Universidade Federal de Lavras

CRISTIANE DOS SANTOS COELHO

RELATO DE EXPERIÊNCIA NA IMPLANTAÇÃO DE UM MÉTODO ÁGIL EM UMA EQUIPE DE DESENVOLVIMENTO DE SOFTWARE

LAVRAS-MG

2011

CRISTIANE DOS SANTOS COELHO

RELATO DE EXPERIÊNCIA NA IMPLANTAÇÃO DE UM MÉTODO ÁGIL EM UMA EQUIPE DE DESENVOLVIMENTO DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras para a obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração:
Engenharia de Software.

Orientador:

Prof.º Paulo Henrique de Souza Bermejo, Dr.

Co-orientador:

Prof.º André Luiz Zambalde, Dr.

LAVRAS
MINAS GERAIS – BRASIL
2011

CRISTIANE DOS SANTOS COELHO

RELATO DE EXPERIÊNCIA NA IMPLANTAÇÃO DE UM MÉTODO ÁGIL EM UMA EQUIPE DE DESENVOLVIMENTO DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras para a obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em 21 de Novembro de 2011

Prof.: Antônio Maria Pereira de Resende, Dr. – DCC UFLA

Prof.: Heitor Augustus Xavier Costa, Dr. – DCC UFLA

Prof.º: Paulo Henrique de Souza Bermejo, Dr. – DCC UFLA
(Orientador)

Prof.º André Luiz Zambalde, Dr. –DCC UFLA
(Co-Orientador)

LAVRAS
MINAS GERAIS-BRASIL

Dedico este trabalho aos meus pais José Geraldo Muniz Coelho e Maria Eni dos Santos Coelho, ambos in memoriam, aos meus irmãos Thiago, Tatiane, Thiélio e Bárbara e as minhas sobrinhas Tatielly e Ticielly.

AGRADECIMENTOS

"*Se não houver frutos, valeu a beleza das flores, se não houver flores, valeu a sombra das folhas, se não houver folhas valeu a intenção das sementes*". (Henfil)

Agradeço a Deus pelo o dom da vida e por me guiar nos caminhos difíceis.

Aos meus pais, José Geraldo Muniz Coelho e Maria Eni dos Santos Coelho, que mesmo longe sempre estiveram presentes em meus pensamentos e orações. Obrigada pelo os ensinamentos, coragem, carinho e todos os momentos especiais vividos e eternizados durante permanência física ao meu lado.

Aos meus irmãos, Thiago, Tatiane, Thiélio e Bárbara, pelo amor, respeito, carinho e incentivo.

À minha família, tias e tios, primos e primas, sobrinhas Tatielly e Ticielly, cunhados (as) Eilton e Letícia, pelo apoio e carinho.

Aos meus amigos, Cláudia Dias e família, Marcelo Rufato e família, Natália Bifano, Yara Güim, Aline Antunes, Karine Simões, Samuel e todos que, direta e indiretamente, participaram de mais uma conquista em minha vida.

A toda equipe do LabGTI, Adriano, José Henrique, Guilherme, Danilo, Carla, Samara e André.

Agradeço, também, aos meus orientadores Paulo Henrique de Souza Bermejo e André Luiz Zambalde.

Aos avaliadores Antônio Maria Pereira de Resende e Heitor Augustus Xavier Costa por aceitarem o convite para compor a banca avaliadora e demais professores do DCC/UFLA.

Enfim, a todos que, de algum modo, participaram de mais um capítulo de minha história, os meus sinceros agradecimentos.

RESUMO

Há diversos relatos de que os métodos ágeis proporcionam melhorias relacionadas ao prazo, escopo e qualidade dos produtos e diversas empresas têm migrado para o uso dessas práticas. Entretanto, estudos sobre como alcançar essas melhorias são, ainda, escassos ou inexistentes. Considerando a importância de melhorias no desenvolvimento de software, e o crescente uso das práticas ágeis, em especial a Scrum, este trabalho tem como objetivo verificar e analisar como as práticas ágeis estão sendo tratadas por uma equipe de desenvolvimento que utiliza o método ágil Scrum. Para efetivação deste trabalho, foi realizado um estudo de caso exploratório, em uma organização pública, fundamentado em referências bibliográfica e documental, utilizando observação participante para coleta de dados. Conclui-se que para superar as limitações do método Scrum, práticas comuns a outras metodologias, ágeis ou não, podem ser incorporadas ao processo Scrum, tais como: programação em pares, integração contínua, pequenas entregas, triagem de erros, refatoração, propriedade coletiva do código, build de 10 minutos, testes unitários e design incremental.

Palavras chaves: Metodologias ágeis. Práticas de desenvolvimento de software. Scrum.

ABSTRACT

Agile methods provided improvements related to the delivery time, scope and quality of products and several companies have migrated to the use of these practices. However, studies on how to achieve these improvements are still scarce or nonexistent. Considering the importance of improvements in software development, and the increasing use of agile practices, particularly Scrum, this study aims to verify and analyze how agile practices are being treated by a development team using the Scrum agile method. For realization of this work, we performed an exploratory case study in a public organization, based on documentary and bibliographical references, using participant observation to collect data. It is concluded that, to overcome the limitations of the method Scrum practices common to other methodologies, agile or not, can be incorporated into the Scrum process, such as pair programming, continuous integration, small deliveries, sorting errors, refactoring, property collective code, build 10 minutes, unit testing and incremental design.

Keywords: Agile methodologies. Software development practices. Scrum.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Com o avanço das tecnologias, o uso de software está cada vez mais presente nas atividades diárias de pessoas e empresas, exercendo papel importante em diferentes aspectos do cotidiano e permitindo realizar tarefas de maneira mais rápida e eficiente (KATSURAYAMA, 2008).

Para as organizações, os softwares são ferramentas fundamentais que proporcionam uma vantagem competitiva, onde a qualidade é considerada como um elemento essencial no sucesso dos negócios empresariais. Diante disso, as exigências e necessidades do mercado aumentaram, motivando as empresas desenvolvedoras de software a se preocupar intensivamente na melhoria de seus processos e produtos.

Satisfazer as expectativas de qualidade, tanto de uso quanto de produção, é um dos grandes desafios enfrentados pelas empresas desenvolvedoras de software. Entretanto, frequentemente, muitas ainda lidam com problemas relacionados a falhas no funcionamento do produto, atrasos nas entregas e elevados custos de manutenção.

Em resposta a este cenário, diversas metodologias surgiram para sistematizar e aprimorar o desenvolvimento de software. Os autores as classificam em duas modalidades: Tradicionais - as quais enfatizam a documentação de cada etapa do desenvolvimento; e Ágeis - consideradas um novo paradigma de desenvolvimento de Software, buscando focar menos em formalidades e privilegiando resposta rápida às mudanças de requisitos e ambiente (SCHWABER; BEEDLE, 2002; KOSCIANSKI; SOARES, 2007; BECK *et al*., 2011).

As metodologias ágeis utilizam um conjunto de princípios e práticas para concepção de produtos que atendam as expectativas de todos os envolvidos no projeto.

Uma maneira de garantir a satisfação tanto do cliente quanto da equipe é a adoção de ferramentas, técnicas e modelos, os quais são utilizados

por diversas empresas com o objetivo de melhorar os processos organizacionais e gerenciais no desenvolvimento de software.

Na literatura sobre métodos ágeis, tem-se que estes apresentam práticas simples e objetivas para superar os desafios do desenvolvimento de sistemas, proporcionando melhorias nos processos e nos produtos. Entretanto há poucos relatos de como essas melhorias são alcançadas. Neste sentido, estudos sobre as práticas dos métodos ágeis e seus benefícios são de fundamental importância para as organizações e para as diversas áreas de conhecimento da engenharia de software.

1.1 Problematização e questão de pesquisa

Um dos métodos ágeis que tem se destacado no mercado é o Scrum, a qual apresenta uma abordagem sustentada pela experiência e pela observação, aplicando alguns conceitos da teoria de gerenciamento e controle de processos (KOSCIANSKI; SOARES, 2007).

A eficiência e sucesso das metodologias ágeis têm se confirmado em alguns casos. Empresas que adotaram o Scrum, por exemplo, obtiveram melhorias em termos de gerência, satisfação do cliente e comunicação (Dingsoyr *et al*., 2006). Em contra partida, outras métodos, como *Extreme Programming* (XP) e *Test Driven Development* (TDD), têm demonstrado pouca adesão às práticas de controle e gerenciamento de software.

Considerando a importância e o destaque que o método ágil Scrum tem alcançado, julga-se pertinente a avaliação da compatibilidade, análise e tratamento das práticas Scrum.

Neste sentido, alguns dos problemas abordados nesta pesquisa são os seguintes: “Quais são os benefícios do uso das metodologias ágeis? O que é feito no desenvolvimento ágil para se alcançar as melhorias? e Quais são as melhorias proporcionadas pelas práticas Scrum?”.

1.2 Objetivos

Este trabalho teve como objetivo geral a verificação e análise de como as práticas ágeis estão sendo tratadas por uma equipe de desenvolvimento que utiliza Scrum, ou seja, quais as práticas ágeis efetivas na equipe de desenvolvimento.

Visando o objetivo geral deste trabalho, os seguintes objetivos específicos foram alcançados:

1) Identificação das práticas de desenvolvimento de software nas metodologias ágeis.
2) Verificação quais das práticas do Método ágil Scrum utilizadas na equipe em estudo.
3) Identificação dos benefícios do uso da agilidade no projeto em estudo.
4) Proposta de melhorias ao processo de desenvolvimento no contexto projeto em estudo.

1.3 Estrutura do trabalho

O capítulo 1 introduz este trabalho com a apresentação dos conceitos e motivação, em seguida apresenta o problema e os objetivos da pesquisa.

No capítulo 2 tem-se o referencial teórico com os conceitos de metodologias ágeis, métodos ágeis e trabalhados relacionados sobre as práticas ágeis.

O capítulo 3 descreve os fundamentos e procedimentos da metodologia de pesquisa utilizada durante o desenvolvimento deste trabalho.

No capítulo 4, tem-se a apresentação dos resultados e discussão, descrevendo o objeto de estudo, a equipe e o processo de desenvolvimento do ProManager, as práticas Scrum executadas no projeto, práticas incorporadas ao processo Scrum, problemas encontrados e proposta de melhorias.

O capítulo 5 apresenta conclusão e considerações para trabalhos futuros. Finalizando, têm-se as referências.

