UNIVERSIDADE DE CAXIAS DO SUL
Centro de Computação e Tecnologia da Informação
Curso de Bacharelado em Sistemas de Informação

Daiane Morandi da Costa

CUSTOMIZAÇÃO DE PROCESSO DE DESENVOLVIMENTO DE SOFTWARE BASEADA EM MODELAGEM ÁGIL

Caxias do Sul

2010

Daiane Morandi da Costa

CUSTOMIZAÇÃO DE PROCESSO DE DESENVOLVIMENTO DE SOFTWARE BASEADA EM MODELAGEM ÁGIL

Trabalho de Conclusão de Curso para obtenção do Grau de Bacharel em Sistemas de Informação da Universidade de Caxias do Sul.

Iraci Cristina da Silveira De Carli
Orientadora

Caxias do Sul

2010

Dedico este trabalho aos meus pais
Albino Ferreira da Costa (*in memoriam*)
e Regina Morandi da Costa.

AGRADECIMENTOS

Primeiramente agradeço à Deus, pela vida e pelas oportunidades, uma delas de estar concluindo esta graduação na Universidade de Caxias do Sul. Por sempre me iluminar nas minhas escolhas e dar a força necessária para lutar pelos meus objetivos pessoais e profissionais.

Agradeço a minha família, mãe, pai (*in memoriam*), irmã e cunhado que são as pessoas que acompanharam toda a minha trajetória até aqui e sempre estiveram presentes nos momentos em que precisei. Com certeza serão as pessoas que estarão comigo para sempre.

Agradeço ao meu namorado César, pelo amor, dedicação e paciência em todos os momentos, principalmente pela compreensão nos momentos de ausência.

Agradeço a minha professora orientadora, Iraci Cristina da Silveira de Carli, pela sua disponibilidade, atenção e dedicação ao longo destes dois semestres de trabalho. Agradeço por por me mostrar o caminho e por acreditar em mim.

Um agradecimento especial aos meus colegas e amigos Fabiano Marcon, Fabrício Scariott e Rodolfo Stangherlin, equipe do AVA, pelas contribuições e pela troca de experiências que contribuíram para o meu aprendizado e crescimento profissional. Agradeço a essa equipe pela oportunidade de aplicação do estudo de caso realizado neste trabalho.

E por fim, agradeço a todos os colegas, amigos e demais professores que contribuíram de alguma forma com a realização deste trabalho.

RESUMO

A definição de um processo de desenvolvimento de *software* não é uma tarefa trivial para as organizações, principalmente quando esse processo é utilizado para realizar manutenção. Pois, além de estar desenvolvendo o *software*, é necessário manter o sistema atual em operação. As dificuldades para a produção de sistemas vão muito além das questões técnicas. Fatores organizacionais, culturais e humanos são responsáveis por algumas das variáveis que contribuem para tornar o desenvolvimento de sistemas de *software* uma atividade altamente complexa.

Uma questão muito importante a ser considerada quando se pensa em definir uma metodologia de desenvolvimento de *software* é a evolução do sistema. A evolução do *software* é imprescindível para as organizações que são dependentes de seus sistemas, os quais são considerados importantes ativos de negócio. O fato de não existir um processo ideal, possibilita que sejam feitas melhorias nos processos existentes.

A customização de processos não significa simplesmente adotar métodos específicos, ferramentas ou modelos de processo padrão. A customização deve ser encarada como algo específico para cada organização. Geralmente as organizações quando buscam a melhoria de seus processos de desenvolvimento procuram se apoiar em métodos já conhecidos e utilizados, como as metodologias "tradicionais" e "ágeis".

Neste trabalho é apresentada a customização do processo de desenvolvimento de *software* de uma equipe específica que realiza manutenção evolutiva em um *software web*.

Palavras-chave: Manutenção de *software*, Customização de processos, Métodos ágeis

ABSTRACT

The definition of a process of software development is not a simple task for organizations, especially when this process is used to perform maintenance. Because, besides being developing the software, you must keep the current system in operation. The difficulties in producing systems go far beyond technical issues. Organizational factors, cultural and humans are responsible for some of the variables that contribute to making the development of software systems a highly complex activity.

A very important issue to consider when thinking about defining a methodology for software development is the evolution of the system. The software evolution is essential for organizations that are dependent on their systems, which are important business assets. The fact that there is an ideal process, possible that improvements be made in existing processes.

The customization process does not mean simply adopting specific methods, tools or models of standard process. The customization must be seen as something specific to each organization. Often when organizations seek to improve their development processes seek to rely on known methods and used as the methodologies "traditional" and "agile."

In this paper the process of customizing the software to a specific team that performs maintenance on an evolutionary software web.

Keywords: Software maintenance, Customization process , Agile methods

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AM	Agile Modeling
ASD	Adaptive Software Development
BPM	Business Process Management
BPMN	Business Process Modeling Notation
CM	Crystal Methods
CMMI	Capability Maturity Model Integration
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
GQM	Goal-Question-Metric
ISO/IEC	International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC).
LD	Lean Development
MPS.BR	Modelo de Processo de Software Brasileiro
NBR	Norma Brasileira
SEI	Software Engineering Institute
TCC	Trabalho de Conclusão de Curso
XP	Extreme Programming

SUMÁRIO

1 . INTRODUÇÃO

A crescente demanda por sistemas e a alta velocidade com que seus requisitos evoluem têm evidenciado que desenvolvimento de *software* exige flexibilidade em seu processo. As dificuldades para a produção de sistemas vão muito além das questões técnicas. Fatores estratégicos, comerciais e humanos são responsáveis por algumas das variáveis que contribuem para tornar o desenvolvimento de sistemas de *software* uma atividade altamente complexa.

Uma questão muito importante a ser considerada quando se pensa em definir uma metodologia de desenvolvimento de *software* é a evolução do sistema. Pois o desenvolvimento de *software* não pára quando o sistema é entregue para operação, pelo contrário, os sistemas devem inevitavelmente mudar para que permaneçam úteis. A evolução do *software* é imprescindível para as organizações que são dependentes de seus sistemas, os quais são considerados importantes ativos de negócio. SOMMERVILLE (2007), afirma que a maior parte do orçamento nas grandes empresas é destinada à manutenção de sistemas existentes.

De acordo com a norma internacional NBR ISO/IEC 12207 (1998), a manutenção faz parte do ciclo de vida do *software*, portanto o processo de desenvolvimento de *software* continua sendo utilizado para realizar a manutenção do *software.* Segundo SOMMERVILLE (2007), a manutenção de *software* é um processo geral de mudanças de um sistema depois que ele é entregue, onde existem três tipos diferentes de manutenção: manutenção corretiva, manutenção adaptativa e manutenção evolutiva.

A manutenção corretiva é universalmente conhecida por se referir à manutenção de reparo de defeitos de *software*. Compreende a correção de erros do sistema. A manutenção adaptativa compreende a adaptação do sistema às mudanças do ambiente, como *hardware* ou plataforma do sistema operacional. E a manutenção evolutiva refere-se ao aperfeiçoamento do sistema, onde os requisitos do *software* mudam em resposta à mudanças organizacionais ou de negócios.

Para as organizações, a escolha da metodologia mais adequada para o desenvolvimento de *software* não é uma tarefa trivial, levando em consideração os inúmeros fatores envolvidos. Por outro lado, é um fator preponderante para o sucesso da organização. Embora um bom processo não possa garantir o sucesso de um projeto, certamente a adoção de

um processo inadequado pode comprometê-lo (BANKI e TANAKA, 2008). Segundo Sommerville (2007), não existe processo ou metodologia ideal, portanto é preciso escolher um conjunto de práticas de desenvolvimento adequado às características do projeto e da equipe. Desta forma, a natureza única de cada projeto e a necessidade de alta qualidade e produtividade tornam importante a busca por práticas de desenvolvimento.

Sommerville (2007) afirma que há uma relação estreita entre a qualidade de um processo de desenvolvimento e a qualidade de produtos desenvolvidos por meio deste processo. E é devido a esse fato que muitas empresas tem buscado o aprimoramento de seus processos de desenvolvimento de *software*.

A customização ou aprimoramento de processos não significa simplesmente adotar métodos específicos, ferramentas ou algum modelo de processo padrão. Existem muitos fatores organizacionais, culturais que influenciam no processo de desenvolvimento de cada organização. A customização de um processo sempre deve ser encarada como algo específico para cada organização (SOMMERVILLE, 2007).

Geralmente as organizações quando buscam a melhoria de seus processos de desenvolvimento procuram se apoiar em métodos já conhecidos e utilizados. Diante desse fato, são considerados como modelos de metodologias conhecidas as chamadas “tradicionais” e “ágeis”.

Nas metodologias tradicionais, conhecidas também como “pesadas” ou orientadas a planejamentos, Fowler (2005) afirma que o objetivo é desenvolver um processo onde as pessoas envolvidas são partes substituíveis, ou seja, os indivíduos não são tão importantes, somente suas funções.

Já na visão de Ambler (2004), os métodos tradicionais são considerados “pesados” pela necessidade de uma grande quantidade de documentação. Além disso, são resistentes a mudanças, possuindo um ciclo de desenvolvimento pouco adaptável e que deve ser completamente executado.

Por outro lado, os Métodos Ágeis de desenvolvimento de *software* sugerem uma abordagem mais humanística, com foco na entrega rápida e constante de *software* com valor de negócios (FOWLER, 2005). Segundo BECK (2004), são “métodos leves para pequenas e médias equipes, desenvolvendo *software* com requisitos vagos ou com mudanças rápidas nos requisitos”. E, ainda conforme BECK (2004), “apresentam uma alternativa para documentação no desenvolvimento de *software*” surgindo como uma reação aos modelos

tradicionais de desenvolvimento de *software*, como o RUP (*Rational Unified Process*).

Conforme Pressman (2006), "em essência, os métodos ágeis foram desenvolvidos em um esforço para vencer as fraquezas percebidas e reais da engenharia de *software* convencional".

As metodologias ágeis surgiram nos anos 90 devido à necessidade de preencher os espaços deixados pelos métodos tradicionais. Os métodos ágeis que foram chamados de "leves", foram a alternativa para projetos cujos requisitos mudam frequentemente.

Estas metodologias surgiram para suprir a necessidade de um processo de desenvolvimento de *software* cada vez mais ágil e adaptável às frequentes modificações do mercado. Neste intuito, utilizam princípios e práticas que buscam tratar de forma diferenciada o processo de mudança dos *software*s, visando reagir rapidamente a essas mudanças, isto é, ser adaptável.

Em 2001, um grupo de pesquisadores autodenominados de Aliança Ágil (*Agile Alliance*), entre eles Martin Fowler, Alistair Cockburn, Jim Highsmith, Kent Beck, Mike Beedle e outros, motivados pela suas experiências em desenvolvimento de *software*, iniciaram uma discussão sobre como desenvolver *software* de forma mais rápida e eficaz, orientado principalmente à simplicidade. Esta discussão resultou no chamado Manifesto Ágil, que em síntese aponta:

> Desejamos descobrir melhores caminhos para desenvolver *software* fazendo e ajudando outros a fazerem. Valorizamos os indivíduos e interações através de processos e ferramentas; O desenvolvimento de *software* deve possuir uma documentação compreensiva; A colaboração do cliente e respostas às mudanças através de um plano específico. (AGILE MANIFESTO, 2010).

É importante observar que ser ágil não quer dizer ser radical e nem quer dizer que existe uma única solução para projetos de desenvolvimento de *software*. De acordo com Jacobson (2002), "agilidade tornou-se atualmente uma palavra mágica quando se descreve um processo moderno de *software*. Tudo é ágil. Uma equipe ágil é uma equipe esperta, capaz de responder adequadamente a modificações".

Um ponto bastante favorável aos métodos ágeis são as aplicações baseadas na *web* (*world wide web*), que têm evoluído constantemente, e os sistemas voltados à *web* causaram um grande impacto na história da computação. Seu crescimento vem se evidenciando cada

vez mais e, com isso, as aplicações *web* precisaram de mais atenção no âmbito do desenvolvimento de *software* (PRESSMAN, 2006).

Para que se tenha qualidade nos sistemas que são construídos orientados à *web*, esse desenvolvimento deve ser planejado considerando-se as características singulares desses sistemas e, portanto, demandam metodologias de desenvolvimento apropriadas (PRESSMAN,2006).

Como a *web* é um ambiente de desenvolvimento dinâmico e com mudanças constantes, as metodologias tradicionais orientadas a documentação são menos adequadas que as metodologias ágeis. Dessa forma, as metodologias ágeis vem despertando atenção crescente do mercado (PRESSMAN, 2006).

1.1 Problema de Pesquisa

Cada processo ou metodologia para desenvolvimento de *software* possui características próprias. Processos de *software* são formados por estruturas complexas e necessitam de julgamento humano para a sua utilização. Existem vários processos de desenvolvimento, o que dificulta a sua automatização, fazendo com que muitas empresas desenvolvam processos diferentes para as suas necessidades. O fato de não existir um processo ideal, possibilita que sejam feitas melhorias nos processos existentes (SOMMERVILLE, 2007).

Muitas empresas que se dedicam ao desenvolvimento de *software* investem na melhoria dos seus processos de desenvolvimento. Segundo Sommerville (2007), a melhoria dos processos de *software*, significa compreender os processos existentes e modificá-los para que possam atender as necessidades relativas à qualidade do produto, custo e tempo de desenvolvimento.

Considerando que o desenvolvimento de sistemas voltados para *web* sofre mudanças constantes em seus requisitos e que normalmente possuem um prazo curto de entrega, é importante fazer com que o desenvolvimento desses *softwares* se torne ágil.

Quando essas empresas possuem equipes pequenas responsáveis pela criação e manutenção de seus *softwares* próprios específicos[1], voltados para *web,* percebe-se uma grande necessidade de métodos de trabalho rápidos e enxutos, para aumentar a agilidade do

1 *Softwares* que atendem necessidades específicas de uma empresa diferentemente dos pacotes comerciais.

processo de desenvolvimento, o que vai refletir na qualidade e manutenção do produto final, o *software*.

