis.** Imasters, 2007. Disponível em http://imasters.uol.com.br/artigo/7396/gerencia/um_cardapio_de_metodologias_ageis/. Acessado em outubro de 2009.

CRABTREE, C. et al. **An empirical characterization of scientific software development projects according to the Boehm and Turner model: a progress report.** Workshop on Software Engineering for Computational Science and Engineering - ICSE 2009, 2009.

CONFORTO, E.; AMARAL, D. **Escritório de projetos e gerenciamento ágil: um novo enfoque para a estrutura de apoio à gestão de projeto ágeis.** Anais do Encontro Nacional de Engenharia de Produção - ENEGEP, 2007.

CURTIS, B.; KELLNER, M.; OVER, J. **Process modeling.** Communications of the ACM, 1992.

DEMARCO T. **Software Engineering: an idea whose time has come and gone.** IEEE Software. 2009

DRAPPA, A.; LUDEWIG, J. **Quantitative modeling for the interactive simulation of software projects.** Journal of Systems and Software, 1999.

FRANCO, E. **Aplicando a gestão ágil de projetos para o desenvolvimento de novos produtos na indústria de software.** Anais do Encontro Nacional de Engenharia de Produção – ENEGEP, 2006.

FRUHLING, A.; TYSER, K.; VREEDE, G. **Experiences with extreme programming in telehealth: developing and implementing a biosecurity health care application.** Proceedings of the 38th Hawaii International Conference on System Sciences, 2005.

GIL, A. **Como elaborar projetos de pesquisa.** Editora Atlas, 1988.

GINSBERG, M.; QUINN, L. P**rocess tailoring and the software capability maturity model.** Software Engineering Institute - SEI, 1995.

GRIFFITHS, M. **Using agile alongside the PMBOK.** PMI Global Congress Proceedings, 2004.

HABRA, N. et al. **Initiating software process improvement in very small enterprises: experience with a light assessment tool.** ACM - Information and Software Technology, 2008.

HIGHSMITH, J. **Agile project management: creating innovative products.** Addison-Wesley, 2004.

KANE, D. et al. **Agile methods in biomedical software development: a multi-site experience report.** BMC Bioinformatics, 2006.

KEENAN, F. **Agile process tailoring and problem analysis (APTLY).** Proceedings of the 26th International Conference on Software Engineering, 2004.

KELLY, D.; SANDERS, R. A**ssessing the quality of scientific software.** First International Workshop on Software Engineering for Computational Science & Engineering, 2008.

KIRK, D. **A flexible software process model.** Proceedings of the 26th International Conference on Software Engineering, 2004.

KNIBERG, H. **Scrum e XP direto das trincheiras: como nós fazemos Scrum.** InfoQ, 2007.

LAKEY, P. **A hybrid software process simulation model for project management.** ProSim 03, 2003.

LETONDAL, C.; ZDUN, U. **Anticipating scientific software evolution as a combined technological and design approach.** Proceedings of Second International Workshop on Unanticipated Software Evolution, USE 2003, 2003.

MACCORMACK, A. **Product development practices that works: how internet companies build software.** MIT Sloan Management Review, p. 75-84, 2001.

MARTINS, J. **Técnicas para gerenciamento de projetos de software.** Brasport, 2007.

MCT - MINISTÉRIO DA CIÊNCIA E TECNOLOGIA. **Qualidade e produtividade no setor de software brasileiro: resultados da pesquisa 2005.** Disponível em: http://www.mct.gov.br/index.php/content/view/3476.html, acesso em dezembro de 2009.

MOHAMED S. **Agile tailoring tool (ATT): a project specific agile method.** Advance Computing Conference - IACC 2009. IEEE International, 2009.

O'CONNOR, R.; COLEMAN, G. **An investigation of barriers to the adoption of software process best practice models.** Australasian Conference on Information Systems, 2007.

OLIVEIRA, A.; GUIMARÃES, F; ARAÚJO, I. **Utilizando metodologias ágeis para atingir MPS.BR nível F na Powerlogic.** Revista Visão Ágil, ano II, edição 1, 2007.

PEDREIRA, O. **A systematic review of software process tailoring.** ACM SIGSOFT Software Engineering Notes, 2007.

PRESSMAN, R. **Engenharia de Software.** McGraw-Hill, 2006.

RAFFO, D.; HARRISON, W. **Combining process feedback with discrete event simulation models to suport software.** FEAST 2000, 2000.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process.** Version 2001.3, CD-ROM, Rational Software, 2001.

SANDERS, R.; KELLY, K. **Dealing with risk in scientific software development.** IEEE Software, 2008.

SCHWABER, K. **Scrum development process.** OOPSLA'95 Conference, 1985.

SCHWARTZ, J. **Construction of software, problems and practicalities.** Addison-Wesley, 1975.

SEGAL, J. **When software engineers met research scientists: a case study.** Empirical Software Engineering, 2005.

SEGAL, J.; MORRIS, C. **Developing scientific software.** IEEE Software, Jul/Aug. 2008.

SEGAL, J.; MORRIS, C. **Guest editors' introduction: developing scientific software, part 2.** IEEE Software, Jan/Feb. 2009.

SELLTIZ, C. et al. **Métodos de pesquisa nas relações sociais.** Editora Herder, 1967.

SEBRAE - SERVIÇO BRASILEIRO DE APOIO ÀS MICRO E PEQUENAS EMPRESAS. **Fatores condicionantes e taxa de mortalidade de empresas no Brasil.** Relatório de Pesquisa. Brasília, 2004.

SHORE, J.; WARDEN, S. **A arte de desenvolvimento ágil.** Alta Books, 2008.

SIQUEIRA, H. **Mapeamento das práticas de Scrum nas áreas de processo do CMMI e uma proposta para sua aderência.** Universidade Federal de Pernambuco, Trabalho de Graduação em Ciência da Computação, 2007.

SLIGER, M. **Relating PMBOK practices to agile practices.** StickyMinds.Com, 2006. Disponível em http://www.stickyminds.com/s.asp?F=S10365_COL_2, acessado em julho de 2009.

SOMMERVILLE, I. **Software Engineering.** Addison-Wesley, 5th. Ed., 1995.

SOMMERVILLE, I. **Engenharia de Software.** Pearson/Addison-Wesley, 8ª Ed., 2007.

STANDISH GROUP. **The chaos report.** The Standish Group, Inc, 2004.

STORRLE, H. **Making agile processes scalable.** ProSim 03, 2003.

SUIKKI, R. et al. **Project management competence development framework in turbulent business environment.** Elsevier, 2006.

TAYLOR, P et al. **Preparing small software companies for tailored agile method adoption: minimally intrusive risk assessment.** Software Process: Improvement and Practice, 2007.

WEGNER P. **Interactive foundations of object-based programming.** IEEE Computer, 1995.

WILSON G. **Software carpentry: getting scientists to write better code by making them more productiv.** Computing in Science & Engineering, 2006.

WOOD, W.; KLEB, W. **Exploring XP for scientific research.** IEEE Software, 2003.

ZIEMER, S. **Managing change in software development.** Central and East European Conference on Software Engineering Techniques, 2007.

ZIV, H. **The uncertainty principle in software engineering.** Proceedings of ICSE'97, 1997.