2 REFERENCIAL TEÓRICO

Em um mercado globalizado e altamente competitivo, como é o caso do software, a busca pela melhoria dos processos e dos produtos é cada vez mais frequente. O interesse contínuo por melhorias de processos e métodos de desenvolvimento se insere nas atividades diárias das organizações.

Neste capítulo são descritos conceitos relacionados à: metodologias ágeis e seus métodos de desenvolvimento, bem como trabalhos relacionados ao tema da pesquisa.

2.1 Metodologias Ágeis

As metodologias de desenvolvimento de sistemas contribuem de forma eficiente e eficaz para obtenção da qualidade do produto.

Segundo Beck *et al.* (2011), as metodologias ágeis possuem uma abordagem mais simples e objetiva que as metodologias tradicionais, com foco em menos documentação, privilegiando respostas rápidas as mudanças de requisitos e uma ativa colaboração do cliente, além de permitir a minimização dos riscos de desenvolvimento.

As metodologias ágeis surgiram em resposta às deficiências percebidas no desenvolvimento tradicional, oferecendo uma estrutura para desenvolvimento de software altamente colaborativo. São projetadas especificamente para facilitar a comunicação, a colaboração e a coordenação dentro de um ambiente dinâmico, com foco em atividades que contribuem diretamente para o objetivo final do projeto.

Algumas práticas, tais como salas compartilhadas para estimular a comunicação informal e os ciclos de *feedback* com participação da equipe e do usuário durante todo o desenvolvimento do projeto, permitem que o

produto de software seja desenvolvido de forma incremental em parceria com o cliente. Dessa forma, com essa característica de proximidade do usuário, é possível identificar e definir as reais necessidades do cliente e do produto.

O uso das metodologias ágeis impulsionou-se em fevereiro de 2001 quando 17 especialistas em desenvolvimento de software apresentaram e discutiram os métodos: *Extreme Programing* (XP), *Scrum*, *Dynamic Systems Development Method* (DSDM), *Crystal*, entre outros. O resultado foi o estabelecimento do "Manifesto Ágil", apresentando ideias e valores específicos como (Beck *et al*., 2011):

- Valorizar os indivíduos e interações mais que os processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Respostas às mudanças mais que seguir um plano.

A proposta do "Manifesto Ágil" é trabalhar com prioridades, executando tarefas de forma mais simples e objetiva, sem deixar de analisar as características do desenvolvimento tradicionais.

Durante a reunião, intitulada "Manifesto Ágil", foram estabelecidos os doze princípios a serem considerados durante o desenvolvimento de software (Beck *et al*., 2011):

1) A prioridade mais alta é satisfazer o cliente por meio de entrega pronta e contínua de software de valor;
2) Acolher mudanças nos requisitos, mesmo no final do desenvolvimento. Processos ágeis valorizam a mudança para vantagem competitiva do cliente;
3) Entrega software funcionando com frequência, de algumas semanas a alguns meses, preferencialmente usando uma escala de tempo menor;

4) O pessoal do negócio e os desenvolvedores devem trabalhar juntos diariamente ao longo do projeto;
5) Construir projetos em volta de indivíduos motivados. Dê a eles o ambiente e o apoio que necessitam e confie que eles vão fazer o serviço;
6) O método mais eficiente e efetivo para levar informação para uma equipe de desenvolvimento é a conversa face a face;
7) Software funcionando é a principal medida de progresso;
8) Processos ágeis promovem desenvolvimento sustentável;
9) Os patrocinadores, desenvolvedores e usuários devem poder manter um ritmo constante indefinidamente;
10) Simplicidade, arte de maximizar a quantidade de trabalho não realizada, é essencial;
11) As melhores arquiteturas, requisitos e projetos surgem de equipes auto organizadas;
12) Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, depois sintoniza e ajusta seu comportamento de acordo com a reflexão;

O objetivo do desenvolvimento ágil é aumentar a capacidade de reagir e responder mudanças de requisitos, de ambiente, de clientes e necessidades tecnológicas em todos os níveis organizacionais (ABRAHAMSSON *et al*., 2003). Para isso, diversos métodos ágeis são utilizados como *Extreme Programing* (XP), *Scrum*, *Dynamic Systems Development Method* (DSDM), *Crystal*,*Test Driven Development* (TDD), *Adaptive Software Development* (ASD), *Feature Driven Development* (FDD) e *Learn Development*.

Segundo Highsmith (2002), um método ágil deve ser flexível o suficiente para permitir adaptações de seus princípios e práticas, conforme a necessidade do projeto e da equipe. Além disso, deve promover um processo altamente colaborativo e disciplinado que incentiva o trabalho em equipe,

com entregas rápidas de software de alta qualidade, e uma abordagem que alinha o desenvolvimento com as necessidades do cliente e os objetivos da empresa (HIGHSMITH, 2002; CORAM; BOHNER, 2005).

Os diversos métodos ágeis de desenvolvimentos apresentado ao longo dos anos apresentam características especificas, mas sempre apoiados nos valores e princípios da abordagem ágil.

Neste contexto, software são implementados em ciclos de desenvolvimento que geralmente duram entre uma a quatro semanas. Cada iteração envolve uma equipe de trabalho e inclui planejamento, análise de requisitos, design, codificação, testes unitários e homologação. com o objetivo de ter uma versão disponível no final de cada iteração (AMBLER, 2005).

Com a crescente adoção de métodos ágeis, Huo (2004) afirma que os gerentes de projeto necessitam cada vez mais entender a aplicabilidade desses métodos aos seus projetos e os fatores que impulsionam as características de desempenho do projeto.

2.2 Métodos Ágeis

Dos diversos métodos apresentados ao longo dos anos, os mais utilizados no mercado são o Scrum, o qual propõe uma alternativa para gerenciamento e controle de todo projeto, *Extreme Programming* (XP) e *Test Driven Development* (TDD), ambos com práticas para o desenvolvimento de sistemas.

A seguir são descritos os principais métodos ágeis encontados na literatura e utilizados pelas empresas desenvolvedoras de soltwares.

2.2.1 Scrum

Arquitetada no modelo de gestão da indústria automobilística, o Scrum tem se destacado entre as empresas ao longo dos anos (Schwaber, 2004). Isso ocorre devido suas características de gerenciamento e controle de projetos que podem ser utilizada em diferentes contextos.

Scrum foi inspirada na estratégia de reinício do jogo, comumente utilizada em jogo de Rúgbi. Foi apresentada nos anos 90 por Jeff Sutherland, John Scumniotales e Jeff McKenna e posteriormente formalizada por Ken Schwaber e Mike Beedle, que, em 2002, fundaram a Scrum *Alliance*, juntamente com a Agile *Alliance*.

Scrum *Alliance* é uma organização sem fins lucrativos, criada para compartilhar informações sobre o método. Sua missão é aumentar a consciência e compreensão dos processos Scrum, fornecendo recursos para pessoas e organizações e promovendo melhorias necessárias para obter sucesso no uso de suas práticas (SCRUM ALLIANCE, 2011).

O processo Scrum é dividdo em ciclos iterativos denominados *Sprints*, onde, iniciamente, a equipe de desenvolvimento se reúne com o cliente, ou proprietário do produto, objetivando priorizar o trabalho a ser feito, selecionando as tarefas e os responsáveis.

Durante o ciclo, *Sprint*, a equipe executa breves reuniões diárias para discutir o que foi feito, o que não foi realizado e quais foram as dificuldades e limitações encontradas. No final de cada *Sprint*, a equipe oferece um incremento do produto potencialmente entregável.

O método de desenvolvimento Scrum é constituído de papéis, formalidades e artefatos, apresentados no Quadro 1 (SCHWABER; BEEDLE, 2002; SCHWABER, 2004).

Papéis	
Product Owner	Representante de todos os interessados do projeto e responsável pela gestão do *Product Backlog*, de modo a maximizar o valor do projeto.
Team	Equipe responsável pelo desenvolvimento do projeto. Incluem desenvolvedores, analistas de negócio, testadores. O time é auto-organizado e possui autonomia para tomar decisões com foco nos objetivos estabelecidos.
Scrum Master	Tem a missão de garantir o uso do Scrum, bem como a aplicação correta das práticas. Age como facilitador, intermediando negociações entre o *Product Owner* e a equipe no processo de desenvolvimento do produto.
Cliente	Cliente do projeto. Tem participação ativa na definição das funcionalidades e prioridades. É considerado como membro da equipe.
Formalidades	
Sprint Planning	Reunião de planejamento em que o *Product Owner* apresenta a lista de prioridades, *Product Backlog*, para a equipe.
Daily Meeting	Reunião diária de curto espaço de tempo durante o qual os membros da equipe sincronizam o seu trabalho e do progresso, relatando quaisquer impedimentos ao *Scrum Master*.
Sprint Review	Reunião de balanço de todo trabalho realizado durante a *Sprint*.

Sprint Retrospective	Reunião de retrospectiva com duração fixa de 3 horas em que a equipe discute sobre o que funcionou e o que não funcionou durante a *Sprint*. É a oportunidade de o time determinar o que poderia ser mudado para tornar a próxima *Sprint* ser mais produtiva.
Artefatos	
Product Backlog	Lista de prioridades de requisitos do projeto com os tempos estimados.
Sprint Backlog	Lista de tarefas que define o trabalho da equipe para um *Sprint*. Cada tarefa identifica os responsáveis por fazer o trabalho e a quantidade estimada de trabalho restante na tarefa durante a *Sprint*.
Burndown Chatt	Estima o restante de trabalho ao longo do tempo em uma *Sprint*.

Quadro 1 Descrição dos papéis, formalidades e artefatos do Scrum.

Apresentando uma abordagem sustentada pela experiência e observação e aplicando alguns conceitos da teoria de controle de processos, Scrum compõe um ciclo de vida baseado em três fases: planejamento, desenvolvimento e pós-planejamento (KOSCIANSKI; SOARES, 2007).

O ciclo de vida do Scrum é apresentado na Figura 1.