Foi considerando este cenário de equipes pequenas, formadas por analistas de sistemas e programadores, que trabalham na manutenção evolutiva de sistemas *web*, que identificou-se a necessidade de customizar uma metodologia de desenvolvimento de *software.* Tais equipes precisam de métodos de trabalho que propiciem melhor controle das mudanças, melhor comunicação entre equipe técnica e maior agilidade.

1.2 Questão de Pesquisa

Baseando-se nas características do problema descrito, foi elaborada a seguinte questão de pesquisa: Como proporcionar maior agilidade ao processo de desenvolvimento de *software web* com orientação a objetos em equipes pequenas que realizam manutenção evolutiva?

1.3 Objetivos

Este trabalho tem por objetivo propor uma metodologia de desenvolvimento de *software* baseado na modelagem ágil, a ser utilizada para a manutenção evolutiva de *softwares.* O foco será nas necessidades de equipes pequenas, de até 10 pessoas, formadas por analistas de sistemas e programadores, que desenvolvem *softwares* específicos para *web* utilizando orientação a objetos.

Para Pressman (2006), a metodologia de desenvolvimento de *software* engloba o processo, os métodos e as ferramentas. O processo é definido como um *framework* para as tarefas que são necessárias para a construção de *software* de alta qualidade. Os métodos fornecem a "técnica" de como fazer para construir *softwares*. E as ferramentas são mecanismos automatizados de apoio ao processo e aos métodos.

Para apoiar a metodologia proposta, será disponibilizado um ambiente com uma ferramenta para a análise e projeto, a serem utilizadas pela equipe técnica para a modelagem do sistema e sua comunicação.

A avaliação do modelo proposto será realizada através de um estudo de caso utilizando a metodologia proposta.

1.4 Metodologia e organização do trabalho

Este trabalho foi desenvolvido através da abordagem qualitativa utilizando o método de pesquisa exploratória. Segundo Lakatos e Marconi (2005), os estudos exploratórios ou formuladores têm por objetivos principais, desenvolver, esclarecer e modificar conceitos e ideias, visando a formulação de problemas mais precisos ou a criação de hipóteses. Também podem ser usados para aumentar o conhecimento do pesquisador acerca de um fenômeno a ser investigado em um estudo posterior, mais estruturado. Habitualmente envolvem levantamento bibliográfico e documental, entrevistas não padronizadas e estudos de caso (GIL, 1999).

Para a execução do trabalho foram definidas as seguintes atividades:

- Realizar estudos bibliográficos sobre manutenção de *software*, customização de processos de desenvolvimento de *software* e métodos ágeis de desenvolvimento de *software*.
- Definir a metodologia de customização de processos de desenvolvimento de *software* baseando-se nos estudos bibliográficos. A customização deverá ser aplicada em uma equipe específica que possua as características apresentadas no problema de pesquisa.
- Realizar o estudo do processo atual de desenvolvimento de *software* utilizado pela equipe na manutenção evolutiva, aplicando a metodologia de customização definida. Esse estudo deve compreender a definição dos objetivos da customização, a análise do processo atual, a apuração dos resultados dessa análise, e por fim a identificação dos aprimoramentos a serem realizados de forma a atender os objetivos da customização.
- Sistematizar uma nova metodologia de desenvolvimento de *software* que atenda aos objetivos da customização e as necessidades da equipe que está sendo estudada.
- Definir um processo de avaliação de ferramentas que será utilizada para selecionar a ferramenta que irá apoiar a metodologia proposta.
- Realizar a avaliação de ferramentas existentes de acordo com os critérios definidos no processo de avaliação. Essa avaliação deve estar baseada nas necessidades da equipe e da metodologia proposta.
- Disponibilizar um ambiente de ferramentas que apoie o modelo proposto.
- Aplicar um estudo de caso utilizando o modelo proposto.

- Realizar a avaliação do modelo proposto.

O trabalho está organizado em 10 capítulos, onde no capítulo 1 é apresentada a introdução composta pela contextualização do assunto, problema de pesquisa, questão de pesquisa, objetivos e metodologia do trabalho.

No capítulo 2 são apresentados os estudos bibliográficos realizados sobre manutenção de *software* e customização de processos de desenvolvimento de *software*. Este estudo possibilitou a definição da metodologia de customização de processo que foi aplicada em uma equipe específica que possui as características apresentadas no problema de pesquisa (descrito na seção 1.1, página 14).

O capítulo 3 apresenta os estudos bibliográficos sobre os métodos ágeis de desenvolvimento de *software*. Este estudo compõe o embasamento teórico para a criação da proposta de customização do processo estudado, baseada em práticas ágeis de desenvolvimento.

No capítulo 4 é apresentado o estudo do processo atual que foi executado de acordo com a metodologia de customização que foi definida no capítulo 2 e utilizada no trabalho. Neste capítulo são apresentados os dados de medição do processo seguindo a abordagem Objetivo - Questão - Métrica (GQM – *Goal-Question-Metric*), as técnicas utilizadas para análise do processo atual assim como os resultados obtidos, e por fim um modelo do processo de desenvolvimento atual.

O capítulo 5 apresenta o novo processo customizado baseado nos aprimoramentos identificados no capítulo 4. O novo modelo de processo proposto tem por objetivo promover a customização do processo de forma a atender objetivos organizacionais traçados na medição do processo (GQM).

O capítulo 6 descreve o processo de avaliação de ferramentas que foi executado, com base nas normas NBR ISO/IEC 9126 e NBR ISO/IEC 14598, para a seleção da ferramenta que foi utilizada na metodologia de desenvolvimento proposta neste trabalho.

No capítulo 7 é apresentado o estudo de caso que foi aplicado a metodologia de desenvolvimento de *software* proposta neste trabalho através da customização do processo de uma equipe específica.

No capítulo 8 é apresentada a conclusão do trabalho que descreve os objetivos alcançados, as lições aprendidas, os problemas enfrentados, assim como as sugestões de trabalhos futuros.

E por fim, nos capítulos 9 e 10, são apresentados, respectivamente, o referencial bibliográfico e os anexos que compõem este trabalho.

2. CUSTOMIZAÇÃO DE PROCESSOS DE MANUTENÇÃO DE SOFTWARE

Este capítulo tem por objetivo apresentar o estudo bibliográfico realizado sobre manutenção de *software* e customização de processos de desenvolvimento de *software*.

Na seção 2.1 são apresentados os conceitos de manutenção de *software,* os diferentes tipos e causas de manutenção e a sua importância no ciclo de vida do *software*.

Já na seção 2.2 é apresentado o estudo sobre customização de processo de *software* e a metodologia para customização que foi utilizada neste trabalho.

2.1 Manutenção de *Software*

De acordo com a norma NBR ISO/IEC 12207 - Tecnologia da Informação - Processos de Ciclo de Vida de *Software* (ABNT, 1998), a manutenção faz parte do ciclo de vida do *software*. Ainda, segundo essa norma, o objetivo do processo de manutenção é modificar um produto de *software* existente, preservando sua integridade (ABNT, 1998). O processo é ativado quando o produto de *software* é submetido a modificações, a partir do momento em que se necessita efetuar modificações no código e na documentação do sistema devido a um problema, adaptação ou necessidade de melhoria. O processo de manutenção chega ao seu final no momento da descontinuação do *software*, ou seja, quando não se vai mais utilizá-lo.

A evolução do *software* ocorre com o tempo, independente do tamanho, complexidade ou domínio de aplicação. As modificações, chamadas de manutenção de *software*, orientam esse processo de evolução e ocorrem quando erros são corrigidos, quando o *software* é adaptado a um novo ambiente, quando o cliente solicita novas funcionalidades e quando a aplicação é repensada ou reprojetada para fornecer benefícios em um contexto moderno (PRESSMAN, 2006).

A manutenção de *software* é um conceito antigo e extremamente importante e que, muitas vezes, não tem sua importância reconhecida (PRIKLADNICKI, 2003). Por muito tempo era uma fase negligenciada do processo de desenvolvimento de *software*. Na prática, a falta de controle e disciplina nas atividades de desenvolvimento quase sempre se traduz em problemas durante a manutenção do *software* (PRESSMAN, 2006). Entre os diversos problemas, Pressman (2006) destaca os principais, entre eles:

• É difícil e às vezes impossível rastrear a evolução através de muitas versões ou lançamentos. As mudanças não estão corretamente documentadas;

• É difícil rastrear o processo através do qual o *software* foi desenvolvido.

• A documentação não existe ou é de má qualidade. Reconhecer que um *software* deve ser documentado é o primeiro passo, mas toda a documentação gerada deve estar de acordo com a lógica adotada.

• A maioria dos *softwares* não é projetada para sofrer mudanças.

• A manutenção não é considerada importante. Grande parte desta percepção vem do elevado nível de frustração associado ao trabalho de manutenção.

Todos estes problemas podem ser atribuídos ao grande número de sistemas existentes que foram desenvolvidos sem levar em consideração alguma metodologia, regras, etc. De um modo geral, a engenharia de *software* atualmente oferece algumas soluções para os problemas associados à manutenção (PRESSMAN, 2006).

Manutenção de *software* deve, portanto, ser entendida como o processo de introdução de alterações no sistema, após a sua entrega aos usuários e subsequente entrada em operação. As modificações envolvidas no processo podem ser simples (quando se trata de introduzir pequenas modificações, alterando a codificação executada no estágio de implementação), extensas (quando se trata de corrigir erros de projetos, cometidos no estágio correspondente) e drásticas (quando se trata de corrigir erros de especificação cometidos no estágio de definição) (ROCHA, 1987).

Segundo Sommerville (2007), a manutenção de *software* é um processo geral de mudanças de um sistema depois que ele é entregue, onde existem três tipos diferentes de manutenção: manutenção corretiva, manutenção adaptativa e manutenção evolutiva.

A manutenção corretiva é universalmente conhecida por se referir à manutenção de reparo de defeitos de *software*. Compreende a correção de erros do sistema que até então não haviam sido evidenciados.

A manutenção adaptativa compreende a adaptação do sistema às mudanças do ambiente, como hardware ou plataforma do sistema operacional.

E, a manutenção evolutiva refere-se ao aperfeiçoamento do sistema, onde os requisitos do *software* mudam em resposta à mudanças organizacionais ou de negócios. Este tipo compreende as mudanças pós-implantação que, na maioria das vezes, são consequência de novos requisitos gerados em resposta as mudanças nos negócios ou necessidades dos usuários.

Para Pressman (2006) existem quatro tipos de manutenções, além das manutenções corretiva e adaptativa citadas por Sommerville (2007), Pressman (2006) define mais duas, a manutenção perfectiva ou de melhoria e a manutenção preventiva ou reengenharia.

Manutenção perfectiva ou de melhoria são mudanças normalmente sugeridas pelo usuário ou pela equipe de programação ao desenvolvedor, visando a otimização do *software*. Equivalente a manutenção evolutiva de Sommerville (2007).

Manutenção preventiva ou reengenharia consiste em modificar o *software* para melhorar a confiabilidade ou a manutenibilidade futura.

É impossível produzir-se *software,* de qualquer tamanho, imune à necessidade de introdução de alterações. Assim sendo, nada mais sensato do que procurar antecipar-se aos fatos, por ocasião da construção do *software*, como um esforço no sentido de torná-lo suscetível à manutenção, o que acarretará na redução dos custos futuros de manutenção e, consequentemente, dos custos operacionais.

Software é suscetível à manutenção à medida que, desde a sua concepção, tenha sido especificado e desenvolvido com tal propósito. Para tanto, a manutenibilidade deverá ser perseguida desde o início do desenvolvimento.

A manutenção de *software* existente pode ser responsável por mais de 60% de todo o esforço despendido por uma organização de desenvolvimento, e a porcentagem continua a crescer à medida que mais *software* é produzido (PRESSMAN, 2006).

Uma das razões para o problema de manutenção de *software* é a mobilidade do pessoal de *software*. Provavelmente a equipe (ou pessoas) que realizou o trabalho original não está mais presente. Ou até pior, gerações subsequentes de pessoal de *software* que realizaram modificações também não estão mais presentes. Isso é um problema para as equipes atuais que precisam realizar modificações no *software* sem o devido conhecimento sobre o mesmo.

Modificação é inevitável quando sistemas baseados em computador são construídos; consequentemente, precisamos desenvolver mecanismos para avaliar, controlar e realizar modificações.

Segundo Pressman (2006), manutenção de *software* é, sem dúvida, muito mais que "consertar erros". Apenas cerca de 20% de todo o trabalho de manutenção é gasto "consertando erros". Os restantes 80% são gastos adaptando sistemas existentes a modificações no seu ambiente externo, fazendo melhorias solicitadas por usuários e submetendo uma aplicação a reengenharia, para uso futuro. Quando a manutenção é

considerada como abrangendo todas essas atividades, é relativamente fácil ver porque absorve tanto esforço.

A evolução do *software* é imprescindível para as organizações que são dependentes de seus sistemas, os quais são considerados importantes ativos de negócio. SOMMERVILLE (2007), afirma que a maior parte do orçamento nas grandes empresas é destinada à manutenção de sistemas existentes.

O processo de desenvolvimento de *software* continua durante a manutenção, e exige maior agilidade visto que o *software* já está em operação. E é necessário manter o *software* funcionando e entregar as manutenções de forma que não interrompam o funcionamento do *software.* Por esse motivo, destaca-se uma grande importância para a gestão de mudanças que deve acontecer durante a fase de manutenção do *software* assim como

2.2 Customização de processo de desenvolvimento de *software*

Segundo Sommerville (2007), há uma relação estreita entre a qualidade de um processo de desenvolvimento e a qualidade de produtos desenvolvidos por meio deste processo. E é devido a esse fato que muitas empresas tem buscado o aprimoramento de seus processos de desenvolvimento de *software*.

Sommerville (2007) afirma que o aprimoramento (ou customização) de processos não significa simplesmente adotar métodos específicos, ferramentas ou algum modelo de processo padrão. Existem muitos fatores organizacionais, culturais que influenciam no processo de desenvolvimento de cada organização. A customização de um processo sempre deve ser encarada como algo específico para cada organização.

Existem algumas normas que podem auxiliar na melhoria de processos, como a ISO/IEC 15504-*Process Assessment*, em português Processo de Avaliação (ISO, 1998), e o MPS.BR- Modelo de Processo de *Software* Brasileiro (SOFTEX, 2009).