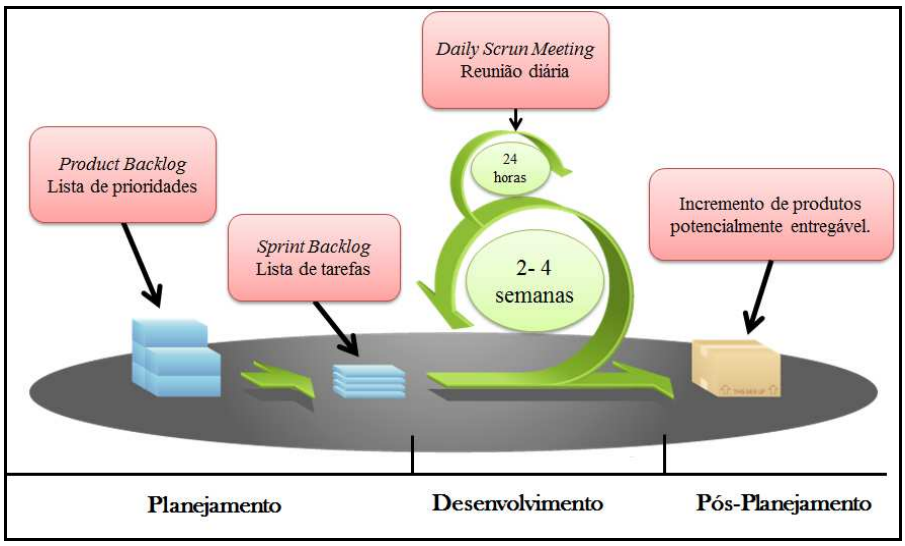

Figura 1 Ciclo de vida do Scrum.
Fonte: Adaptado de MOUNTAIN GOAT SOFTWARE, 2011.

Na fase de planejamento, o *Product Owner* e o cliente definem e descrevem os requisitos em um documento, *Product* B*acklog*, ordenados por prioridade. Em seguida, realiza-se o planejamento, o qual inclui a definição da equipe, as ferramentas para o uso, a identificação de possíveis riscos do projeto e lista de tarefas, *Sprint Backlog*, com os requisitos prioritários. Como resultado, tem-se proposta de uma arquitetura de desenvolvimento.

A fase de desenvolvimento é dividida em ciclos, denominados *sprints*, incluindo a reunião diária, *Daily Meeting*, e semanal, *Sprint Review* . O software é implementado e novas funcionalidades são acrescentadas a cada iteração/ciclo. Os riscos identificados são observados e controlados durante todo o desenvolvimento do sistema. Ao final desta fase, a equipe entrega um incremento funcional do projeto.

A última fase, o pós-planejamento, constituí a integração do software, bem como os testes finais, homologação e documentação do usuário. Em seguida, é feita uma análise do progresso do projeto e demonstração do produto para o cliente.

2.2.2 Extreme Programming (XP)

O primordial dos métodos ágeis Extreme *Programming* (XP), criado por Kent Beck em 1997, é considerado o ponto de partida para as várias abordagens ágeis de desenvolvimento de software.

Um dos métodos ágeis mais utilizados no desenvolvimento de software, oferecendo como objetivo o desenvolvimento rápido, buscando sempre a satisfação do cliente.

Extreme Programming enfatiza o trabalho em equipe altamente colaborativa que se auto organiza em torno do problema para resolvê-lo o mais eficientemente possível. A estratégia do XP é conduzida por cinco valores essencias: comunicação, simplicidade, *feedback*, respeito e coragem (BECK, 2004; WELLS, 2009).

A **comunição** tem a finalidade de manter um relacionamento direto entre cliente e programadores, onde todos os envolvidos no projeto fazem parte da equipe, trabalhando em conjunto para criar a melhor solução para o problema.

A **simplicidade** visa o desenvolvimento de um produto com *design* simples e limpo, criando apenas o necessário e o que foi solicitado pelo cliente.

Os desenvolvedores obtêm o ***feedback*** por meio de testes de software realizados durante toda a confecção do software. Pequenas entregas permitem a verificação do entendimento e mudanças, caso necessário, dos requisitos.

Todos os envolvidos no projeto se dão o **respeito** que merecem como membro da equipe, contribuindo de forma individual ou em grupo. Desenvolvedores respeitam a experiência dos clientes e vice-versa.

Os valores e as práticas do XP proporciona **coragem** aos desenvolvedores a responder às necessidades de mudanças quando essas ocorrem. Além disso, cada membro da equipe deve ter a coragem de expressar suas dúvidas, medos e experiências.

O processo XP é composto por seis fases: exploração, planejamento de entregas, iterações para lançamento, produção, manutenção e morte do projeto (WELLS, 2009).

1) **Exploração:**

Nesta fase, os clientes juntamente com o analista de negócios levantam informações e escrevem histórias de usuários. Ao mesmo tempo, a equipe de desenvolvimento define as ferramentas, as tecnologias e as práticas empregadas no projeto. Esta fase pode levar algumas semanas ou alguns meses, dependendo do conhecimento e da maturidade dos desenvolvedores com as tecnologias utilizadas.

2) **Planejamento de entregas:**

O cliente define as prioridades de cada história de usuário e os desenvolvedores fazem estimativas do esforço necessário de cada um deles. Acordos são feitos sobre o conteúdo da primeira entrega e uma agenda é determinada em conjunto com o cliente. Esta fase tem duração de alguns dias e a entrega não deve ocorrer em mais de três meses.

As estimativas de esforço associado à implementação das histórias do ponto de vista dos programadores são usadas como uma medida para acompanhar a velocidade do desenvolvimento. O planejamento pode ser feito com base no tempo ou escopo.

A velocidade de projeto é usado para estabelecer quantas histórias podem ser implementadas antes de uma determinada data ou quanto tempo vai implementar um conjunto de histórias.

3) **Iterações:**

Fase que inclui vários ciclos de trabalho antes de o sistema ser entregue. Na primeira iteração, é estabelecida uma arquitetura de sistema usada na implementação do projeto, podendo ser alterada durante o período de desenvolvimento. No final da última iteração, o sistema está pronto para entrar em produção. O trabalho é expresso em tarefas, destinadas a cada programador, mas realizadas com a prática de programação em pares.
Produção é a fase que requer testes adicionais e revisão de desempenho antes que o sistema seja transferido para o ambiente do cliente. Ao mesmo tempo, decisões sobre a adição de novas funcionalidades são tomadas.

4) **Produção**

A fase de produção requer testes adicionais e revisão de desempenho antes do sistema ser transferido para o ambiente do cliente. Ao mesmo tempo, adição de novas funcionalidades para a versão atual é estudada, discutida e planejada.

5) **Manutenção**

Na fase de manutenção, enquanto a primeira versão está em uso, novas iterações são desenvolvidas. A equipe faz manutenção no sistema durante o desenvolvimento de novas iterações. Para realizar esta tarefa requer suporte ao cliente. Assim, a velocidade de desenvolvimento pode ser reduzida após o início de produção do sistema. A fase de manutenção pode exigir novos funcionários dentro da equipe e mudanças em sua estrutura.

6) **Morte do projeto**

Ocorre quando o cliente não tem mais funcionalidades a serem incluídas no sistema. Isso requer que o produto atenda as expectativas e necessidades dos usuários, tais como desempenho, usabilidade e confiabilidade. Nesta fase, toda a documentação necessária é finalizada e não ocorrem mais alterações na arquitetura. A morte do projeto também

acontece quando o sistema entra em desuso ou o custo para manutenção se torna muito elevado.

A Figura 2 ilustra as etapas de desenvolvimento no ciclo de vida do XP.

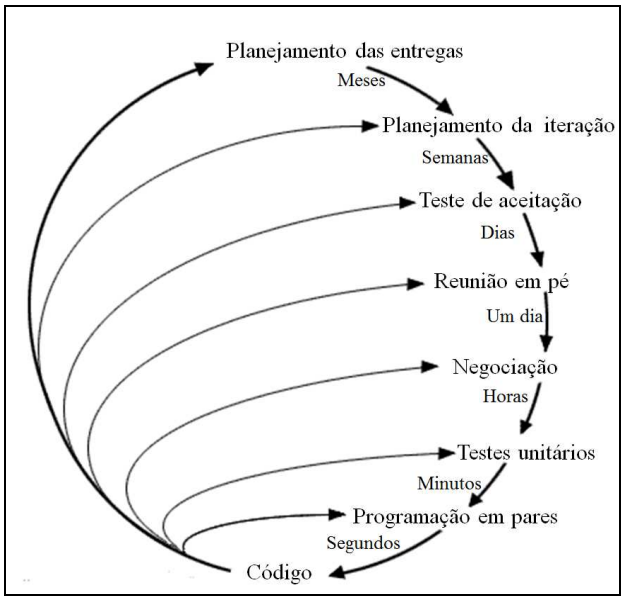

Figura 2 Etapas do ciclo de vida do XP
Fonte: Adaptado de WELLS, 2009.

Para que as fases do ciclo de vida da metodologia XP sejam efetivadas com sucesso, um conjunto de práticas, Figura 3, é necessário. A seguir são descritas as principais práticas definidas por Beck (BECK, 1999; GONZÁLEZ, 2004).

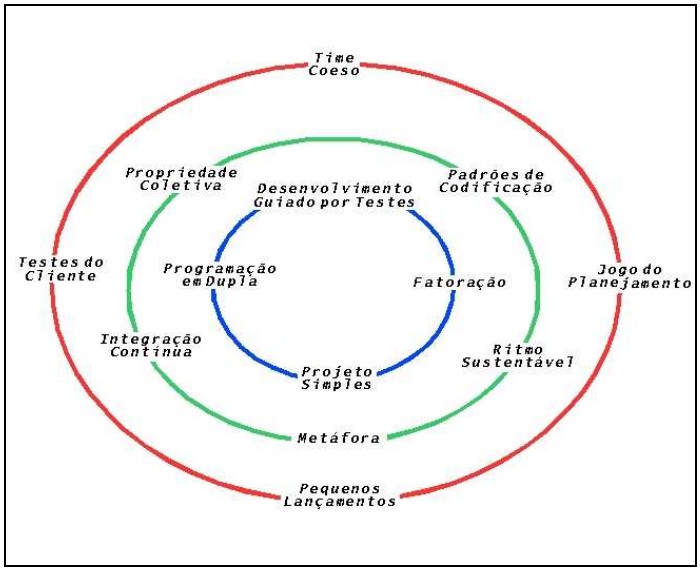

Figura 3 Práticas do XP
Fonte: Adaptado de WELLS, 2009.

1. Planejamento

O planejamento consiste em decidir as prioridades de cada história de usuário, baseando-se em requisitos atuais e não futuros. Além disso, a XP procura estreitar as fronteiras, promovendo relacionamento amigável, entre cliente e desenvolvedores, os quais devem cooperar para o sucesso do projeto.

Desta forma, enquanto o cliente e *stakeholders* decidem sobre o escopo, a composição das versões e as datas de entrega, os desenvolvedores decidem sobre as estimativas de prazo, o processo de desenvolvimento e o cronograma detalhado para que o software seja entregue nas datas especificadas.

2. Entregas frequentes

Visa à construção de um produto simples, atualizado à medida que novos requisitos são implementados. Cada versão entregue deve conter o menor tamanho possível, contendo os requisitos de maior prioridade e no

tempo de um no máximo dois meses, possibilitando *feedback* rápido do cliente.

3. Metáfora

Descrição do sistema sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.

4. Projeto simples

O software é construído de forma simples, focando sempre os requisitos atuais. Eventuais requisitos futuros devem ser adicionados a medida que vão surgindo e de acordo com as necessidades de serem implementados.

5. Testes Primeiro (*test-first)*

Os programadores escrevem testes unitários antes do código, o qual é testado durante o desenvolvimento, focalizando a validação do software.

6. Programação em pares

A codificação é feita em pares, dois desenvolvedores trabalham em um único computador. Um desenvolvedor implementa o código e outro observa continuamente o trabalho sendo feito, procurando identificar erros sintáticos e semânticos e pensando estrategicamente em como melhorar o código.

7. Refatoração

Tem como objetivo o aprimoramento do código existente, sem alteração do comportamento externo. Deve ser feita apenas quando for necessária.

8. Propriedade coletiva

Programadores têm a liberdade melhorar o código em qualquer parte do sistema, promovendo a rotatividade do mesmo. Uma grande vantagem desta prática é, caso um membro da equipe deixe o projeto antes do fim, a equipe conseguirá continuar o projeto com menos dificuldades, pois todos conhecem o código do software.

9. Integração contínua

Novo código é integrado com o sistema atual sempre que alterado. Ao integrar, o sistema é construído a partir do zero e os testes devem passar ou as mudanças são descartadas.

10. Trabalho semanal de 40 horas

A equipe não pode trabalhar uma segunda semana consecutiva de horas extras. Mesmo ocorrendo isoladamente, horas extras usadas com muita frequência é um sinal de problemas mais profundos precisam ser resolvidos. Uma solução para essa situação é o replanejamento das atividades.

11. Participação ativa do cliente

A participação do cliente durante o desenvolvimento do projeto é fundamental, com disponibilidade para tirar as dúvidas da equipe, evitando atrasos e até mesmo implementações erradas. O cliente deve ser considerado como parte integrante do time de desenvolvimento.

12.Código padrão

Adoção de regras e padrões na arquitetura do código, para que este possa ser compartilhado entre os programadores.

13.Cartões CRC

Técnica utilizada para representar requisitos do sistema, com suas responsabilidades e colaboradores.

2.2.3 *Test Driven Development* - TDD

Test Driven Development (Desenvolvimento Guiado a Testes) é uma extensão da pratica *test-first* (testes primeiro) da metodologia *Extreme Programming*, criada por Kent Beck, em que o programador escreve os testes antes do código funcional (BECK, 1999).

TDD é uma forma de programação que incentiva a um bom *design* e é um processo disciplinado o que ajuda a evitar erros de programação, promovendo a qualidade desde o início do desenvolvimento.

O método ágil TDD oferece diversos benefícios como a compreensão do programa, incentivando programadores a explicar o seu código usando casos de teste; eficiência, falhas são identificadas, de forma mais rápida, antes do novo código ser adicionado ao sistema; testabilidade, escrever código automaticamente testável.

O ciclo de TDD, composto por cinco etapas fundamentais, é apresentado na Figura 4 (ASTELS, 2003):

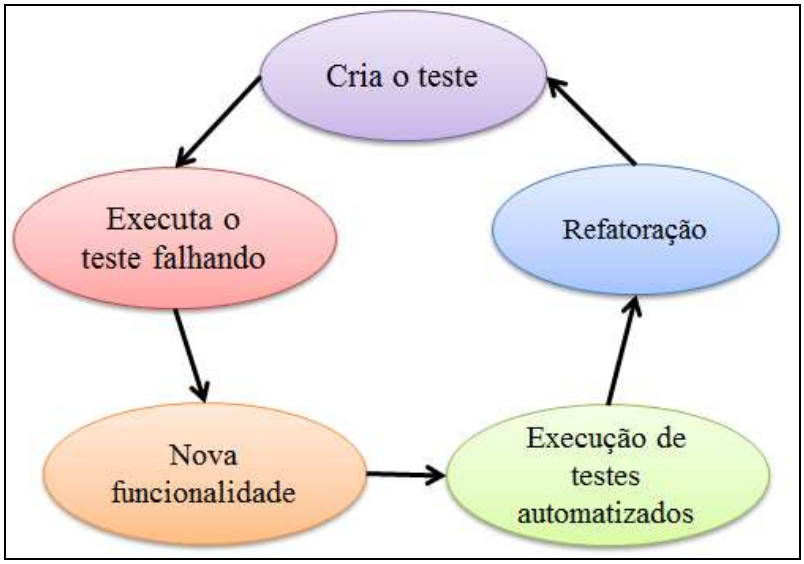

Figura 4 Ciclo de vida TDD
Fonte: Adaptado de ASTELS, 2003.

A primeira etapa consiste em criar o teste para uma nova funcionalidade. A segunda é executa o teste para verificar se o mesmo está correto. O teste não deve passar neste momento, pois a função não foi implementada ainda.

Na terceira etapa, o código é escrito, apenas o mínimo possível para passar no teste (Astels, 2003). Em seguida, os testes devem ser executados para averiguar se a mudança não apresentou problema em algum outro lugar no sistema. Uma vez que passar por testes, a estrutura interna do código deverá ser melhorada através de refatoração. O ciclo é repetido até que a concepção do software seja finalizada.

O desenvolvimento de software com TDD garante que os programadores tratem a implementação de casos de testes. Assim, tem-se o beneficio de automatizar os testes, verificando a necessidade de melhorias no nas funcionalidades testadas (STOBER; HANSMANN, 2010).

Além disso, o TDD permite assegurar a cobertura dos testes para todas as partes do código desenvolvidos e que somente o código escrito que seja necessário para passar estes casos de teste. Código adicional seria considerado "lixo" em termos de desenvolvimento limpo e coeso (ASTELS, 2003).

Com a abordagem de desenvolvimento orientado a testes é extremamente importante que os testes sejam executados constantemente, em pequenos intervalos de tempo, para fornecer um *feedback* instantâneo (BECK , 2002).

2.2.4 *Crystal methodologies*

Crystal é um dos métodos ágeis mais leves, com abordagens ao desenvolvimento de software adaptáveis . É composta de uma família de metodologias: *Crystal Clear*, *Crystal Yellow*, *Crystal Orange*, *Crystal Orange Web*, *Crystal Red*, *Crystal Magneta*, *Crystal Blue*, cujas

características únicas são motivadas por diversos fatores como tamanho da equipe, complexidade do sistema e prioridades do projeto (COCKBURN, 2004).

A família de Cristal aborda a percepção de que cada projeto pode exigir um conjunto de práticas e processos, adaptados, para satisfazer as características únicas do projeto. Possui foco nas habilidades, comunicação, permitindo que o projeto seja mais eficaz e mais ágil do que a abordagem dos processos tradicionais (HIGHSMITH, 2002; COCKBURN, 2004).

Segundo Cockburn (2004), o desenvolvimento de software é "um jogo cooperativo de invenção e de comunicação", o qual se concentra nas pessoas, interação, habilidades, talentos e comunicações como efeitos de primeira ordem sobre o desempenho, onde o processo continua a ser importante, mas secundário.

Existem apenas duas regras absolutas em Crystal: o uso de ciclos incrementais não superior a quatro meses e o uso de oficinas de reflexão, de modo que o método possa ser auto-adaptado (HIGHSMITH, 2002).

Alguns princípios da metodologia incluem o trabalho em equipe, comunicação e simplicidade. Como outras metodologias ágeis, Crystal promove pequenas entregas do software, participação ativa do cliente, adaptabilidade e documentação mínima.

2.2.5 *Adaptive Software Development* (ASD)

Desenvolvido por James A. Highsmith , em 1992, o método ASD (*Adaptive Software Development*) agrega conceitos da teoria de sistemas adaptativos complexos, incentivando, fortemente, o desenvolvimento iterativo e incremental, com prototipagem constante (ABRAHAMSSON *et al.*, 2002).

Segundo Highsmith (2002), as práticas da metodologia ASD são dirigidas por uma abordagem de adaptação e aceitação de mudanças de

forma contínua, aplicando os seis princípios básicos durante todo o ciclo de vida ASD, os quais são apresentados no Quadro 2.

Princípios	**Descrição**
Foco na missão	Diretriz que estabelece o que fazer no projeto, mas não prescreve o como será feito.
Desenvolvimento baseado em componentes	Atividades de desenvolvimento não devem ser orientadas para a tarefa, mas sim focar no desenvolvimento do produto, ou seja, a construção do sistema de um pequeno pedaço de cada vez.
Iteração	Desenvolvimento em etapas focado em refazer em vez de fazer certo da primeira vez.
Tolerância às mudanças	Mudanças são frequentes no desenvolvimento de software. Portanto, é mais importante ser capaz de se adaptar ao sistema do que é controlá-lo.
Incremento de produtos	Prazos fixos de entrega, forçando os desenvolvedores na compreensão dos requisitos efetivos a serem entregues em cada *release*.
Orientação a riscos	O desenvolvimento de itens de alto risco é prioritário.

Quadro 2 Descrição dos princípios da metodologia ASD
Fonte: Adaptado de Highsmith, 2002.

A Figura 5 apresenta o ciclo de vida adaptativo da metodologia ASD, o qual é composto de três fases: especulação, colaboração e aprendizado.

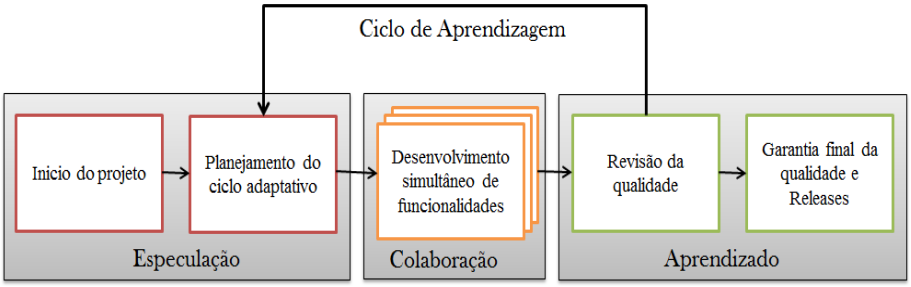

Figura 5 Ciclo adaptativo da metodologia ASD.
Fonte: Adaptado de Highsmith, 2002.