A norma ISO/IEC 15504 define um modelo bidimensional que tem por objetivo a realização de avaliações de processos de *software* com o foco da melhoria dos processos (gerando um perfil dos processos, identificando os pontos fracos e fortes, que serão utilizados para a elaboração de um plano de melhorias) e a determinação da capacidade dos processos viabilizando a avaliação de um fornecedor em potencial (ISO, 1998). Segundo Rocha, Maldonado e Weber (2001), quando o objetivo for a melhoria dos processos, a organização

pode realizar a avaliação gerando um perfil dos processos que serão usados para a elaboração de um plano de melhorias. Porém, a organização deve definir os objetivos e o contexto, bem como escolher o modelo e o método para a avaliação e definir os objetivos de melhoria.

O MPS.BR (SOFTEX, 2009) foi desenvolvido a partir das normas NBR ISO/IEC 12207 (NBR ISO/IEC 12207, 1998), ISO/IEC 15504 (ISO, 1998) e CMMI-DEV - *Capability Maturity Model Integration for Development* (SEI, 2006). Segundo Softex (2009), modelo MPS.BR baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de *software* e serviços correlatos.

O modelo CMMI - *Capability Maturity Model Integration* foi publicado em 2002 com o objetivo de integrar os diversos modelos de maturidade até então publicados pelo SEI - *Software Engineering Institute*, além de incorporar lições aprendidas ao longo de vários anos de utilização. Dois conceitos fundamentais que permeiam a concepção dos modelos do SEI são as ideias de capacidade e maturidade de processo.

A capacidade descreve o intervalo de resultados esperados que podem ser atingidos através do uso de um processo, e assim provê um meio de predizer os resultados típicos dos projetos a serem desenvolvidos pela organização.

A maturidade representa o potencial para o crescimento da capacidade de uma organização e indica tanto a riqueza do processo como a consistência com a qual ele é aplicado no contexto organizacional (SEI, 2010). O modelo CMMI está estruturado em cinco níveis de maturidade e suas áreas de processo estão distribuídas nas categorias de gerência de projetos, gerência de processos, engenharia e suporte. Mais detalhes podem ser obtidos através de http://www.sei.cmu.edu/cmmi/.

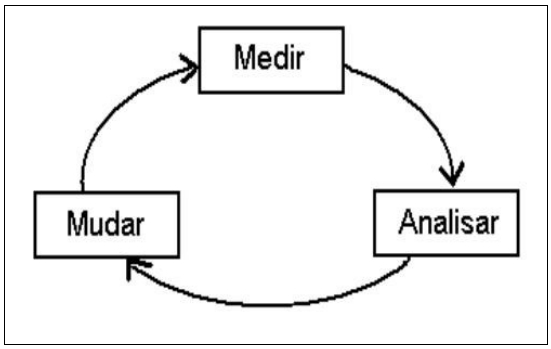
Além das normas citadas, existem abordagens de autores da área de engenharia de *software*, como por exemplo Sommerville (2007) que utiliza o termo "aprimoramento de processo" para referir-se a melhoria ou customização de processo.

Sommerville (2007) define o aprimoramento do processo como "compreender os processos existentes e alterá-los para melhorar a qualidade do produto e/ou reduzir custos e tempo de desenvolvimento". Ainda coloca que o aprimoramento não significa simplesmente adotar métodos específicos ou ferramentas, ou modelos de processos utilizados em outros lugares. Existem, fatores organizacionais, culturais, procedimentos e padrões que variam de organização para organização, o que implica em realizar a customização do processo considerando as características específicas da organização.

A customização de processos é uma atividade cíclica que segundo Sommerville (2007) compreende três estágios: medição de processo, análise do processo e mudança de processo.

- **Medir:** corresponde a **Medição de processo**, é o estágio onde são medidos os atributos do projeto ou produto atual. O objetivo é aprimorar as medidas de acordo com os objetivos da organização envolvida no aprimoramento de processos.
- **Analisar:** corresponde a **Análise do processo,** é onde o processo atual é analisado e modelado, bem como seus pontos fortes e fracos. Após a análise do processo atual é gerado o modelo do processo atual, com a identificação das mudanças a serem realizadas.
- **Mudar:** corresponde a **Mudança de processo**, é onde as mudanças que foram identificadas no estágio de análise são introduzidas.

Na figura 1 é apresentado o ciclo de aprimoramento do processo sugerido por Sommerville (2007).



Fonte: SOMMERVILLE (2007)

Figura 1: Ciclo de aprimoramento do processo

Abaixo serão detalhados os três estágios: medição de processo, análise do processo e mudança de processo.

a) Medição de processo

Sommerville (2007) define que as medições de processo são dados quantitativos sobre o processo de *software*. Classifica em três classes de métricas: o tempo total de execução do processo, os recursos necessários para a execução de um processo e o número de ocorrências de um evento específico, como por exemplo o número de defeitos descobertos durante uma inspeção de código.

Ainda, Sommerville (2007) afirma que a dificuldade da medição é saber o que medir

e coloca que o aprimoramento de processo deve ser guiado por objetivos. Para a definição desses objetivos, Sommerville (2007) sugere a abordagem "Objetivo-Questão-Métrica", também conhecido como *Goal-question-Metric* (GQM)[2]. Essa abordagem compreende a identificação de objetivos, questões e métricas.

- **Objetivos:** correspondem ao que a organização está tentando obter e devem ter alinhamento com o negócio. Um exemplo de objetivo citado por Sommerville (2007) é o aumento da produtividade de programadores. Os objetivos devem orientar as mudanças de processo.
- **Questões:** são refinamentos dos objetivos, em que áreas específicas de incertezas relacionadas aos objetivos são identificadas. Um objetivo pode possuir uma ou várias questões, que são as perguntas que devem ser respondidas para saber se o objetivo foi alcançado. Um exemplo de questão citada por Sommerville (2007) é como o número de linhas de código podem ser aumentadas?
- **Métricas:** são as medições que necessitam serem coletadas para auxiliar a responder as perguntas (questões), e consequentemente, confirmar se o objetivo foi alcançado. Um exemplo de métrica citada por Sommerville (2007) é a produtividade de programadores individuais por linha de código.

A vantagem dessa abordagem aplicada ao aprimoramento de processos é que ela separa assuntos organizacionais (os objetivos) de processos específicos (as questões).

b) Análise do processo

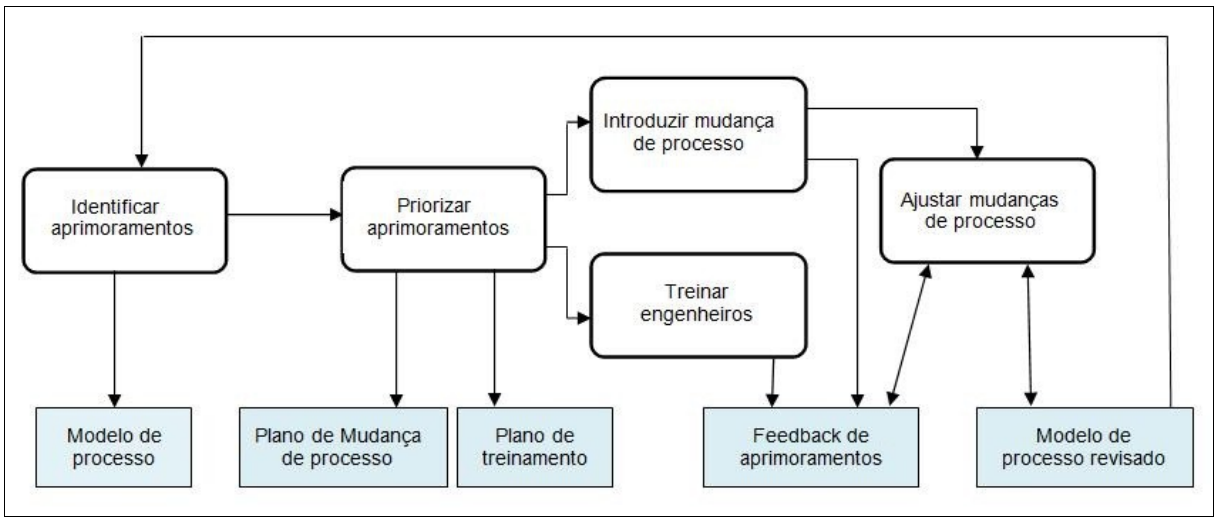
A análise do processo corresponde ao estudo do processo atual e o desenvolvimento de um modelo abstrato do processo que apresente as características principais. O objetivo é compreender os relacionamentos entre as partes do processo. Sommerville (2007) coloca que o estágio inicial de análise é qualitativo, onde o analista está simplesmente tentando descobrir as características do modelo. Os estágios posteriores podem ser quantitativos e fazem uso das medições de processo coletadas. O ponto inicial para a análise do processo deve ser o modelo formal de processo usado, mas raramente incluem detalhes suficiente. As técnicas de análise de processo sugeridas por Sommerville (2007) incluem questionários, entrevistas, observação e estudos etnográficos.

2 Paradigma proposto por Basili e Rombach (1988)

c) Mudança de processo

A mudança de processo envolve a realização de modificações no processo existente. Isso pode ser feito por meio da introdução de novas práticas, métodos ou ferramentas, mudança de ordem das atividades de processo, introdução ou remoção de entregas de processo ou introduzir novos papéis e responsabilidades (SOMMERVILLE, 2007). Na fase de medição do processo são estabelecidos objetivos para o aprimoramento do processo, e serão esses objetivos que irão guiar as mudanças.

De acordo com Sommerville (2007), existem cinco estágios no processo de mudança, que são eles: identificar aprimoramentos, priorizar aprimoramentos, introduzir mudança de processo, treinar a mudança de processo, e ajustar as mudanças de processo. A figura 2 mostra o processo de mudança de processo sugerido por Sommerville (2007):



Fonte: SOMMERVILLE (2007)

Figura 2: Processo de mudança de processo

A **identificação de aprimoramentos** consiste em analisar os resultados da análise do processo atual e identificar os pontos a serem melhorados para atingir os objetivos da mudança. Nesse estágio é proposto um novo modelo de processo que contemple as melhorias identificadas de forma a atingir os objetivos.

A **priorização dos aprimoramentos** corresponde a avaliação das possíveis mudanças e priorizá-las para serem implantadas. Geralmente quando são muitas mudanças propostas não são implementadas em uma única vez, dessa forma implementando primeiramente as mudanças priorizadas.

A **introdução da mudança de processo** consiste em colocar em práticas os novos procedimentos, métodos e ferramentas e integrá-los com as demais atividades do processo.

O objetivo do **treinamento das mudanças de processo** é para que a equipe toda tenha o entendimento e clareza no impacto em suas atividades e com isso gere as oportunidades de melhorias de fato, e consequentemente o aceite das mudanças pela equipe.

E por fim, a fase de **ajuste de mudanças** é necessária para que haja o acompanhamento das mudanças introduzidas e que ocorram as modificações necessárias quando descobertos problemas no novo processo. Essa fase deve durar alguns meses até que a equipe esteja satisfeita com o novo processo.

2.3 Considerações Finais

Nesse capítulo foi apresentado o referencial teórico sobre manutenção e customização de processo de *software*. O objetivo do estudo foi conhecer diferentes abordagens e adquirir conhecimento para subsidiar a definição da metodologia a ser utilizada para customizar processos de manutenção de *software*.

Através do estudo foi identificado o conceito e os diferentes tipos e causas de manutenção de *software*. Segundo a norma NBR ISO/IEC 12207 (ABNT, 1998), a manutenção faz parte do ciclo de vida do *software*, e o objetivo do processo de manutenção é modificar um produto de *software* existente, preservando sua integridade. Sommerville (2007), Pressman (2006) e Rocha (1987) classificam as manutenções em corretivas, adaptativas, evolutivas ou perfectivas e preventivas.

Este trabalho está focado na manutenção evolutiva, que refere-se ao aperfeiçoamento do sistema, onde os requisitos do *software* mudam em resposta à mudanças organizacionais ou de negócios. Esse tipo compreende as mudanças pós-implantação que, na maioria das vezes, são consequência de novos requisitos gerados em resposta às mudanças nos negócios ou necessidades dos usuários (SOMMERVILLE,2007; PRESSMAN,2006; ROCHA,1987).

Também foi realizado um estudo em melhoria de processos de *software*, onde foram estudadas as normas ISO/IEC 15504 (ISO, 1998), MPS.BR (SOFTEX,2009) e CMMI (SEI,2010). Por entender que as normas citadas são genéricas e mais facilmente aplicáveis a grandes equipes, buscou-se por uma abordagem de autores da área de engenharia de *software*, onde foi optado por utilizar as definições de Sommerville(2007), que define a melhoria ou

customização de processos como “aprimoramento de processo” e o define como uma atividade cíclica que passa por três estágios: medição de processo, análise do processo e mudança de processo.

O estágio de medição do processo se faz necessário para obter dados quantitativos sobre o processo que será customizado. Nesse estágio devem ser identificados os objetivos, questões e métricas da customização. Os objetivos da customização devem estar alinhados com as diretrizes de negócio da organização. As questões devem estar relacionadas aos processos específicos das áreas envolvidas e as métricas devem ser estabelecidas de forma que permitam obter dados para auxiliar a responder às questões e confirmar se os objetivos foram alcançados.

No estágio de análise do processo deve ser realizado um estudo do processo atual onde são capturadas informações que permitam desenvolver um modelo abstrato do processo compreendendo suas principais características. Segundo Sommerville (2007), a fase inicial desse estágio é qualitativo e sugere a aplicação de técnicas como entrevistas, etnografia, observação e questionários.

E por fim, o estágio de mudança de processo envolve a realização de mudanças no processo existente afim de atender os objetivos da customização. Sommerville (2007) define um processo de mudança de processos que compreende cinco fases que são:identificar aprimoramentos, priorizar aprimoramentos, introduzir mudança de processo, treinar a mudança de processo, e ajustar as mudanças de processo. Segundo Sommerville (2007), a fase de ajuste de mudanças deve durar meses até que a equipe esteja satisfeita com o novo processo, pois é necessário que haja o acompanhamento das mudanças introduzidas e que ocorram as modificações necessárias.