Na fase de especulação, ocorre o planejamento do ciclo adaptativo, a declaração das missões, definições dos requisitos do usuário e restrições de projeto.

Posteriormente, ocorre a fase de colaboração, onde são implementadas, simultaneamente, as funções do sistema. A fase de aprendizado acontece com a liberação do incremento do software. A equipe recebe o *feedback* do cliente. Revisões técnicas são realizadas, focadas nos interesses do cliente.

2.2.6 *Feature Driven Development* (FDD)

Criado por Jeff Luca, Coad Peter e Stephen Palmer, *Feature Driven Development* (FDD) é uma abordagem ágil e adaptativa para sistemas em desenvolvimento. O método FDD foi projetado para trabalhar com as outras atividades de um projeto de desenvolvimento de software e não requer qualquer modelo de processo específico a ser utilizado (HIGHSMITH, 2002).

A abordagem FDD incorpora o desenvolvimento iterativo com as melhores práticas encontrado para ser eficaz na Indústria. Com ênfase nos aspectos de qualidade durante o processo de desenvolvimento, inclui entregas frequentes e tangíveis, juntamente com um acompanhamento preciso da evolução do projeto (ABRAHAMSSON *et al*., 2002).

Ao contrário de algumas outras metodologias ágeis, FDD afirma ser adequado para o desenvolvimento de sistemas complexos, fornecendo métodos, técnicas e orientações necessárias para os envolvidos no projeto para entregar o sistema. Além disso, FDD inclui os papéis, artefatos, metas e prazos necessários em um Projeto.

2.2.7 *Dynamic Software Development Method* (DSDM)

Baseado nas práticas RAD – *Rapid Application Development*, o DSDM, Desenvolvimento Rápido de Aplicativos, consiste em um modelo de desenvolvimento incremental e iterativo para a construção de sistemas com prazos restritos (PRESSMAN, 2006)

O método defende a filosofia de que nada é construído perfeitamente na primeira vez e considera o desenvolvimento de software como um esforço exploratório, com base em nove princípios fundamentais que giram em torno principalmente as necessidades do negócio, participação ativa dos utilizadores, equipes capacitadas, entrega frequentes, testes integrados e colaboração das partes interessadas (ABRAHAMSSON *et al*., 2002).

2.3 Trabalhos relacionados

Na literatura, existem poucas publicações sobre metodologias ágeis que abordam, especificamente, questões de qualidade de software. Apenas revelam as melhorias que o uso dessas metodologias proporciona. Porém,

alguns autores observaram que os desenvolvedores ágeis implementam questões de qualidade diferentemente dos desenvolvedores tradicionais.

Segundo Coram e Bohner (2005), os métodos ágeis têm vantagens, especialmente em acomodar mudanças devido às exigências voláteis. No entanto, eles também apresentam riscos com a gestão das várias etapas do projeto. Mnkandla e Dwolatzky (2006) afirma que há falta de uma técnica abrangente, para avaliar como os processos ágeis atendem aos requisitos de qualidade e melhorias do software. Portanto, o uso desses métodos apresenta um conjunto de vantagens e desvantagens.

O maior desafio dos gestores é determinar se, para um conjunto de atividades do projeto, a metodologia utilizada é apropriada (Coram e Bohner, 2005). Uma característica importante no gerenciamento de projetos, além da compreensão desses riscos é encontrar formas de monitorar, mitigar e gerir esses riscos.

Segundo Huo *et al.* (2004), os métodos ágeis de desenvolvimento podem produzir software mais rápido, mas também é necessário saber como eles atendem as exigências de qualidade. É preciso decidir onde colocar o ponto de equilíbrio na documentação e planejamento para alcançar a flexibilidade e os benefícios prometidos na filosofia ágil.

Bhalerao e Puntambekar (2009) consideram alguns fatores importantes para o uso de metodologias ágeis como conhecimento e fundamentos teóricos dos processos ágeis, bem como a aplicabilidade destes no ambiente de desenvolvimento. É difícil para a equipe implementar os princípios ágeis na organização de acordo com suas necessidades.

Além disso, os métodos ágeis têm seu próprio ciclo de desenvolvimento que traz mudanças tecnológicas, gerenciais e ambientais na organização. Por isso, algumas empresas optam por adotar práticas de desenvolvimento híbrido, onde os processos assumem características de metodologias ágeis e tradicionais.

Percebe-se uma necessidade de desenvolver técnicas que estejam alinhadas com os conceitos ágeis. Um conjunto de práticas aderente aos

métodos ágeis pode ser aplicado para tratar as questões de melhorias do produto no desenvolvimento ágil (Bhalerao e Puntambekar, 2009). Para tal, é necessário definir claramente as fases incluídas em qualquer método ágil e também descrever os artefatos de cada fase.

De acordo com Ambler (2005), a qualidade de software em metodologias ágeis pode ser realizada por meio de um desenvolvimento orientado a testes. A garantia da qualidade em desenvolvimento ágil responde às mudanças de acordo com as exigências do cliente, possibilitando entregas testadas e aprovadas no final de cada iteração.

A qualidade e melhorias dos produtos, por meio do desenvolvimento ágil, podem ser definidas e avaliadas pelo resultado de algumas práticas como: trabalho em equipe, desenvolvimento incremental e interativo, desenvolvimento orientado a testes, modelagem e técnicas de comunicação eficazes.

3 METODOLOGIA

Segundo Jung (2009), a metodologia de pesquisa é um conjunto de métodos, técnicas e procedimentos que tem por finalidade viabilizar a execução da pesquisa, a qual tem como resultado um novo produto, processo ou conhecimento.

Na sequência, busca-se detalhar o tipo de pesquisa e os procedimentos metodológicos utilizados para realização deste trabalho.

3.1 Tipos de pesquisa

Silva e Menezes (2001) afirmam que a pesquisa pode ser classificada de várias maneiras, as quais abordam os seguintes pontos: a natureza da pesquisa, a forma de abordagem do problema, os objetivos e os procedimentos técnicos.

A Figura 6 expõe a classificação em que este trabalho se enquadra, conforme destaque.

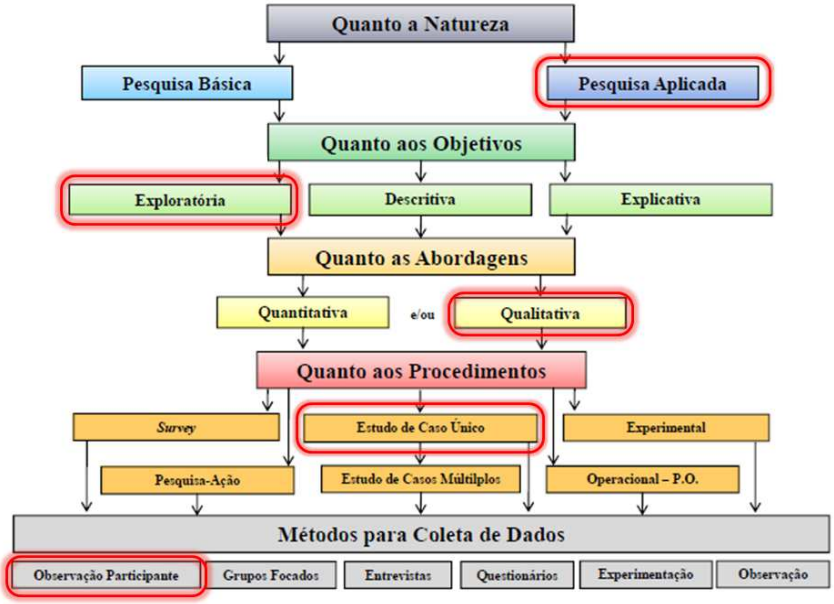

Figura 6 Classificação dos tipos de pesquisa.
Fonte: Adaptado de Jung (2009).

Do ponto de vista da natureza do problema da pesquisa, este trabalho se classifica como pesquisa tecnológica ou aplicada. Apresenta uma abordagem qualitativa com propósito exploratório, utilizando como procedimento técnico o estudo de caso em um laboratório de desenvolvimento de uma instituição pública.

A pesquisa aplicada é uma investigação que tem como objetivo gerar conhecimentos para aplicação prática, que por sua vez são dirigidos à solução de problemas específicos envolvendo verdades e interesses locais (Silva e Menezes, 2001).

Quanto aos objetivos da pesquisa, considera-se exploratória. A investigação exploratória proporciona maior familiaridade com o problema, com vista a torná-lo mais explícito, envolvendo o levantamento bibliográfico e entrevistas com pessoas experientes no problema (Gil, 2002). Favorável quando há limitações de informações e necessidades de um estudo aprofundado sobre um determinado processo e/ou fenômeno (HAIR *et al*., 2009).

Do ponto de vista da abordagem do problema, a pesquisa se classifica como qualitativa. Silva e Menezes (2001) consideram que a pesquisa qualitativa possui uma dinâmica entre o mundo real e o sujeito, onde o processo e os seus significados são os alvos principais da abordagem.

O presente trabalho baseia-se nas teorias dos métodos e gerenciamento ágil de projetos, considerando os processos, os métodos e melhorias da qualidade dos produtos. Jung (2004) diz que na pesquisa qualitativa admite-se a interferência dos valores do pesquisador e considera-se a existência de múltiplas realidades.

Em relação aos procedimentos técnicos, a pesquisa é rotulada como estudo de caso. Os procedimentos consistem em um estudo profundo e exaustivo de um ou poucos objetos, de maneira que permita seu amplo e detalhado conhecimento (Fachin, 2001). De acordo com Jung (2004), o

estudo de caso é uma ferramenta importante para os pesquisadores e tem por finalidade entender “como” e “por que” funcionam as “coisas”.

Na subseção 3.2 são descritos os procedimentos metodológicos utilizados na realização deste estudo.

3.2 Procedimentos Metodológicos

A pesquisa foi realizada no período de fevereiro a outubro de 2011 em uma organização pública situada no estado de Minas Gerais, Brasil.

Uma das maneiras de apresentar e visualizar a metodologia de pesquisa é através do desenho da mesma. O desenho da pesquisa contempla os seus componentes explicando-os em sequências lógicas (Yin, 2010). Os aspectos do desenho da pesquisa estão representados na Figura 7.

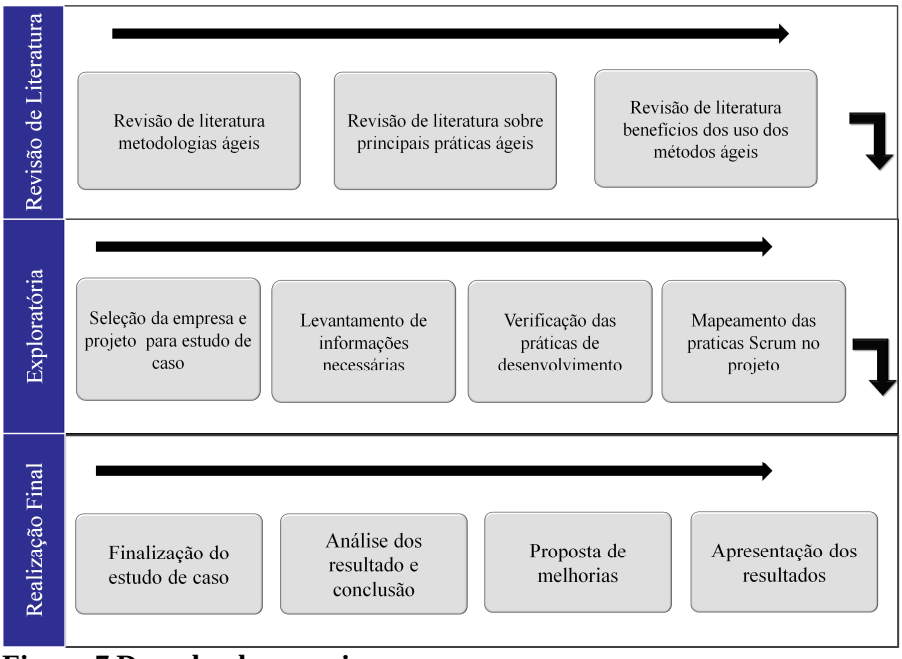

Figura 7 Desenho da pesquisa

O estudo foi divido em duas etapas, onde a primeira etapa foi realizada uma revisão de literatura com o propósito de identificar estudos relacionados à melhoria dos processos ágeis. A revisão de literatura é indicada, pois possibilita o levantamento dos estudos relacionados, bem como o tratamento dos métodos e benefícios do desenvolvimento ágil, permitindo melhor compreensão do assunto a ser investigado.

Para revisão de literatura foi realizada uma pesquisa em diferentes bases de dados acessadas via Portal Brasileiro de Busca (Portal de Periódicos) da CAPES – Agência Brasileira de Apoio e Fomento a Pesquisa, utilizando a combinação de termos: *Agile Development*, *Agile Methodologies*, *Agile Practices*, *Scrum* e *Extreme Programming.*

A segunda etapa da pesquisa compõe a fase exploratória, que para fins de averiguação e aprimoramento dos resultados, selecionou-se uma organização desenvolvedora de software para a realização de um estudo de caso exploratório na tentativa de aplicação e uso de um método ágil em uma equipe de desenvolvimento.

Este trabalho foi realizado no contexto de um projeto de desenvolvimento de software conduzindo na Universidade Federal de Lavras, Minas Gerais. Para a coleta dados, foi utilizada a técnica de pesquisa participante, desenvolvida a partir da interação entre o pesquisador e os membros da equipe do projeto investigado.

A partir do levantamento das informações da primeira etapa foi realizado um mapeamento das práticas Scrum utilizadas pela equipe e quais práticas não foram cumpridas. Em seguida foram observadas quais atividades desenvolvidas pela equipe tratavam melhorias no produto em desenvolvimento.

4 RESULTADOS E DISCUSSÃO

Este capítulo descreve o produto em estudo (ProManager), a equipe e tecnologia envolvidas, as percepções do processo e discussão de elementos de qualidade associados à metodologia ágil Scrum e práticas no objeto de pesquisa.

4.1 Descrição do produto de software ProManager

ProManager é um sistema de gestão estratégica que permite elaboração e acompanhamento de projetos e planos diretores e de negócios, públicos ou privados de forma rápida e segura.

A figura 8 mostra a tela de painel de controle do sitema.

Figura 8 Tela de painel de controle do ProManager.

O ProManager permite o cadastro de diferentes tipos de usuários com escala de permissões, além de cadastro de áreas, temas, metas, ações, indicadores de desempeho e planos de gestão, bem como alterar e excluir cadastros.

A funcionalidade "Painel de Bordo" comporta ao usuário acompanhar o desempenho da organização.

O software utiliza como referência os princípios do [1]*Balanced Scorecard* (BSC) para gerenciamento de desempenho e alinhamento estratégico de um conjunto de instituições nele cadastradas.

Oferece serviços alinhados à realidade de empresas privadas e públicas no desenvolvimento, comunicação e acompanhamento de: planos diretores institucionais (PDIs); plano diretor de tecnologia da informação (PDTI); planos estratégicos (PEs); planos estratégicos de tecnologia da informação, marketing e produtos.

A gestão de desempenho efetuada pelo sistema é baseada em *Scorecards*. Uma vez associado à determinada área, um *Scorecards* destina-se a gerenciar o desempenho dela. Os *Scorecards* contêm um conjunto de quatro perspectivas: finanças, processos internos, clientes, aprendizado e crescimento.

Essas perspectivas contêm um conjunto de objetivos que, por sua vez, possuem um conjunto de métricas associadas. A partir dessas métricas, pode-se medir o desempenho da organização, ou de parte dela, em alcançar objetivos estabelecidos. A medição do alcance dar-se-á por meio da comparação do valor alcançado pela métrica com a meta estabelecida.

O alinhamento estratégico é efetuado por meio do cascateamento de *Scorecards*, conforme a disposição hierárquica da instituição. Por meio do cascateamento, são relacionados objetivos e métricas de níveis inferiores a objetivos e métricas de níveis superiores. A partir daí, pode-se impor metas em níveis inferiores de modo a fazer com que áreas da instituição trabalhem em prol da estratégia global da organização.

---

[1]*Balanced Scorecard* (BSC) é um método de gestão criado por Kaplan e Norton no início dos anos 90, utilizado como ferramenta para avaliação e medição do desempenho, implementando novas estratégias de gerenciamento nas empresas (PRADO, 2002).

Como resultado, o sistema deverá, através de indicadores e relacionamento estabelecidos, expor a situação da organização em atender às estratégias de negócio, bem como o desempenho das diversas áreas em atender a tais estratégias. A partir daí, pode-se identificar pontos falhos a fim de corrigir os direcionamentos necessários ao alcance dos objetivos.

Atualmente o sistema está sendo adaptado às necessidades de planejamento do PDI – Plano de Desenvolvimento Institucional de uma Universidade Federal.

4.2 Descrição da equipe

Considerando a terminologia do Scrum, a equipe de desenvolvimento analisada é composta por um *Product Owner* (supervisor geral), *Team* (time - três analistas desenvolvedores), *Scrum Master* (um gerente líder), um gerente de produto, um *design* gráfico e um analista de negócio.

Esta equipe está envolvida no projeto ProManager, que tem como objetivo o desenvolvimento de um sistema de gestão de desempenho de negócios, baseado no conceito da ferramenta *Balanced Scorecard*.

O cliente é uma organização do setor público, que está em fase de concepção do plano de desenvolvimento institucional.

A equipe do projeto utiliza o método ágil *Scrum* para gerenciar o projeto, adotando algumas práticas ágeis de gerenciamento como *product backlog*, ciclo em *Sprints*, reuniões diárias e reunião de retrospectiva da *Sprint*, conceitos descritos na subseção 2.3.1.

Por se tratar de uma equipe recém-formada, o primeiro desafio enfrentado pela equipe foi se adaptar aos princípios Scrum, bem como aplicá-los ao processo de desenvolvimento do ProManager.

4.3 Processo de desenvolvimento do ProManager

O Processo de desenvolvimento do ProManager constitui em quatro fases: Concepção**,** ênfase no escopo do sistema; Elaboração, ênfase na arquitetura; Construção, ênfase no desenvolvimento; e Transição, ênfase na implantação.

Fase1 - Concepção

É de responsabilidade do *Product Owner*. Seu objetivo é colher as informações necessárias sobre o sistema pretendido junto ao cliente para definição do escopo do projeto. Os produtos oriundos desta fase são: o modelo abrangente do sistema e a lista de requisitos.

O desenvolvimento de um modelo abrangente é uma atividade inicial, realizada por membros do domínio do negócio e por desenvolvedores, sob a orientação de um modelador de objetos experiente. É realizado um estudo sobre o domínio do sistema, suas funcionalidades e requisições, onde cada participante irá propor um modelo que satisfaça o problema em questão. Um dos modelos propostos será selecionado por consenso. Os produtos oriundos desta fase são os diagramas de classes com foco na forma do modelo, contendo métodos e atributos identificados.

Fase 2 – Elaboração

Com foco na arquitetura, esta fase é de responsabilidade do gerente de projetos (*Scrum Master*). Seu objetivo é, juntamente com o time de desenvolvimento, interpretar as requisições dos clientes para que o sistema pretendido seja implementado com qualidade e atendendo às necessidades do cliente.

A elaboração é dividida em duas etapas a serem cumpridas: planejamento e reunião de planejamento do *Sprint* (*Sprint Planning Meeting*).

i) Planejamento

Tem como participantes o cliente, o time de desenvolvimento, o *Product Owner* e o *Scrum Master*.

É preciso conhecer o problema, levantar os requisitos, estimar o esforço e definir as entregas. A participação tanto da equipe de desenvolvimento quanto do cliente é fundamental nesta etapa, a qual será o ponto de partida para um novo ciclo de desenvolvimento.

O objetivo do planejamento é definir os requisitos (*User Stories*) de maior valor e prioridade. Para cada caso de uso ou *User Stories* analisado, o cliente deve indicar quais as funcionalidades de maior prioridade e de menor prioridade. O resultado desta reunião de planejamento é a lista de requisitos do projeto (*Product Backlog*).

ii) Reunião de planejamento do *Sprint* (*Sprint Planning Meeting*)

A reunião *Sprint Planning* é onde o *Scrum Team* (*Scrum Master* e time) e o *Product Owner* determinam quais funcionalidades e atividades serão realizadas no próximo *Sprint*. O artefato gerado nessa etapa é a lista de tarefas com os responsáveis e tempos estimados. (*Sprint Backlog*).

Fase 3 - Construção

Possui foco no desenvolvimento do software, com entregas de incrementos do software no final de cada *Sprint*.

Uma *Sprint* pode ter duração de 2 a 4 semanas, dependendo do escopo do projeto. Esse prazo deve ser respeitado ao máximo, para que o produto seja garantido ao cliente e ao usuário final.

A primeira *Sprint* é sempre experimental, onde a equipe implementa o primeiro caso de uso do sistema. Nesta fase são aplicadas as técnicas de *pair programming*, para aperfeiçoar o processo de entendimento do código.