Sommerville (2007) define que a mudança de processo pode ser feita através da introdução de novas práticas, métodos ou ferramentas, alteração na ordem das atividades de processo, introdução ou remoção de entregas de processo ou introdução de novos papéis e responsabilidades. Ainda, Sommerville (2007) destaca que as mudanças sempre devem ser guiadas pelos objetivos definidos no estágio de medição.

No capítulo 4 estão descritos os estágios de medição e análise de processo, seguindo a abordagem proposta por Sommerville (2007), e no capítulo 5 é apresentada a proposta de customização do processo visando alcançar os objetivos definidos para a customização.

3 . MÉTODOS ÁGEIS

Este capítulo tem por objetivo apresentar o estudo bibliográfico sobre os métodos ágeis de desenvolvimento de *software*, abordando o histórico, conceitos, valores e princípios que norteiam esses métodos.

Este estudo fez-se necessário para compor o embasamento teórico para a proposta de customização do processo adotando algumas das práticas ágeis.

O capítulo inicia com uma apresentação sobre a origem dos métodos ágeis, segue por uma breve explicação dos métodos *Adaptive Software Development* (ASD), *Dynamic System Development Methods* (DSDM), *Crystal Methods* (CM), *Feature Driven Development* (FDD), *Lean Development* (LD), *Agile Modeling* (AM), *Extreme Programming* (XP) e *Scrum*. Nesses dois últimos foi realizado um estudo mais aprofundado pois foram os métodos que mais influenciaram este trabalho.

3.1 O Manifesto Ágil

Os Métodos Ágeis de desenvolvimento de *software* surgiram como uma reação aos métodos clássicos de desenvolvimento de *software* e da necessidade de se criar uma alternativa aos "processos pesados", caracterizados por Ambler (2004) pelo foco excessivo na documentação completa e resistentes às mudanças. Na definição de Pressman (2006), os métodos ágeis foram desenvolvidos em um esforço para vencer as fraquezas percebidas e reais da engenharia de *software* convencional. E para Fowler (2005), os Métodos Ágeis de desenvolvimento de *software* sugerem uma abordagem mais humanística com foco na entrega rápida e constante de *software* com valor de negócios.

O termo "Metodologias Ágeis" tornou-se popular em 2001 quando um grupo de criadores e representantes dos chamados Métodos Ágeis se reuniram com o objetivo de discutir alternativas aos modelos clássicos de desenvolvimento, e dessa reunião surgiu a criação da Aliança Ágil[3] e o estabelecimento do Manifesto para Desenvolvimento Ágil de *Software*, mais conhecido como Manifesto Ágil[4]. Este grupo era formado por dezessete especialistas em processos de desenvolvimento de *software* representantes dos métodos

3 Disponível em http://www.agilealliance.org
4 Disponível em http://www.agilemanifesto.org

Extreme Programming (XP), *Scrum*, *Adaptive Software Development* (ASD), *Dynamic Systems Development Method* (DSDM), *Crystal Methods* (CM), *Feature Driven Development* (FDD), *Lean Development* (LD), entre outros.

O Manifesto Ágil é um conjunto de valores e princípios que definem critérios para o desenvolvimento ágil de *software*.(AMBLER, 2004). Na tabela 1 são apresentados os quatros valores do Manifesto ágil.

Tabela 1: Valores do manifesto ágil

	Quatro valores do Manifesto Ágil:
1	**Indivíduos e interações** mais que processos e ferramentas
2	**Software em funcionamento** mais que documentação abrangente
3	**Colaboração com o cliente** mais que negociação de contratos
4	**Responder a mudanças** mais que seguir um plano

Fonte: AGILE MANIFESTO (2010)

Com o objetivo de melhor compreender as filosofias do Manifesto Ágil, os membros publicaram uma coleção de doze princípios aos quais os métodos de desenvolvimento ágil de *software* devem se adequar. Na tabela 2 são apresentados estes princípios.

Tabela 2: Doze princípios do Manifesto Ágil

	Princípios
1	Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2	Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3	Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4	Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5	Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6	O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7	Software funcionando é a medida primária de progresso.
8	Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9	Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10	Simplicidade (a arte de maximizar a quantidade de trabalho não realizado) é essencial.
11	As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12	Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Fonte: AGILE MANIFESTO (2010)

O Manifesto Ágil apresenta uma nova filosofia para o desenvolvimento de *software* e é importante destacar que não é contra os modelos clássicos de desenvolvimento, apenas

propôs uma abordagem mais humanística com foco na entrega rápida e constante de *software* com valor de negócios. Fowler (2005) mostra que certos valores continuam sendo importantes, como a modelagem de dados. A documentação continua fundamental, mas com a preocupação da não geração enorme de papéis que provavelmente nunca serão lidos. O planejamento deve ser utilizado, mas com a consciência de que existem limitações sobre qualquer plano. Os métodos ágeis são baseados nos princípios propostos pelo Manifesto Ágil (AGILE MANIFESTO, 2010) e podem ser considerados como um meio termo entre a ausência de processo e o processo exagerado, diferenciando-se dos métodos tradicionais basicamente em dois aspectos: são adaptativos ao invés de preditivos e são orientados às pessoas ao invés dos processos.

Pode-se dizer, então, que os Métodos Ágeis se caracterizam por serem incrementais, cooperativos, diretos e adaptativos. Incrementais, dadas as pequenas versões e ciclos rápidos de desenvolvimento; cooperativos, por estimular a proximidade com o cliente e a interação entre os programadores; diretos, pela simplicidade de aprendizado e de documentação; e, finalmente, adaptativos, pela habilidade de acomodar mudanças ao longo do projeto (ABRAHAMSSON *et al*, 2003; FOWLER, 2005).

Nos métodos ágeis, a comunicação utilizada entre os membros da equipe é realizada, principalmente, de maneira informal ao invés da comunicação formal e documentada (BOEHM, 2002); pessoas podem compartilhar ideias mais rapidamente através de conversas informais do que através da leitura e escrita de documentos. Além disso, o envolvimento dos clientes e usuários no processo de desenvolvimento é valorizado possibilitando que as dúvidas sobre o projeto sejam solucionadas e que as prioridades sejam ajustadas (HIGHSMITH; COCKBURN, 2001).

Existem vários métodos que podem ser classificados como ágeis, onde Highsmith (2002a) e Cohen, Lindvall e Costa (2003) destacam: *Adaptive Software Development* (ASD), *Dynamic Systems Development Method* (DSDM), *Crystal Methods* (CM), *Extreme Programming* (XP), *Feature Driven Development* (FDD), *Lean Development* (LD), e *Scrum*.

Dentre os métodos acima destacados, serão apresentados brevemente o ASD, DSDM, CM, FDD, LD, AM e explorados com maior detalhes os métodos XP e Scrum que serviram de base para a execução deste trabalho.

3.2 *Adaptive Software Development* (ASD)

O método ágil *Adaptive Software Development* (ASD), em português Desenvolvimento de *Software* Adaptativo, tem como principais características, segundo Highsmith (2002a), o desenvolvimento baseado em um processo iterativo e incremental, voltado para sistemas considerados teoricamente de grande porte e muitas vezes complexos. Vale ressaltar, que este método é um dos poucos indicados para sistemas considerados de "grande porte". A presença constante do cliente e a utilização da técnica conhecida como JAD (*Joint Application Development*), em português Aplicação Conjunta de Desenvolvimento, são outras características deste método.

Ainda segundo Highsmith (2002a), o ASD possui ciclos que duram de 4 a 8 semanas, divididos em 3 fases: Especular, Colaborar e Aprender. A fase conhecida como Especular tem como função principal fixar os prazos, determinar os objetivos e definir um plano para a execução dos trabalhos. A fase denominada de Colaborar tem como objetivo básico a construção ou desenvolvimento do projeto. Finalmente, a fase chamada de Aprender objetiva conhecer e melhorar o que foi desenvolvido durante a fase Colaborar.

3.3 Dynamic Systems Development Method (DSDM)

Dynamic Systems Development Method (DSDM) ou, Método de Desenvolvimento de Sistemas Dinâmicos em português, originou-se em 1994 na Inglaterra pela formação de um consórcio de organizações britânicas, tendo como objetivo fomentar a utilização de práticas alternativas no desenvolvimento de *software*, com atuação em diversos países como EUA, Suécia, Holanda, França e Índia. Dane Falkner[5] (presidente da organização Surgeworks nos EUA) e Arie van Bennekum[6], são alguns dos vários membros desse consórcio que divulgam as práticas do DSDM[7]

O DSDM é um método que aplica prototipagem a fim de garantir a entrega contínua de um produto tangível. A entrega dos produtos intermediários é determinada por faixas fixas de tempo. Highsmith (2002a), afirma que "nada é construído perfeitamente na primeira vez".

5 Dane Falkner: especializada em treinamento e consultoria em DSDM. Foi responsável pelo consórcio DSDM nos EUA em 2001.
6 Arie van Bennekum é sócio proprietário da Solvision. Mais informações em http://www.thevisionweb.nl/
7 Disponível em http://www.dsdm.org

Apresenta ainda, que os princípios que norteiam o DSDM são muito próximos com os apresentados no manifesto ágil, por isso, o DSDM pode ser considerado como um método ágil.

Considerando o domínio de aplicação, o DSDM é mais bem indicado em organizações do tipo comerciais/industriais do que em aplicações científicas. É utilizado para qualquer tamanho de projeto, onde a única condição para grandes projetos, é que seja dividida em pequenas partes para melhor execução das atividades.

Segundo dados da Cutter Consortium[8], o DSDM situa-se entre os quatro métodos ágeis mais utilizados e, é um dos métodos ágeis mais bem definidos e organizados, apresentando características similares ao RUP, além de "contar com fácil adequação a outros métodos ágeis como a XP". Porém, o DSDM pertence a um consórcio, e como tal, para ter acesso à documentação mais detalhada se faz necessário o cadastramento neste consórcio com pagamento de taxas mensais. Este é um dos grandes problemas do DSDM, pois dificulta o acesso e sua disseminação na comunidade de desenvolvimento de *software*.

3.4 Crystal Methods (CM)

O método ágil *Crystal Methods* (CM), ou Métodos Crystal, desenvolvido pela IBM através de Alistar Cockburn, não é considerado apenas um método ágil, mas sim, um conjunto de métodos. Isto porque seu desenvolvedor acredita que diferentes tipos de projeto requerem diferentes métodos. Ou seja, não é um *kit* pronto com métodos, mas sim um conjunto de exemplos ajustáveis de acordo com as circunstâncias.

Foi criado a partir da crença de que os principais obstáculos enfrentados no desenvolvimento de produtos recaíam sobre os problemas de comunicação, os *Crystal Methods* dão grande ênfase às pessoas, à comunicação, às interações, às habilidades e aos talentos individuais, deixando os processos em segundo plano (COHEN, LINDVALL E COSTA, 2003). Apesar da estrutura proposta servir como um guia dos processos mais adequados a uma determinada situação, nos *Crystal Methods*, a definição final dos processos a serem utilizados é responsabilidade da equipe de projeto. Mas duas regras principais são sempre seguidas: ciclos de desenvolvimento incrementais com duração de no máximo quatro meses e reuniões de reflexão que estimulam a colaboração entre

8 Mais informações em http://www.cutter.com/index.html

integrantes da equipe de projeto (COHEN, LINDVALL E COSTA, 2003). Na tabela 3 são apresentadas as características principais de projetos que utilizam *Crystal Methods*.

Tabela 3: Características de projetos que utilizam *Crystal Methods*

Característica	Valores sugeridos
Tamanho da Equipe	A família dos *Cristal Methods* acomoda equipes de qualquer tamanho, preferencialmente compostas por pessoas talentosas
Duração das iterações	Até 4 meses para projetos grandes e altamente críticos
Equipes distribuídas	Os *Cristal Methods* foram concebidos para atender ao conceito de equipes distribuídas.
Aplicações de alta criticidade	Os *Cristal Methods* atendem a projetos críticos, incluindo aqueles que envolvem risco de visa e de valores monetários.

Fonte: adaptado de COHEN, LINDVALL E COSTA (2003)

3.5 Feature Driven Development (FDD)

Feature Driven Development[9] (FDD), ou Desenvolvimento Dirigido à Característica, como outros métodos ágeis, foca na entrega de pequenas iterações com a presença de alguma característica tangível. O FDD foi desenvolvido por Peter Coad e Jeff De Luca, e uma das principais obras[10] foi publicada em 2002, após a utilização deste método com sucesso em uma instituição bancária, segundo Highsmith (2002a). É um método de desenvolvimento de *software* específico para aplicações críticas de negócio (DIAS, 2005).

Diferentemente de outros Métodos Ágeis, o FDD se baseia em processos bem definidos e que podem ser repetidos. Sua abordagem se concentra nas fases de projeto e construção, com maior ênfase na modelagem, em um ciclo de vida iterativo e também em atividades de gerenciamento de projetos (DIAS, 2005).

Highsmith (2002a) afirma que, por ser iterativo e com ciclo de vida curto, o FDD é indicado para o desenvolvimento de *software* onde os requisitos mudam rapidamente, como acontece no método ágil SCRUM. Afirma, ainda, que "trabalha melhor, para processos simples e bem definidos". FDD é indicado para projetos de tamanho médio, que envolvam entre 10 e 30 pessoas na equipe de desenvolvimento do projeto.

Os princípios base do FDD são apontados abaixo (HIGHSMITH, 2002a):

- Necessidade de se automatizar a geração de *software* para projetos de grande escala; Um processo simples e bem definido é fundamental;

9 Mais informações em http://www.featuredrivendevelopment.com/

10 Palmer, S.R, Felsing J.M. Feature-Driven Development, Upper Saddle River, NJ, Prentice Hall, 2002

- As etapas de um processo devem ser lógicas e óbvias para cada integrante da equipe de desenvolvimento;
- Bons processos atuam na retaguarda, permitindo que a equipe se dedique ao alcance dos resultados;
- Ciclos de vida curtos e iterativos são mais indicados.

Um projeto conduzido pelo método FDD possui as seguintes etapas: Desenvolvimento de um modelo geral; Construção da lista de funcionalidades; Planejamento por funcionalidades; Projeto e desenvolvimento por funcionalidades. Na tabela 4 são apresentadas as características de um projeto desenvolvido com FDD.