Ao final da implementação das funcionalidades listadas no *Product Backlog*, são realizados os testes, funcionalidade e usabilidade, e a refatoração do código, caso necessário.

Dentro da terceira fase três etapas devem ser cumpridas: *Daily Meeting, Sprint Review, Sprint Retrospective.* Breve reunião diária, realizada em pé, também chamado de *Stand up Meeting* ou *Daily Meeting*, em que cada participante fala sobre o progresso conseguido, o trabalho a ser realizado e/ou o que o impede de seguir avançando. A reunião diária deve ter a duração mínima de 10 minutos e máxima de 15 minutos.

Reunião de Revisão (*Scrum Review*) da *Sprint* onde os participantes reveem o trabalho concluído e não concluído. Nesta etapa, é realizada uma apresentação ao cliente do *release* implementado, feita pelo gerente de projetos, Scrum *Master* e o *Product Owner.* Em seguida ocorre a reunião de retrospectiva em que os membros da equipe refletem sobre a *Sprint* passada e propõe melhorias contínuas de processos.

Fase 4 – Transição

O código do projeto é monitorado através do repositório SVN (*Subversion*), software utilizado para controle de versões de sistemas, do laboratório, situado no servidor de desenvolvimento. Os projetos a serem implementados são separados em pastas com nomenclaturas padrão.

É obrigatório fazer *upload* (*commit*) do código implementado ao final de cada dia de trabalho. Assim que a versão da *Sprint* for liberada, está deve ser instalada no servidor de homologação. Se aprovada na apresentação para o cliente ao final da *Sprint*, o *release* deve ser liberado para utilização no servidor de produção.

4.4 Mapeamento das práticas Scrum executadas no ProManager

Segundo Schwaber e Beedle (2002), o aumento na aceitação do método ágil Scrum se deve ao fato deste fornecer um processo apropriado para o desenvolvimento de software, podendo, devido às características na ênfase de valores e práticas para o gerenciamento de um projeto, ser facilmente ajustada a outros métodos (LARMAN, 2003).

Conforme conceitos descritos o Capítulo 2, para que as práticas Scum seja implementadas na íntegra, as formalidades *Sprint Planning, Daily Meeting, Sprint Review* e *Sprint Retrospective* devem ser executadas nas fases de planejamento, desenvolvimento e pós planejando, respectivamente, gerando os artefatos, *Product Backlog, Sprint Backlog* e *Burndown Chart* (Gráfico de desempenho).

As formalidades são executadas e características do produto são discutidas pela a equipe de desenvolvimento, o *Product Owner,* o *Scrum Master* e o Cliente.

O processo de desenvolvimento do ProManager baseia-se nas práticas do método ágil Scrum com seus papéis, formalidade e artefatos. No entanto, as práticas Scrum que a equipe efetivamente utiliza são as cerimônias *Sprint Planning* e *Daily Meeting*, gerando os artefatos *Product Backlog, Sprint Backlog* e documento de requisitos.

As formalidades *Sprint Review* e *Sprint Retrospective,*as quais estão no escopo do processo de desenvolvimento do projeto não são executadas na fase de desenvolvimento.

Houve dificuldades em cumprir o planejamento, em que a duração da *Sprint* que deveria ser de 2 a 4 semanas se estendeu mais que o planejado devido aos limites de capacitação da equipe que demandou mais tempo para resolução de erros de funcionalidade, interface e lógica do negócio, apresentados na versão que está em fase de implementação.

Outra fase do processo não executada corretamente, conforme definição Scrum, é a reunião diária (*Daily Meeting*), isso ocorreu devido a

incompatibilidade de horários de trabalho da equipe, visto que é composta por estudantes de graduação da Universidade Federal de Lavras com compromissos acadêmicos considerados como prioritários.

Uma das maiores dificuldade que a equipe encontrou, ao adotar o método ágil Scrum, foi se adaptar às práticas deste método e ajustar o projeto e a equipe, de forma a obter um produto que atenda as expectativas dos usuários e dos envolvidos no projeto.

Além disso, para tratar questões de melhorias do produto, a equipe incorporou práticas de desenvolvimento comum a outros métodos, ágeis ou não.

4.5 Práticas incorporadas ao processo de desenvolvimento do ProManager

Ao longo dos anos, o aumento na aceitação do método ágil Scrum, proporcionou sua popularização e diversos relatos sobre seus benefícios (Schwaber e Beedle, 2002). Entretanto, suas práticas estão alinhadas a eficiência dos processos e não à melhoria do produto.

A equipe ProManager, além das práticas de gerenciamento e controle do métodos Scrum, utiliza práticas comuns às demais metodologias ágeis incorporadas as atividades do projeto.

O *team* do ProManager agrega algumas técnicas de desenvolvimento advindas do XP como *pair programming*, *stand up meetings*, integração contínua de forma bem simples armazenando *builds* em um servidor de desenvolvimento/produção, SVN (*Subversion*), triagem de erros, refatoração, propriedade coletiva do código, uso de ferramentas para monitoramento de atividades, testes unitários e *design* incremental

Além disso, a equipe adota algumas ferramentas e *framework* comum ao desenvolvimento de software orientado a objetos, os quais

contribuem efetivamente para a concepção de um produto com qualidade. Características que são observadas a cada entrega do software.

As práticas incorporadas ao processo Scrum são adicionadas, principalmente, nas fases de elaboração, construção e transição do produto, permitindo o alcance de melhorias na confiabilidade, funcionalidade, usabilidade, manutenibilidade, correção, eficácia e satisfação.

Para o alcance do requisito funcionalidade, práticas como planejamento, *design* simples, participação ativa e *feedback* do cliente, entregas frequentes e teses de software, permitem o melhor entendimento dos requisitos e do problema a ser solucionado. Além da implementação correta do sistema.

Para a construção de um sistema funcional, a equipe executa pequenos testes manuais como: testes de unidade, regressão, caixa-preta, funcional, interface e segurança, durante o desenvolvimento para assegurar a acurácia do software. Utiliza um sistema de *login* para garantir a segurança de acesso, protegendo informações e dados, de forma que pessoas ou sistemas não autorizados possam acessá-los e alterá-los.

Para assegurar confiabilidade do sistema, evitando falhas decorrentes de defeitos no software, a equipe se atenta aos detalhes da especificação dos requisitos. Aplica a prática de uso de metáforas, oriundas do método ágil XP.

As metáforas descrevem o sistema sem a utilização de termos técnicos, usando objetos do mundo real como referências. Além desta prática que permite entender as características e necessidades do sistema, a equipe trabalha, também, com pequenas entregas frequentemente, possibilitando o *feedback* constante do cliente. Isto evita surpresas caso o software apresente alguma falha, aumentando a probabilidade do produto final estar de acordo com os requisitos do cliente.

Além disso, na fase de transição, utiliza um sistema de controle de versão, permitindo, caso haja falhas, a recuperação de dados em tempo hábil. Realiza exclusão lógica no banco de dados para proporcionar a

recuperabilidade e disponibiliza suporte ao usuário, quando este não consegue executar alguma funcionalidade do software.

Outra prática de desenvolvimento, amplamente utilizada, é a programação em par, executada durante o processo de concepção do software. Os desenvolvedores do ProManager são jovens programadores em fase de aprendizado. A programação em pares, além de ter como resultado um código coeso e enxuto, permite aos desenvolvedores estarem continuamente aprendendo um com o outro e discutindo suas experiências.

A usabilidade do software é promovida por meio do *design* da interface que utiliza padrões de usabilidade como ícones intuitivos e cores agradáveis. Além disso, o produto pode ser apresentado em dois idiomas (português e inglês). Outra prática é a realização de testes de acordo com os princípios de usabilidades de Nielsen: eficiência, satisfação subjetiva, capacidade de memorização, prevenção de erros, facilidade de aprendizado.

A manutenibilidade do sistema é obtida através da implementação de códigos com tratamento de exceções e falhas, os quais são testados e verificados, de forma simples, durante o desenvolvimento.

Uma atividade que proporciona fácil manutenção do software é o uso de padrões de projeto. A equipe implementa os padrões MVC (*Model View Controller*), que divide o sistema em camadas, facilitando encontrar erros, quando estes ocorrem, e *Strategy*, que define um conjunto de algoritmos que varia independentemente dos clientes que o utilizam. *Strategy* comporta a construção de um código legível, utilizando o paradigma orientado objetos (FREEMAN *et al*., 2007).

A prática de *feedback* possibilita ao desenvolvedor obter informações constantes do código e do cliente, promovendo a qualidade no quesito correção. Ou seja, a cada *feedback* o programador poderá fazer correções e refatoração, caso necessário, gerando um produto final correto conforme especificações e objetivos visados pelo cliente.

A Figura 9 ilustra as principais práticas de desenvolvimento utilizadas pela equipe do ProManager em cada fase do processo para melhorias do processo e do produto.

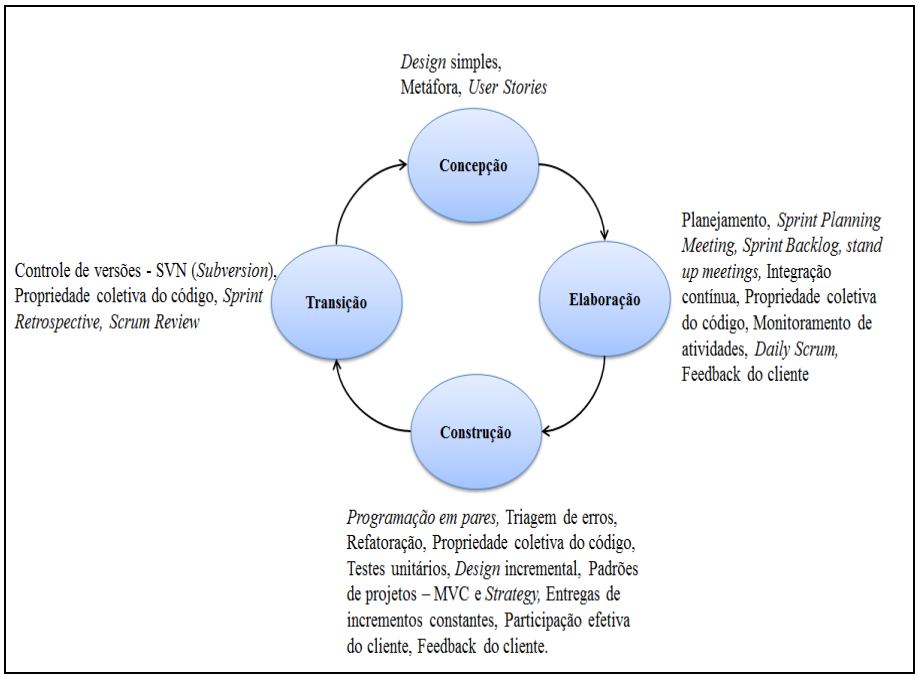

Figura 9 Principais práticas de desenvolvimento utilizadas pela equipe do ProManager em cada fase do processo

4.6 Problemas encontrados e proposta de melhorias no ProManager

As metodologias ágeis têm apresentado números satisfatórios em relação ao escopo, custo e qualidade dos produtos, utilizando um conjunto de diferentes técnicas e práticas de desenvolvimento que compartilham valores e princípios.

Mesmo incorporando outras práticas de desenvolvimento ao processo Scrum, o projeto em estudo apresentou alguns problemas como:

falhas na comunicação, planejamento das *Sprints*, falhas no processo de desenvolvimento, erros de especificação e atrasos nas entregas, impactando na qualidade do processo e, consequentemente, na qualidade do produto.

Para os erros de funcionalidade, recomenda-se o uso de práticas de desenvolvimento da metodologia TDD (*Test Driven Development*), descrita na subseção 2.3.3, onde testes são escritos antes do código. Essa prática, permitirá a verificação de falhas no sistema a cada funcionalidade implementada.

A equipe faz testes durante o desenvolvimento, mas os mesmos são manuais. Para melhoria no processo de desenvolvimento e resultados satisfatórios, uma sugestão é utilizar ferramentas para automatizar os testes: JUnit, JMeter, *Selenium*, *Metrics*, e gerar relatórios mais eficientes de forma a evitar que erros encontrados na versão em teste não reapareçam nas próximas entregas.

As metodologias ágeis prezam a simplicidade, promovendo um processo de desenvolvimento iterativo com entrega de incrementos e de documentação mínima. Nesse caso, a documentação do projeto se encontra no documento de requisitos, definido juntamente com o cliente, especificação do sistema, os *Product Backlog* e os comentários na codificação. Esses artefatos servirão de base para homologação do sistema, além de impactar na manutenibilidade do sistema. Um código devidamente documentado e construído utilizando padrões de projeto será de fácil manutenção.

Visando encontrar o ponto de equilíbrio entre as práticas de garantia da qualidade e a abordagem Scrum, o time incorporou práticas de qualidade de forma a adaptar a metodologia Scrum às necessidades do projeto. Mesmo com a ocorrência de erros no sistema, que está em fase de testes, a equipe vêm superando os desafios e apresentando resultados preliminares satisfatórios em termos de qualidade do software.

5 CONCLUSÕES

As metodologias ágeis surgiram em resposta ao insucesso apresentado pelas metodologias tradicionais e vieram para ficar. Entretanto, percebe-se que não se trata de um padrão único de desenvolvimento, pelo contrário, os métodos apresentados no “Manifesto Ágil” e ao longo desses anos podem ser adaptadas ou até mesmo utilizadas em conjunto, visando o melhor desempenho da organização.

Scrum é um método ágil fundamentado nos princípios de gerenciamento e controle. É amplamente utilizado, proporcionando melhorias em termos de gerência, satisfação do cliente e comunicação. Contudo, mostrou-se limitada em termos de práticas de garantia da qualidade de software.

Para superar essas limitações, equipes, como a deste estudo, incorporam práticas de desenvolvimento, ágeis ou não, para garantir melhoria no desempenho e mesmo no produto de software. Tais práticas são: programação em pares, integração contínua, pequenas entregas, triagem de erros, refatoração, propriedade coletiva do código, *build* de 10 minutos, testes unitários e *design* incremental.

Este trabalho teve como objetivo verificar e analisar como as práticas ágeis estão sendo tratadas por uma equipe de desenvolvimento que utiliza Scrum, ou seja, como a equipe Scrum busca melhorias do produto de software.

O objetivo foi alcançado, uma vez que, o estudo mostrou que para tratar melhorias do produto de software no processo Scrum a equipe incorporou práticas comuns a outros métodos, unindo princípios de um ou mais procedimentos de desenvolvimento de software, ágeis ou não, ao projeto e à equipe, prezando desempenho, qualidade e satisfação do cliente.

A viabilidade de incorporar práticas de desenvolvimento comum a outros métodos é considerável, visto que possibilita o uso de uma

metodologia mais completa que atenda as necessidades do desenvolvimento ágil.

Para os pesquisadores, este trabalho contribui como um ponto de partida para o aprimoramento dos processos Scrum com vista na melhoria do produto. Para os praticantes, contribui para a evolução das metodologias ágeis com vista ao aprimoramento da qualidade de software.

5.1 Trabalhos futuros

Como trabalho futuro vê-se a oportunidade de realizar a verificação, feita neste estudo, em uma equipe que possua processos de desenvolvimentos amplamente definidos e com uma equipe que apresenta maior nível de experiência.

Há, também, a oportunidade de evoluir os processos Scrum para torná-lo um método de desenvolvimento mais amplo, possuindo características de gerenciamento e desenvolvimento.

REFERÊNCIAS

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M. T.; RONKAINEN, J.; **New Directions on Agile Methods: A Comparative Analysis**; 2003.

ABRAHAMSSON, P.; SALO, O.; RONKAINEM, J.; WARSTA, J.; **Agile software development methods – Review and analysis**; 2002.

ASTELS, D.; **Test Driven Development: A Practical Guide**; Prentice Hall, 2003.

AMBLER, S.; **Quality in an Agile World, Software Quality Professional;** Software Quality Professional*;* Vol. 7, No. 4, p. 34-40; 2005.

BECK et al.; **AGILE MANIFESTO**. 2001. Disponível em: http://www.agilemanifesto.org/. Acesso em: 01/06/11.

BECK, K.; **Extreme Programming Explained: Embrace Change;** 2rd ed. Addison-Wesley Professional, 2004.

BECK, K.; **Extreme Programming Explained**; 1ª ed., 1999.

BECK K.; **Test-driven development: by example**; Addison-Wesley Professional; 1ª ed., 2002.

BHALERAO, S.; PUNTAMBEKAR, D.; **Generalizing Agile Software Development Life Cycle;** International Journal on Computer Science and Engineering Vol.1(3), 2009, 222-226, 2009.

COCKBURN, A. **Crystal Clear: A Human-Powered Methodology for Small Teams**; Addison-Wesley Professional, 2004.

CORAM, M.; BOHNER, S.; **The Impact of Agile Methods on Software Project Management**; Engineering of Computer-Based Systems, IEEE International Conference and Workshops on the; p363 – 370; 18 de abril de 2005.

DINGSOYR, T.; HANSSEN, G. K.; DYBA, T.; ANKER, G.; NYGAARD, J. O.; **Developing Software with Scrum in a Small Cross-Organizational Project**, pp. 5–15, 2006.

FACHIN, O.; **Fundamentos de metodologia**. 3. ed. São Paulo: Saraiva, 2001.

FREEMAN ERIC; FREEMAN ELISABETH; SIERRA K.; BATES, B.; **Use a Cabeça! Padrões de Projetos (Design Patterns)**; 2a Edição, 2007.

GIL, A. C.; **Como elaborar projetos de pesquisa;** 4. ed. São Paulo: Atlas, 2002.

GONZÁLEZ, C. S.; **ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source inovadoras**, 2004.

HAIR J. F.; BLACK W. C.; BABIN B. J.; ANDERSON R. E.; TATHAM R. L.; **Análise multivariada de dados**, 6ª ed. Porto Alegre: Bookman, 2009.

HIGHSMITH, J. **Agile Software Development Ecosystems.** Foreword by Tom De Marco. Addison-Wesley Pearson Education, 2002.

HUO, M.; VERNER, J.; ZHU, L.; MUHAMMAD, A. B.; **Software Quality and Agile Methods**; Computer Software and Applications Conference, compsac 2004. Proceedings of the 28th Annual International; 2004.

JUNG, C. F.; **Metodologia aplicada a projetos de pesquisa: Sistemas de Informação & Ciência da Computação**. Proposta de TCC e Projeto de Pesquisa; Taquara, 2009.

JUNG, C. F.; **Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos**. Rio de Janeiro: Axcel Books do Brasil, 2004.

KATSURAYAMA, A. E.**; Apoio à Garantia da Qualidade do Processo e do Produto em Ambientes de Desenvolvimento de Software Orientados a Organização**, 2008.

KOSCIANSKI, A.; SOARES, M. S.; **Qualidade de Software. Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2ª Edição. 2007.

LARMAN, C.; **Agile and Iterative Development: A Manager's Guide**, Addison –Wesley, 2003

MNKANDLA, E.; DWOLATZKY, B.; **Defining Agile Software Quality Assurance**; Software Engineering Advances, International Conference on; Oct. 2006; 36 - 36 ; 19/12/ 2006.

MOUNTAIN GOAT SOFTWARE; **Introduction to Scrum - An Agile Process**; Disponível em: http://www.mountaingoatsoftware.com/topics/scrum. Acesso em: 08/10/2011.

NAIK, K.; TRIPATHY, P.; **Software Testing and Quality Assurance – Theory and Practice**. Chapter 17- Software Quality. P-519. 2008

PRADO, L. J.; **Guia Balanced Scorecard**; 1ed., 2002.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. São Paulo McGraw-Hill, 2006.

SCHWABER, K.; BEEDLE, M.; **Agile Software Development with Scrum**. Prentice Hall, 2002.

SCHWABER, K.; **Agile Project Management with Scrum**. Microsoft Press. 1ª Edição, 2004.

SCRUM ALLIANCE; **Who is the Scrum Alliance?** - Disponível em: http://www.scrumalliance.org/. Acesso em: 15/09/11.

SIAKAS, V. K.; GEORGIADOU E.; **PERFUMES: A Scent of Product Quality Characteristics;** Technological Educational Institute of Thessaloniki, Greece, 2005.

SILVA, E. L.; MENEZES, E. M.; **Metodologia da pesquisa e elaboração de dissertação**. Florianópolis, 3ª ed., 2001.

STOBER, T.; HANSMANN, U.**; Agile Software Development - Best Practices for Large Software Development Projects**; 2010.

WELLS, D.; **Extreme Programming: A gentle introduction;** Disponível em: http://www.extremeprogramming.org; 2009.

YIN, R. K.; **Estudo de caso: planejamento e métodos**. 4. ed. Porto Alegre: Bookman, 2010.