Tabela 4: Características de projetos que utilizam FDD

Característica	Valores sugeridos
Tamanho da Equipe	Variável de acordo com a complexidade das funcionalidades a serem desenvolvidas.
Duração das iterações	Até 2 semanas
Equipes distribuídas	O FDD foi criado para trabalhar com equipes múltiplas e, apesar não haver indicação formal para equipes distribuídas, é passível de adaptação.
Aplicações de alta criticidade	Indicado para desenvolvimento de software crítico

Fonte: adaptado de COHEN, LINDVALL E COSTA (2003)

3.6 Lean Development (LD)

Com raízes na indústria automotiva dos anos 80, em especial no Modelo Toyota de Produção, o *Lean Development* (LD) é considerado o Método Ágil com maior foco estratégico. Iniciado por Bob Charette, o *Lean Development* tem como principais objetivos reduzir em um terço o prazo, o custo e o nível de defeitos no desenvolvimento de *software*. Para tanto, requer um grande comprometimento da alta administração e uma predisposição a mudanças radicais (COHEN, LINDVALL e COSTA, 2003).

Highsmith (2002a) aponta como princípios fundamentais do *Lean Development:*

- A satisfação do cliente é a prioridade principal;
- Prover sempre o maior valor possível para o dinheiro;
- O sucesso depende da participação ativa dos clientes;
- Todo projeto baseado no LD requer um esforço conjunto de toda a equipe;
- Tudo pode ser mudado;
- Domine, não aponte as soluções;

- Complete, não desenvolva;
- Prefira uma solução a 80% hoje, a uma solução a 100% amanhã;
- O minimalismo é essencial;
- A necessidade determina a tecnologia;
- O incremento do produto corresponde a um incremento de funcionalidade e não de tamanho;
- Nunca force o LD além de seus limites.

Cohen, Lindvall e Costa (2003) afirmam que "[...] como o *Lean Development* é mais uma filosofia de gerenciamento que um processo de desenvolvimento", os itens referentes ao tamanho da equipe, à duração das iterações, ao tratamento de equipes centralizadas ou distribuídas e à criticidade da aplicação não são diretamente endereçados pelo método.

3.7 Agile Modeling (AM)

A Modelagem Ágil foi introduzida por Scott W. Ambler em 2002 e tem como objetivo principal a geração de modelos e documentação eficazes que deem suporte ao desenvolvimento de sistemas, sem que o processo perca a agilidade (FAGUNDES, 2005).

Segundo Martins (2007), a AM é uma abordagem para a modelagem e não uma metodologia para desenvolvimento de *software*, onde normalmente é utilizado como um complemento para metodologias como XP, FDD, dentre outras. A filosofia da AM pode ser resumida como "modelar somente o necessário e nada mais", sempre usando as ferramentas mais simples e apropriadas para cada caso.

Ambler (2004) define AM como um "conjunto de práticas guiado por princípios e valores para profissionais de *software* aplicarem em seu dia-a-dia". A AM é uma atitude, não um processo prescritivo, ou seja, não define procedimentos detalhados sobre como criar um determinado tipo de modelo. Mas fornece conselhos sobre como modelar de forma eficiente. Ainda conforme Ambler(2004), a modelagem é uma parte importante de qualquer processo de desenvolvimento. Métodos ágeis como XP, Scrum, DSDM incluem atividades de modelagem.

Na próxima página, as tabelas 5, 6 e 7 mostram respectivamente os valores, princípios e práticas da AM.

Tabela 5: Valores da Modelagem Ágil (AM)

Valores da AM
- Comunicação
- Simplicidade
- Retorno
- Coragem
- Humildade

Fonte: AMBLER (2004)

Tabela 6: Princípios Básicos e Suplementares da AM

Princípios Básicos da AM	Princípios Suplementares da AM
- O software é seu objetivo principal	- Conteúdo é mais importante que a forma
- Possibilitar o próximo trabalho é seu objetivo secundário	- Todos podem aprender com todos
- Diminua a carga de trabalho	- Conheça seus modelos
- Adote a simplicidade	- Adaptação local
- Encampe a mudança	- Comunicação aberta e honesta
- Mude incrementalmente	- Trabalhe com o instinto das pessoas
- Modele com um propósito	
- Tenha mais de um modelo	
- Incentive o trabalho de qualidade	
- Maximize o retorno que seus clientes obterão	

Fonte: AMBLER (2004)

Tabela 7: Práticas básicas e suplementares da AM

Práticas básicas	Práticas suplementares
1. Modelagem iterativa e incremental	**1. Produtividade**
- Aplique o(s) artefato(s) correto(s)	- Aplique as convenções (standards) da modelagem
- Crie diversos modelos em paralelo	- Utilize os padrões (patterns) com moderação
- Itere em em outro artefato	- Reuse os recursos já existentes
- Modele incrementalmente	
	2. Documentação
2. Trabalho de equipe	- Descarte os modelos temporários
- Modele com outras pessoas	- Formalize os modelos de contrato
- Organize uma participação ativa	- Atualize apenas quando necessário
- Promova a posse coletiva	
- Mostre os modelos publicamente	**3. Motivação**
	- Modele para entender
3. Simplicidade	- Modele para comunicar
- Crie conteúdo simples	
- Mostre os modelos de modo simples	
- Use as ferramentas mais simples	
4. Validação	
- Considere a testabilidade	
- Comprove com código	

Fonte: AMBLER (2004)

3.8 Extreme programming (XP)

Segundo Cohen, Lindvall e Costa (2003), *Extreme Programming* (XP), ou Programação Extrema, é o método ágil de maior expressão criado nos últimos anos.

A *Extreme Programming* é um método ágil para pequenas e médias equipes desenvolverem *software*, em ambientes com requisitos instáveis. Foi criado em 1998 por Kent Beck, Ron Jeffries e Ward Cuninghan, a partir de um projeto piloto na Chandler (BECK *et al*, 1998), o XP vem ganhando cada vez mais adeptos, ampliando sua participação no mercado (HIGHSMITH, 2002a). Beck (2004) explica que o termo *Extreme* (extremo) é utilizado dado que a XP reúne um conjunto de práticas de desenvolvimento já existentes e reconhecidas como "boas práticas" no desenvolvimento de *software*, mas as leva ao extremo, ao limite. Ambler (2004) comenta que a XP "veio para ficar".

Beck (2004), um dos principais autores e fundadores da XP, a define como "um método ágil para equipes pequenas e médias desenvolvendo *software* com requisitos vagos e em constante mudança". Ainda segundo Beck, a XP "representa uma disciplina e não um método de desenvolvimento de *software*", pois XP baseia-se em regras que devem ser usadas todo tempo e não em uma sequencia de procedimentos estáticos. As regras são conhecidas como práticas e tornam possível a aplicabilidade da programação extrema no *software*.

3.8.1 Valores da XP

A XP segue um conjunto de valores, princípios e práticas adotados durante o processo de desenvolvimento por uma equipe com papéis específicos, com o objetivo de alcançar a eficiência e efetividade (BECK, 2004).

Os quatro valores da XP são:

a) Comunicação: A comunicação é o primeiro valor da XP e é considerado um ponto forte. Busca dinamismo em um ambiente onde as mudanças podem ser constantes. Sugere qualquer forma de comunicação que seja eficaz (*chat*, *e-mails*, conversas pessoalmente), ficando a critério da equipe definir a melhor forma de se comunicar.

b) Simplicidade: O *software* deve ser desenvolvido com a máxima simplicidade. Martins (2007) afirme que manter o sistema simples é uma tarefa muito difícil, onde é

importante sempre se questionar: “Qual é a solução mais simples e que funcione?”

c) *Feedback*: O *feedback* (realimentação) deve ser feito a cada iteração (de cinco até quinze minutos) ou a cada *release*, ocupando pouco tempo no desenvolvimento de requisitos falsos ou em funções que provavelmente não terão utilidade. Quanto mais rápido os problemas forem descobertos, mais rápida será a solução dos mesmos. Da mesma forma, toda a oportunidade descoberta logo será mais rapidamente aproveitada.

d) Coragem: A XP valoriza a coragem, que quando combinada com os demais valores é muito valiosa. A coragem é importante para mudar, inovar e aceitar que não se sabe tudo.

3.8.2 Práticas da XP

A XP baseia-se em práticas que devem ser aplicadas em conjunto, pois segundo Beck (2004) não fazem sentido isoladamente. Nenhuma prática consegue se manter por si só. As práticas da XP são:

a) Jogo do planejamento: no início de cada interação, clientes, gerentes e programadores se encontram para definir, estimar e priorizar os requerimentos. A ideia é que se elabore um plano aproximado no início no projeto e se faça um refinamento à medida que as necessidades e requisitos se tornem mais conhecidos;

b) Programação em pares: dois programadores utilizando o mesmo equipamento escrevem o código;

c) Pequenas versões: as versões devem ser tão pequenas quanto possível e trazerem valor para o negócio. Uma versão inicial do *software* deve ser colocada em produção após um pequeno número de iterações e, em seguida, outras versões devem ser disponibilizadas tão logo faça sentido;

d) Metáforas: clientes, gerentes e programadores criam metáforas ou conjunto de metáforas para modelagem do sistema;

e) Projeto simples: os programadores são estimulados a desenvolver o código do *software* o mais simples possível;

f) Testes: os programadores devem criar os testes de unidade antes ou mesmo durante

o desenvolvimento do código do sistema. Os clientes, por sua vez, escrevem os testes de aceitação. Ao final de cada iteração a bateria de testes deve ser conduzida;

g) Refatoração: técnica que permite a melhoria de código sem a mudança de funcionalidade, deve ser executada pela equipe do projeto, o tempo todo;

h) Integração contínua: os programadores devem integrar os novos códigos ao *software* tão rapidamente e com a maior frequência possível;

i) Propriedade coletiva do código: o código do programa deve ser propriedade de toda a equipe e qualquer integrante pode fazer alterações sempre que for necessário;

j) Cliente no local: o cliente deve trabalhar com a equipe de projeto a todo o momento, respondendo perguntas, realizando testes de aceitação e assegurando que o desenvolvimento do *software* esteja sendo feito a contento;

k) Semana de 40 horas: como trabalhar por longos períodos reduz o desempenho, o conteúdo de cada iteração deve ser planejado de forma a não haver necessidade de realização de horas extras, fazendo com que os programadores estejam renovados e ansiosos a cada manhã e cansados e satisfeitos à noite;

l) Padrão de codificação: no início do projeto deve ser criado um padrão de codificação, simples e aceito por toda a equipe, que deverá ser seguido de forma a não permitir a identificação de quem desenvolveu determinada parte do código e a auxiliar a condução do trabalho.

Assim como Beck, demais especialistas concordam que a XP não é decorrência da aplicação destas práticas isoladamente, mas sim do resultado de sua combinação (COHEN, LINDVALL E COSTA, 2003). Na tabela 8 é apresentado um resumo de características principais que norteiam a aplicação da XP:

Tabela 8: Características principais para aplicação da XP

Característica	Valores sugeridos
Tamanho da Equipe	Equipes formadas por 2 a 10 integrantes
Duração das iterações	Duração usual de 2 semanas por iteração
Equipes distribuídas	Dada que a equipe deve trabalhar preferencialmente no mesmo local físico, o XP não é indicado para equipes distribuídas
Aplicações de alta criticidade	Pode ser utilizado no desenvolvimento de software de baixa, média ou alta criticidade

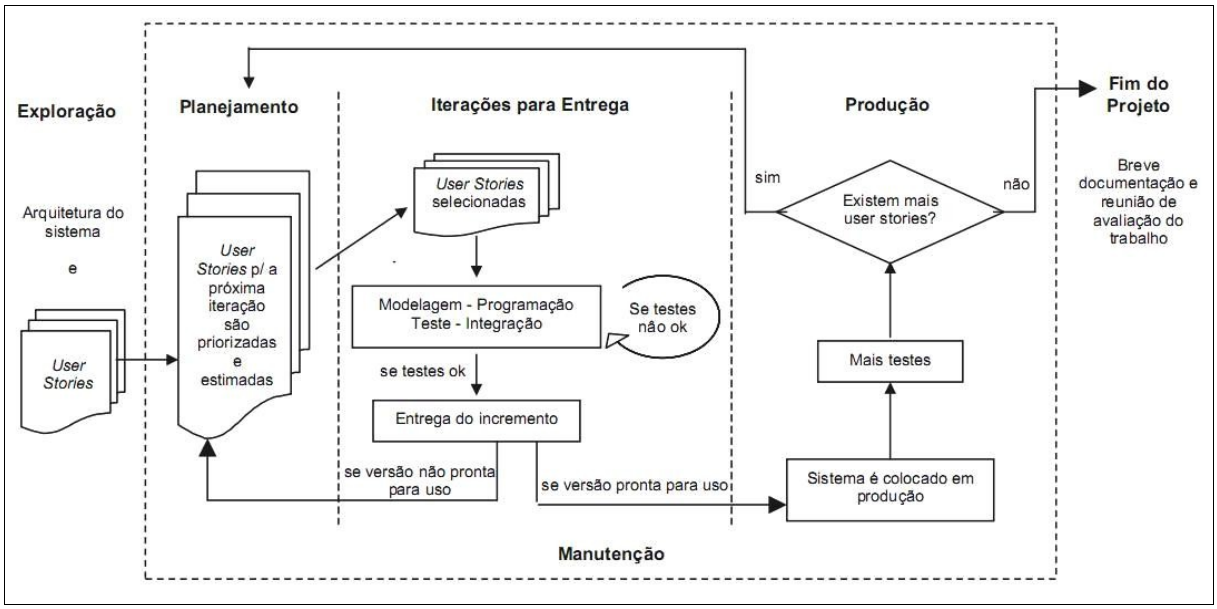
Fonte: adaptado de COHEN, LINDVALL E COSTA (2003)

3.8.3 Fases do processo XP

O projeto ideal em XP, apontado por Beck (2004), é aquele que inicia por uma curta fase de desenvolvimento, seguida de anos de produção e refinamentos simultâneos e finalmente encerra quando o projeto não faz mais sentido.

Durante todo o processo de desenvolvimento Beck (2004) sugere a realização de reuniões diárias para que a equipe fique em sincronia e se mantenha informada do trabalho e dificuldades de todos.

O ciclo de vida de um projeto XP é bastante curto e composto pelas fases de Exploração, Planejamento, Iterações para Entrega, Produção, Manutenção e Morte, conforme pode ser observado na figura 3.



Fonte: FAGUNDES (2005)

Figura 3: Ciclo de vida de um processo XP

a) Fase de Exploração

Segundo Beck (2004), o objetivo da fase de exploração é entender o real escopo do sistema e este entendimento deve ser o suficiente para que ele possa ser estimado.

Na fase de exploração as *user stories*, ou histórias do usuário, são escritas pelo cliente, para que o cliente possa descrever suas necessidades (ABRAHAMSSON *et al*, 2003). Beck (2004) comenta que o segredo para o cliente escrever histórias úteis é dar um *feedback* rápido, logo nas primeiras histórias, para que ele aprenda a especificar o que é realmente útil

para a equipe de desenvolvimento. Na figura 4 é apresentado um exemplo de cartão de *user stories*.

Data: 22/12/2000			**Dificuldade:** Média
Número da História: 005 – Criar Recibo	**Prioridade do Usuário:** Alta		**Técnico:** João Camargo
Referência Anterior: 001 – Efetuar Venda	**Risco:**		**Estimativa do Técnico:** 20 h
Descrição: Manter um recibo aberto com uma descrição breve de cada item escaneado e seu preço. Quando a venda for concluída, incluir o total na parte inferior do recibo.			
Notas:			
Acompanhamento da tarefa			
Data	**Estado**	**A realizar**	**Comentário**

Fonte: adaptado de BECK (2004)

Figura 4: Exemplo de cartão de user story

Ao mesmo tempo em que os clientes estão escrevendo as histórias, a equipe de desenvolvimento deve ir estudando e testando diferentes tecnologias para que se possa definir a arquitetura do sistema. É sugerido convidar um especialista para que não se perca muito tempo nessa fase de definição de arquitetura.

Beck (2004) aponta que essa fase de exploração da arquitetura é importante para que os programadores possam estimar as histórias na fase seguinte com maior segurança.

b) Fase de Planejamento

Esta fase tem por objetivo a definição em conjunto, cliente e desenvolvedores, da menor data e o maior conjunto de histórias que serão realizadas na primeira entrega (BECK, 2004).

Com as histórias finalizadas, o Jogo do Planejamento é iniciado em conjunto com as equipes cliente e desenvolvimento. O cliente deve definir as histórias mais importantes e que, na sua visão, devem fazer parte da primeira entrega. Dessa forma é gerada uma lista de histórias priorizadas de acordo com critérios do cliente, onde os desenvolvedores irão fazer as estimativas de tempo para o desenvolvimento. Esta estimativa é empiricamente determinada, baseada na experiência dos programadores.

c) Fase Iterações para entrega

Segundo Ambler (2004), é nesta fase que ocorre o maior trabalho de desenvolvimento,

compreendendo a modelagem, programação, escrita e execução dos testes de unidade e aceitação. Beck (2004) define que nesta fase as atividades são divididas para serem executados em iterações de duram de 1 a 4 semanas. Para cada história executada que faz parte da iteração o cliente escreve os testes de aceitação e os programadores escrevem os testes de unidade Os testes funcionais criado pelo cliente são executados no final de cada iteração (ABRAHAMSSON *et al*, 2003).

Através das reuniões diárias é possível a equipe se manter atualizada e Beck (2004) alerta que quando detectado desvios do planejamento, algo deve ser modificado. Talvez adicionar ou remover histórias ou mudar escopo. Talvez o processo de desenvolvimento precise de mudanças como melhorar as formas de trabalhar com a tecnologia, ou adaptar as ferramentas, modelos.

Ao final de cada iteração que produzir uma versão disponível para uso se iniciará a fase de produção.

d) Fase de Produção

Segundo Beck (2004), novos testes devem ser realizados para se certificar que o *software* está pronto para entrar em produção.

Esta fase se inicia ao final da primeira iteração e corresponde à liberação do sistema para uso em ambiente de produção (AMBLER, 2004). E após a liberação em produção o ritmo de evolução do sistema diminui (BECK, 2004).

e) Fase de Manutenção

Para Beck (2004), manutenção é o estado normal de um projeto XP, pois é necessário simultaneamente produzir novas funcionalidades e manter o sistema em utilização.

Para cada nova demanda (novas funcionalidades) se inicia o processo pela fase de exploração e segue as demais fases citadas. Beck (2004) enfatiza que desenvolver um sistema que já está em produção é muito diferente do que desenvolver um sistema novo. A equipe de desenvolvimento deve ser mais cuidadosa com as modificações, testes de integrações para não causar impactos no que já está funcionando em produção. O ritmo da fase de iterações diminui visto que a equipe que desenvolve pode estar alocada também em suporte aos usuários do sistema (ABRAHAMSSON *et al*, 2003). Porém, por outro lado, por já conhecerem a arquitetura e as funcionalidades do sistema, a equipe terá mais precisão em suas estimativas.

A equipe deve estar preparada para sofrer modificações no time. Beck (2004) coloca que talvez seja interessante fazer um rodízio no "*help-desk*" por exemplo, pois tem coisas que só se aprende dando suporte à produção. E quando novos membros entrarem na equipe, Beck (2004) sugere aguardar duas ou três iterações para que os novos membros tenham um tempo de aprendizado, através dos pares de programação, leitura da documentação e código existente e quando se sentirem prontos deve ir assumindo responsabilidades aos poucos, com menor carga de trabalho e ir aumentando gradualmente até que se nivelem aos demais membros.

f) Fase de Morte (Fim do projeto)

Beck (2004) considera que "Morrer bem é tão importante quanto viver bem". Um indicativo que esta fase se aproxima é quando o cliente não produz novas histórias, significa que o cliente está satisfeito com o sistema e que não vislumbra novas histórias num futuro próximo. Beck(2004) coloca que esse é o bom motivo para morrer. Mas também coloca um motivo não tão bom, que é quando o sistema não está mais agregando valor de negócio, ou então quando o cliente precisa de novas funcionalidades que não valem a pena o esforço pelo custo-benefício ou a taxa de erros sobe para um nível intolerável.

3.9 Scrum

Schwaber e Sutherland (2009) definem que Scrum é baseado nas melhores práticas aceitas pelo mercado, utilizadas e provadas por décadas. Ele é fundamentado na teoria de controle de processos empíricos, emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar riscos.

O nome Scrum é derivado de uma jogada de *rugby*[11], onde os jogadores do mesmo time se reúnem para planejar a próxima jogada. É uma forma de mostrar que o projeto deve ser conduzido em pequenos ciclos, mas com uma visão de longo prazo, que é ganhar o jogo. (MARTINS, 2007).

Scrum é um processo para construir *software* incrementalmente em ambientes complexos, onde os requisitos não são claros ou mudam com muita frequência. Segundo

11 Rugby é um esporte onde duas equipes de 15, 10 ou 7 jogadores cada uma, jogando de forma leal, portando, passando, chutando ou apoiando a bola, marquem a maior quantidade de pontos possível. A equipe que marcar mais pontos será a vencedora da partida. E, SCRUM é o nome dado a disputa da bola. Disponível em : <http://www.rugbynews.com.br>.

Martins (2007), Scrum é um processo bastante leve para gerenciar e controlar projetos de desenvolvimento de *software* e para criação de produtos. E ainda define que é um modelo de desenvolvimento ágil de *software* que fornece métodos para se definir o planejamento, os principais papéis de pessoas e a forma de trabalho do time.

A ideia do Scrum é justamente definir papéis bem específicos para as pessoas envolvidas no projeto e como cada pessoa vai jogar, ou seja, o que cada uma vai ter que fazer para o time seguir em frente no jogo: que no caso é o próprio desenvolvimento do *software*.

Scrum concentra-se mais nos aspectos gerenciais do desenvolvimento de *software*, propondo iterações de duas a quatro semanas, chamado *Sprint,* com acompanhamento diário, através das reuniões diárias (em pé). Durante a *Sprint* o produto é projetado, codificado e testado. Por dar menos ênfase ao aspectos técnicos de desenvolvimento, geralmente é combinada com práticas sugeridas por XP.

Segundo Schwaber e Sutherland (2009), o *framework* Scrum consiste em um conjunto formado por **Equipes *Scrum*** e seus papéis associados, **Eventos com Duração Fixa** (*Time-Boxes*), **Artefatos** e **Regras**.

3.9.1 Equipes Scrum e seus papéis

Schwaber e Sutherland (2009) definem que *Equipes Scrum* são projetadas para otimizar flexibilidade e produtividade. Para esse fim, elas são auto-organizáveis, interdisciplinares e trabalham em iterações (*sprints*). Cada Time *Scrum* possui três papéis: o ***ScrumMaster***, ***Product Owner*** e a **Equipe**.

- ***ScrumMaster*:** responsável por garantir que o processo seja compreendido e seguido. O *ScrumMaster*, junto com o *product owner* e *a equipe* de desenvolvimento definem as *Sprints*.
- ***Product Owner*:** responsável por maximizar o valor do trabalho que a Equipe *Scrum* faz. Define quais são as funcionalidades do produto que mais importam, criando assim o que chamamos de *backlog* do produto.
- **Equipe:** executa o trabalho propriamente dito. A Equipe consiste em desenvolvedores com todas as habilidades necessárias para transformar os requisitos do *Product Owner* em um pedaço potencialmente entregável do produto ao final da *Sprint*.

3.9.2 Eventos com duração fixa

Segundo Schwaber e Sutherland (2009) os eventos com duração fixa no *Scrum* são a ***Sprint****, a* ***Reunião de Planejamento da Release,*** *a* ***Reunião de Planejamento da Sprint,*** *a* ***Reunião Diária,*** a **Revisão da *Sprint*** e a **Retrospectiva da *Sprint.***

A ***Sprint*** é uma iteração . A *Sprint* é a principal atividade de um projeto *Scrum* e deve ser preparada de uma forma que dure de 2 a 4 semanas. Durante a *Sprint* o produto é projetado, codificado e testado. Tanto a composição da equipe quanto as metas de qualidade devem permanecer constantes durante a *Sprint*. As *Sprints* ocorrem uma após a outra, sem intervalos entre elas (MONTAIN GOAT SOFTWARE, 2010).

Antes do início das *Sprints* uma **reunião de planejamento da *release*** é realizada para o planejamento de cada entrega. As entregas ou *releases* correspondem a uma versão funcional do sistema disponibilizada em um ambiente de uso para o cliente. O propósito do planejamento da *release* é de estabelecer um plano e metas que a equipe *Scrum* e o resto da organização possam entender e comunicar. O plano da *release* estabelece a meta da *release*, as maiores prioridades do *backlog* do produto, os principais riscos e as características gerais e funcionalidades que estarão contidas na *release*. Ele estabelece também uma data de entrega e custo prováveis que devem se manter se nada mudar. A organização pode então inspecionar o progresso e fazer mudanças nesse plano da *release* a cada *Sprint*. (SCHWABER; SUTHERLAND, 2009)

A primeira atividade de cada *sprint* é uma **reunião de planejamento da s*print*.** Durante esta reunião o *product owner* e a equipe falam sobre os itens com maior prioridade do *backlog* do produto (lista de tarefas do produto) e os objetivos da *Sprint.* Os membros da equipe analisam a capacidade da equipe e definem as tarefas do *backlog* do *sprint*, que é uma lista de tarefas a realizar durante a *Sprint* (MONTAIN GOAT SOFTWARE, 2010).

Em cada dia da *Sprint,* uma **reunião diária** é realizada por todos os membros da equipe, incluindo o *ScrumMaster* e o *product owner.* Esta reunião deve ser realizada em no máximo vinte minutos. Durante esse tempo, os membros da equipe compartilham com os demais o que eles fizeram no dia anterior, o que vão fazer hoje e os obstáculos encontrados. As respostas não são um relatório para o *ScrumMaster,* mas são compromissos perante os pares. As reuniões diárias servem tanto para sincronizar o trabalho dos membros da equipe quanto para discutir o funcionamento da *Sprint* (MONTAIN GOAT SOFTWARE, 2010).

Ao final de uma *Sprint*, a equipe conduz uma **revisão da s*print,*** essa é uma reunião com duração fixa de quatro horas (SCHWABER; SUTHERLAND, 2009) . Durante essa revisão, a equipe demonstra os resultados obtidos e apresenta as funcionalidades adicionadas durante a *Sprint.* O objetivo desta reunião é obter um *feedback* do *product owner* ou de qualquer usuário ou outros convidados que tenham sido convidados para a revisão. Este *feedback* pode resultar em alterações das funcionalidades entregues recentemente. Mas o resultado pode ser apenas a revisão ou adição de itens ao *backlog* do produto (MONTAIN GOAT SOFTWARE, 2010).

Outra atividade realizada no final de cada *sprint* é a **retrospectiva da *sprint***. A equipe inteira participa dessa reunião, incluindo o *ScrumMaster* e o *product owner.* Normalmente essa reunião deve levar de quinze a trinta minutos. A reunião é uma oportunidade para refletir sobre o *sprint* que está terminando e identificar oportunidades de melhoria no novo *sprint.* A equipe discute o que gostaria de iniciar a fazer, parar de fazer ou continuar fazendo (MONTAIN GOAT SOFTWARE, 2010).

3.9.3 Artefatos

Segundo Schwaber e Sutherland (2009), os artefatos do *Scrum* incluem o ***Backlog* do Produto**, o ***Burndown* da *Release***, o ***Backlog* da *Sprint*** e o ***Burndown* da *Sprint*.**

O ***backlog* do produto** é uma lista completa dos requisitos do produto. O *backlog* do produto é priorizado pelo *product owner*, de modo que a equipe trabalha sempre com os requisitos mais prioritários em primeiro lugar. A forma mais popular e bem sucedida para criar um backlog do produto é preenchê-la com histórias de usuários, que são breves descrições de funcionalidades descritas a partir da perspectiva de um usuário ou cliente. O *backlog* do produto nunca está completo. A seleção inicial para o seu desenvolvimento somente mostra os requisitos inicialmente conhecidos e melhor entendidos. O *backlog* do produto evolui à medida que o produto e o ambiente em que ele será usado evoluem. O *backlog* do produto é dinâmico, no sentido de que ele está constantemente mudando para identificar o que o produto necessita para ser apropriado, competitivo e útil. Enquanto existir um produto, o *backlog* do produto também existe (SCHWABER; SUTHERLAND, 2009).

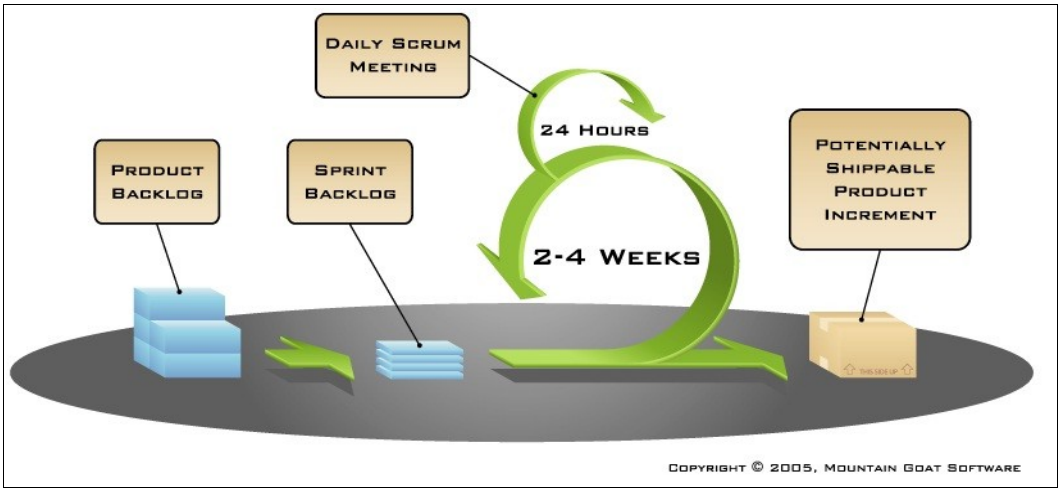
Durante os passos de desenvolvimento, as *Sprints,* a principal ferramenta de medição de desempenho é o ***Burndown Chart*,** que é uma das características mais especiais do

SCRUM e que o torna um grande diferencial, no sentido positivo. O gráfico de ***Burndown* da *Release*** registra a soma das estimativas dos esforços restantes do *backlog* do produto ao longo do tempo. O esforço estimado deve estar em qualquer unidade de medida de trabalho que a equipe *Scrum* e a organização tenham decidido usar. As unidades de tempo geralmente são *Sprints* (SCHWABER; SUTHERLAND, 2009).

O ***Backlog* da *Sprint*** consiste nas tarefas que a equipe executa para transformar itens do *backlog* do produto em um incremento "pronto" (SCHWABER; SUTHERLAND, 2009). Muitas delas são elaboradas durante a reunião de planejamento da *Sprint*. Considerando que o *backlog* do produto é uma lista de recursos a ser construído (muitas vezes escrito na forma de histórias de usuários), o backlog da *Sprint* é a lista de tarefas que a equipe precisa realizar para oferecer as funcionalidades que se comprometeram a entregar durante o *sprint* (MONTAIN GOAT SOFTWARE, 2010).

O ***Burndown* da *Sprint*** é um gráfico da quantidade restante de trabalho do *backlog* da *Sprint* em uma determinada S*print* ao longo do tempo da *Sprint.* Para criar esse gráfico, deve-se determinar quanto trabalho resta somando as estimativas do *backlog* a cada dia da *Sprint.* A quantidade de trabalho restante para uma *Sprint* é a soma do trabalho restante de todo *backlog* da *sprint.* O acompanhamento dessas somas diariamente pode ser utilizado para criar um gráfico que mostre o trabalho restante ao longo do tempo. Traçando uma linha através dos pontos no gráfico, a equipe pode gerenciar seu progresso para completar o trabalho de uma *Sprint*. A duração não é considerada em Scrum, o trabalho restante e a data são as únicas variáveis de interesse (SCHWABER; SUTHERLAND, 2009) .

A figura 5 demonstra graficamente o modelo proposto pelo SCRUM.



Fonte: http://www.mountaingoatsoftware.com/scrum/overview

Figura 5: Representação gráfica do Scrum

3.10 Considerações Finais

Este capítulo apresentou o referencial teórico sobre os métodos ágeis de desenvolvimento de *software*. O estudo proporcionou aquisição de conhecimento para tornar possível o entendimento das práticas, valores, princípios das metodologias baseadas no Manifesto Ágil. Esse estudo foi fundamental para permitir a criação da proposta de customização de processo baseada em métodos ágeis. Esta proposta de customização é apresentada no capítulo 5.

Mike Cohn é um dos membros fundadores da Aliança Ágil e elaborou um estudo cujo objetivo foi verificar se os métodos ágeis FDD, Scrum, XP, CM e DSDM atendiam alguns critérios em relação ao Manifesto Ágil. O trabalho apresentou como resultado que os métodos XP, Scrum e CM eram os métodos que mais atendiam os princípios do Manifesto Ágil, enquanto o DSDM e o FDD atendiam parcialmente (FAGUNDES, 2005).

Na tabela 9 é apresentado o resultado da comparação dos métodos ágeis segundo Mike Cohn, onde na coluna "Critérios" estão descritos os princípios do Manifesto Ágil. As colunas "FDD", "Scrum", "XP", "CM" e "DSDM" estão preenchidas com o critério de "presença" dos princípios nos métodos, onde as respostam podem ser: "Sim", "Não", "Parcialmente", "Depende", "Não enfatizado" e "Sim, mas não enfatizado".

Tabela 9: Comparação dos métodos ágeis

Critérios	FDD	Scrum	XP	CM	DSDM
Equipes auto-organizáveis e adaptáveis	Não	Sim	Sim	Sim	Parcialmente
Ausência de fases no processo de desenvolvimento	Não	Sim	Sim	Sim	Não
Planejamento mínimo	Não	Sim	Sim	Sim	Parcialmente
Escalabilidade	Sim	Sim	Sim	Sim	Sim
Refatoração	Não enfatizado	Sim	Sim	Sim	Sim
Iterativo e incremental	Sim	Sim	Sim	Sim	Sim
O progresso é medido através da entrega dos incrementos	Não	Sim	Sim	Sim	Sim
Projeto simples	Um pouco	Sim	Sim	Sim	Parcialmente
Envolvimento efetivo do cliente	Sim, mas não enfatizado	Sim	Sim	Sim	Sim
Relação adaptável com o cliente	Sim	Sim	Sim	Sim	Sim
Requisitos mutáveis	Não	Sim	Sim	Depende do tipo de projeto	Sim
Inspeções frequentes	Sim	Sim	Sim	Sim	Sim

Fonte: adaptado de FAGUNDES (2005)

Verificou-se que apesar dos métodos ágeis possuírem valores em comum, também possuem algumas diferenças significativas entre eles. O *Scrum* se destaca como o método ágil mais voltado para práticas de gerenciamento e XP como o método mais voltado para práticas voltadas para as atividades de desenvolvimento de *software.* Como a Agile Modeling (AM) é considerada uma abordagem para a modelagem e não um método de desenvolvimento de *software*, a sua utilização é indicada como complemento para os métodos ágeis (AMBLER, 2004 ; MARTINS, 2007). A AM não foi avaliada como os demais métodos, sendo assim considerada aplicável a qualquer método de desenvolvimento ágil.

No estudo realizado encontrou-se muito mais informações sobre os métodos XP e Scrum, inclusive sobre o uso conjunto de XP e Scrum. Um fator que foi bastante importante para este trabalho foi a questão de em XP, a Fase Manutenção ser considera o estado normal de um projeto XP, visto que este trabalho está focado na manutenção evolutiva de *software*.

É importante deixar claro que não foi avaliado neste trabalho a aplicação de padrões de qualidade em métodos ágeis, mas por considerar importante este aspecto foi realizada uma breve pesquisa sobre a compatibilidade dos métodos ágeis com o modelo CMMI.

Nos estudos de Kane e Ornburn (2002), os métodos ágeis são compatíveis com o modelo CMMI. Porém, são necessárias inclusões de algumas práticas nos métodos ágeis para total compatibilização, como, por exemplo, estes métodos satisfazem parcialmente as áreas de processo como Gerenciamento e Desenvolvimento de Requisitos do modelo CMMI. Neste estudo, o autor não apresenta que práticas poderiam ser incluídas, para total compatibilização com o modelo CMMI.

Zanatta (2004) apresenta em seu trabalho uma customização de Scrum para atender algumas áreas de processo do modelo CMMI, e conclui que é possível utilizar métodos ágeis com o modelo CMMI, desde que a organização esteja disponível para aplicar novas perspectivas.

4 . MEDIÇÃO E ANÁLISE DO PROCESSO ATUAL

Este capítulo tem por objetivo apresentar o estudo do processo atual, o qual está sendo objeto de estudo deste trabalho, assim como as demais informações relevantes para contextualizar as características da equipe e organização. O estudo do processo atual faz parte da metodologia de customização de processo adotada neste trabalho e sugerida por Sommerville (2007). A metodologia de customização adotada na execução deste trabalho foi apresentada no capítulo 2, onde Sommerville (2007) define o aprimoramento de processo como sendo uma atividade cíclica composta pelos estágios de medição, análise e mudança de processo. Neste capítulo serão apresentados os estágios medição e análise do processo.

A equipe que está sendo objeto de estudo deste trabalho é composta por um analista de sistemas e programadores e é responsável pela manutenção de um sistema específico e desenvolvido pela própria organização. A organização é uma instituição de ensino superior, comunitária e regional, com atuação na região nordeste do Estado do Rio Grande do Sul.

O sistema o qual essa equipe realiza manutenção é um sistema *web* desenvolvido sob o paradigma de orientação a objetos. É um portal de educação, também conhecido por ambiente virtual de aprendizagem. Por se tratar de um sistema utilizado pela comunidade acadêmica de uma universidade, os usuários são compostos por alunos e professores. É um número bastante expressivo de usuários, e por esse motivo, existe uma "comissão interdisciplinar"[12] que representa os usuários e exerce o papel de cliente no processo de manutenção do *software*. Esta comissão possui um representante, o qual tem autonomia para definir requisitos quando da não disponibilidade dos demais membros da comissão.

O capítulo está organizado em quatro seções, onde na seção 4.1 são apresentadas as definições dos objetivos da customização conforme abordagem GQM (*Goal-Question-Metric*), sugerida por Sommerville (2007) no estágio de medição do processo.

Na seção 4.2 é apresentada a análise do processo atual assim como as técnicas aplicadas e seus resultados obtidos, que fazem parte do estágio de análise do processo.

Já na seção 4.3 é apresentado o modelo do processo atual que foi gerado a partir das informações obtidas através da análise do processo, que é o que deve ser gerado como resultado do estágio da análise do processo.

E por fim, na seção 4.4 são apresentadas as considerações finais do capítulo.

12 Esta comissão é composta por diversos professores da instituição de ensino representando os interesses dos usuários e as várias áreas de conhecimento.

4.1 Medição do processo

Conforme sugerido por Sommerville (2007), o aprimoramento de processo deve ser guiado por objetivos e esses objetivos devem ser definidos através da abordagem Objetivo-Questão-Métrica, ou GQM (*Goal-Question-Metric*), no estágio de medição do processo.

Para o estudo de caso que foi aplicado neste trabalho foram definidos dois objetivos: **aumentar a eficiência da equipe de desenvolvimento** e **aumentar a satisfação do cliente**. Ambos objetivos foram definidos mantendo o alinhamento com o negócio, já que se trata de um sistema que é utilizado pelos clientes da instituição (alunos) e que tem impacto na qualidade de ensino oferecida pela mesma. O sistema é um importante canal de comunicação do aluno com o professor e com os demais colegas.

Seguindo a abordagem GQM, para cada objetivo foram definidas questões e para cada questão métricas. Abaixo são apresentados os objetivos, questões e métricas da customização do processo que este trabalho propõe:

4.1.1 Objetivo 1 - Aumentar a eficiência da equipe de desenvolvimento

O objetivo 1 está relacionado com a produtividade da equipe, visto que fazer as coisas de forma correta evitam retrabalho e impactam no número de nas novas funcionalidades (requisitos) liberadas. Para o objetivo 1 foram definidas as seguintes questões e métricas:

a) Questão Q1.1: Como aumentar o grau de precisão das estimativas de prazo acordadas com o cliente?

- ***Métrica M1.1:*** Percentual da razão entre o número de funcionalidades entregues no prazo sobre o número de funcionalidades estimadas.

b) Questão Q1.2: Como reduzir o índice de erros de programação detectados na fase de testes, antes da entrega?

- ***Métrica M1.2****:* Percentual da razão entre o número de funcionalidades com erros detectados na fase de testes sobre o número de funcionalidades entregues.

c) Questão Q1.3: Como aumentar o envolvimento do cliente no processo de desenvolvimento?

- ***Métrica M1.3:*** Número de contatos formais entre o cliente e a equipe de desenvolvimento durante um ciclo de desenvolvimento.

4.1.2 Objetivo 2 - Aumentar a satisfação do cliente

O objetivo 2 está relacionado com o reflexo do processo de desenvolvimento na qualidade do produto (*software*) sob o ponto de vista do cliente (usuários do sistema). Para o objetivo 2 foram definidas as seguintes questões e métricas:

a) Questão Q2.1: Como aumentar o nível de adequação[13] das funcionalidades do cliente?

- ***Métrica M2.1****:* Percentual da razão entre a quantidade de funcionalidades solicitados depois da entrega sobre o número de funcionalidades entregues.

b) Questão Q2.2: Como reduzir o número de não conformidades detectadas pelo cliente após a entrega?

- ***Métrica M2.2****:* Percentual da razão entre o número de funcionalidades não conformes sobre o número de funcionalidades entregues.

c) Questão Q2.3: Como reduzir a quantidade de erros (*bugs*) detectados pelo cliente após a entrega?

- ***Métrica M2.3:*** Percentual da razão do número de funcionalidades com erros detectados pelo cliente sobre o número de funcionalidades entregues.

Na tabela 10 é demonstrado um resumo das questões e métricas em relação aos objetivos. Na coluna "Objetivos" são informados os objetivos da customização, nas colunas "Questão" e "Métrica" são informadas, respectivamente, as questões e métricas definidas para o objetivo na linha correspondente.

Tabela 10: Resumo das questões e métricas por objetivo

Objetivos	Questão	Métrica
1) Aumentar a eficiência da equipe	Q 1.1	M 1.1
	Q 1.2	M 1.2
	Q 1.3	M 1.3
2) Aumentar a satisfação do cliente	Q 2.1	M 2.1
	Q 2.2	M 2.2
	Q 2.3	M 2.3

13 Adequação é a capacidade do produto de *software* de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados (NBR ISO/IEC 9126-1, 2003).

Os objetivos definidos nesta fase de medição de processo orientaram as mudanças de processo que foram propostas. E as métricas serviram de base para a avaliação do modelo proposto neste trabalho. Conforme sugerido por Sommerville (2007), o ciclo de aprimoramento é uma atividade cíclica onde depois de aplicada a mudança de processo, deverá reiniciar pela medição do processo. Após a implantação das mudanças as métricas serão colhidas com o intuito de responder as questões e por fim, verificar se os objetivos da customização do processo foram alcançados.

4.2 Análise do processo

A análise do processo atual foi realizada utilizando algumas técnicas sugeridas por Sommerville (2007) que são os questionários, entrevistas, reuniões e observação.

De acordo com Lakatos e Marconi (2005), o questionário é um instrumento de coleta de dados, constituído por uma série ordenada de perguntas, que devem ser respondidas sem a presença do entrevistador. Neste trabalho o questionário foi utilizado em caráter exploratório e qualitativo. O uso de questionário neste trabalho é em caráter exploratório e qualitativo, com o objetivo de levantar informações sobre o perfil da equipe e seu processo atual.

As entrevistas são mais informais do que os questionários, pois possibilitam a discussão de assuntos de maneira mais ampla, dessa forma que os entrevistados falem mais abertamente sobre os problemas e dificuldades do processo no dia a dia (SOMMERVILLE, 2007). De acordo com Lakatos e Marconi (2005) a entrevista tem como principal objetivo a obtenção de informações do entrevistado, sobre determinado assunto ou problema. São classificadas em estruturadas e não estruturadas.

As entrevistas estruturadas são elaboradas mediante questionário totalmente estruturado, ou seja, é aquela onde as perguntas são previamente formuladas e tem-se o cuidado de não fugir a elas (LAKATOS; MARCONI, 2005). Assim como o questionário, as entrevistas também tem objetivos exploratórios e qualitativos. Já as não estruturadas, o entrevistador tem a liberdade para desenvolver cada situação em qualquer direção conforme considere adequada.

Já a observação e análise etnográfica permitem um estudo mais profundo dos detalhes do processo, o que pode levar mais tempo e o ideal é que ocorra o acompanhamento de um ciclo por inteiro do processo em questão (SOMMERVILLE, 2007).

4.2.1 Questionário

O questionário foi enviado em meio eletrônico (*e-mail)* aos membros da equipe de desenvolvimento que foi estudada. O questionário foi aplicado com os seguintes objetivos:

- **Objetivo 1:** Identificar o perfil e conhecimento da equipe
- **Objetivo 2:** Capturar informações sobre o processo atual e a percepção da equipe em relação ao processo atual
- **Objetivo 3:** Capturar sugestões de melhorias do processo por parte da equipe
- **Objetivo 4:** Identificar a existência de interesse por adoção de práticas ágeis de desenvolvimento

O questionário aplicado na equipe segue como anexo A deste trabalho e foi dividido em seis seções, de forma que cada objetivo possui uma seção que compreende um conjunto de perguntas.

Segundo Lakatos e Marconi (2005), as perguntas quanto à sua forma, são classificadas em três categorias: abertas, fechadas e múltipla escolha. As questões abertas permitem ao informante responder livremente.

As questões fechadas são aquelas onde o informante escolhe a sua resposta de acordo com as opções disponíveis. As questões fechadas denominam-se dicotômicas quando possuem somente duas alternativas, e tricotômicas quando possuem três alternativas.

E as questões de múltipla escolha são questões fechadas mas que apresentam um série de possíveis respostas. As questões de múltipla escolha podem ser perguntas com mostruário e perguntas de estimação ou avaliação. As perguntas com mostruário são aquelas onde as possíveis respostas estão estruturadas junto à pergunta, devendo o informante assinalar uma ou várias. Possui a desvantagem de sugerir as respostas. E as perguntas de estimação ou avaliação consiste em emitir um julgamento através de de uma escala com vários graus de intensidade para um mesmo item. As respostas sugeridas são quantitativas e indicam um grau de intensidade crescente ou decrescente.

A seguir estão descritas as questões elaboradas para atender cada os objetivo do questionário. As perguntas serão classificadas de acordo com as categorias definidas por Lakatos e Marconi (2005).

- **Objetivo 1: Identificar o perfil e conhecimento da equipe.** Para atender o objetivo 1 foram elaboradas as seguintes questões:

  Questão 1) *Qual é o seu papel na equipe de desenvolvimento?*

É uma pergunta classificada como múltipla escolha com mostruário, onde o informante assinala uma ou várias delas. As alternativas foram compostas segundo a Classificação Brasileira de Ocupações - CBO (MINISTÉRIO DO TRABALHO E EMPREGO, 2002).

Questão 2) *A quanto tempo você trabalha na empresa?*

Questão 3) *A quanto tempo faz parte desta equipe de desenvolvimento que* está sendo *estudada?*

As questões 2 e 3 são abertas, que permitem ao informante responder livremente.

Questão 4) *Você participou do desenvolvimento do software que hoje realiza manutenção evolutiva?*

Questão 5) *Você possui dificuldades em executar a manutenção evolutiva por falta de conhecimento nas funcionalidades já existentes?*

As questões 4 e 5 são questões fechadas onde o informante escolhe a sua resposta de acordo com as opções disponíveis. Por possuir três alternativas é chamada de tricotômica. As alternativas são “Sim”, “Não” e “Parcialmente”.

Questão 6) *Você possui conhecimento nos métodos ágeis de desenvolvimento de software?*

A questão 6 é uma combinação de questão fechada tricotômica com questão aberta. Possui as alternativas são “Sim”, “Não” e “Parcialmente” e a questão aberta é “Se Sim ou Parcialmente, quais?”

- **Objetivo 2: Capturar informações sobre o processo atual e a percepção da equipe em relação ao processo atual.** Para atender o objetivo 2 foram elaboradas as seguintes questões.

  Questão 7) *Considere que uma metodologia de desenvolvimento de software engloba o processo, os métodos e as ferramentas. Você tem clareza dos métodos, técnicas, processo e ferramentas que compõem a 'metodologia' utilizada hoje pela sua equipe?*

A questão 7, assim como a questão 6, é uma combinação de questão fechada tricotômica com questão aberta. Possui as alternativas são “Sim”, “Não” e “Parcialmente” e a

questão aberta é “Comentários”. A formulação da pergunta está baseada na definição de metodologia proposta por Pressman (2006).

Questão 8) *Definindo processo como o conjunto das atividades executadas, sua sequência e seus executores, você considera que sua equipe possui um processo de desenvolvimento de software bem definido (padronizado)?*

A questão 8 também é uma combinação de questão fechada tricotômica com questão aberta. Possui as alternativas são “Sim”, “Não” e “Parcialmente” e a questão aberta é “Comentários”. A formulação da pergunta está baseada na definição de processo proposta por OMG (2010).

Questão 9) *Descreva brevemente, em termos simples, a sequência de atividades que são executadas no processo atual utilizado pela equipe a partir da solicitação do usuário até a disponibilização em produção. (Lembre-se que estamos considerando somente manutenção evolutiva). Para demonstrar o sequenciamento pode enumerar, exemplo: 1, 2, 3...*

A questão 9 é uma questão aberta.

Questão 10) *Considerando características de seu processo atual, responda:*

> *10.1) Gera muita documentação?*
> *10.2) A equipe de desenvolvimento trabalha no mesmo ambiente (sala) de trabalho?*
> *10.3) A equipe tem autonomia e poder de decisão sobre suas atividades?*
> *10.4) A equipe tem bastante contato diário ou semanal com o cliente (solicitante da manutenção evolutiva)?*
> *10.5) Quando ocorrem mudanças que afetam o processo de desenvolvimento, a equipe de adapta facilmente?*
> *10.6) A equipe possui alguma forma de comunicação diária?*

As questões que compõem a questão 10 são fechadas e tricotômicas. As possíveis alternativas são “Sim”, “Não” e “Parcialmente”. A formulação das questões 10.1, 10.2, 10.3, 10.4, 10.5 e 10.6 estão baseadas nos valores do Manifesto Ágil.

- **Objetivo 3: Capturar sugestões de melhorias do processo por parte da equipe.**

Para atender o objetivo 3 foram elaboradas as seguintes questões.

Questão 11) *Considerando uma oportunidade de melhoria no seu processo atual de desenvolvimento, enumere os itens abaixo dando sua prioridade de melhorias, ou seja, enumere 1 para o item que você considera mais importante/prioritário, e 17 para o menos*

importante.

A questão 11 é de múltipla escolha com mostruário. As alternativas estão baseadas nas práticas ágeis de desenvolvimento.

Questão 12) *Escreva em forma de itens suas sugestões de melhorias para o seu processo atual. Considere os métodos, ferramentas, técnicas, enfim, tudo aquilo que entenda que possa proporcionar melhorias no processo e consequentemente nas atividades desempenhadas em seu dia a dia.*

A questão 12 é uma questão aberta.

- **Objetivo 4: Identificar a existência de interesse por adoção de práticas ágeis de desenvolvimento.** Para atender o objetivo 4 foram elaboradas as seguintes questões.

Questão 13) *Você tem interesse em aplicar métodos ágeis em seu processo?*

A questão 13 é uma combinação de questão fechada tricotômica com questão aberta. Possui as alternativas são "Sim", "Não" e "Parcialmente" e a questão aberta é "Se Sim ou Parcialmente, quais?"

Questão 14) *Cite algumas práticas (atividades, técnicas) dos métodos ágeis que gostaria de aplicar no seu processo de desenvolvimento. Pode citar em forma de itens enumerando pela prioridade (utilizando mesmo critério da questão 11).*

A questão 14 é uma questão aberta.

Após o retorno dos questionários respondidos, os resultados foram tabulados e analisados. Pode ser verificado que a equipe é formada por um analista de sistemas e dois programadores, que trabalham na mesma equipe de desenvolvimento há 2,5 anos. Todos participaram do desenvolvimento do sistema o qual realizam manutenção evolutiva hoje e demonstraram interesse em práticas ágeis de desenvolvimento, em especial Scrum e XP.

As respostas estão consolidadas na tabela 11, onde na coluna "Objetivo" estão identificados os objetivos, na coluna "Questão" estão informados os números das questões, e na coluna "Resultado apurado" uma breve descrição do resultado apurado com a aplicação do questionário.

Nas linhas são apresentadas as respostas por questão, onde essas estão agrupadas por objetivo. Já nas colunas "Objetivo" e "Questão" estão descritos os números dos objetivos e

questões, respectivamente. Na coluna "Resultado apurado por perfil" são apresentadas as respostas por perfil do respondente.

Tabela 11: Resultado da aplicação do questionário por perfil

Objetivo	Questão	Resultado apurado por perfil	
		Programador	Analista
1	1	2 responderam programadores	1 respondeu analista de sistemas
	2	1 há 10 anos e outro há 4 anos	Mais de 10 anos
	3	2,5 anos	2,5 anos
	4	Sim	Sim
	5	Sim	Sim
	6	Sim e XP	Sim, XP e Scrum
2	7	1 Sim e outro parcialmente	Sim
	8	1 Sim e outro Não	Não
	9	1) Recebimento da solicitação de manutenção. 2) Se aprovada pela Comissão são geradas as atividades. 3) Analista atualiza modelo de dados (ER), quando necessário. 4) Desenvolvedor codifica e testa. 5) Analista testa. 6) É agendado uma data para liberação em ambiente de produção.	1) Recebida a solicitação. 2) É realizada uma análise de impacto para medir as consequencias. 3) Se a comissão aprovar a modificação é registrado no Redmine um ticket e modificado o ER se for necessário. 4) O desenvolvedor codifica em ambiente de desenvolvimento. 5) São realizados testes pelo desenvolvedor. 6) São realizados testes pelo analista. 7) Se tudo correto, é disponibilizado em ambiente de produção.
	10	Mais direcionado para as características do métodos ágeis	Mais direcionado para as características do métodos ágeis
3	11	Prioridade para: comunicação, especificação de requisitos, documentação do projeto, avaliação do usuário, testes e agilidade.	Maior prioridade para: testes, comunicação, documentação da transição, auto-avaliação da equipe técnica, avaliação do usuário, agilidade e especificação de requisitos.
	12	Melhorar documentação de projeto, documentação final e mais contato com cliente.	Melhorar documentação das funcionalidades e divulgá-las.
4	13	Sim	Sim
	14	Reuniões diárias com equipe técnica, reuniões com o cliente para definição de requisitos, entregas frequentes, documentação da modelagem segundo sugestões da AM.	Reuniões diárias da equipe técnica

4.2.2 Entrevistas

Foram aplicadas as entrevistas estruturadas e não estruturadas. As entrevistas estruturadas foram realizadas em grupo, em formato de reuniões onde o entrevistador conduziu de acordo com o roteiro preparado para a entrevista.

Os principais objetivos das entrevistas foram:

- **Objetivo 1:** Capturar detalhes do processo atual que não foram identificados no questionários como por exemplo as ferramentas utilizadas, e os fluxos de exceção.

- **Objetivo 2:** Identificar as ferramentas utilizadas e os artefatos gerados e durante o processo atual.

- **Objetivo 3:** Verificar a forma de controle e medição utilizada pela equipe no processo atual.

- **Objetivo 4:** Identificar a percepção da equipe em relação aos pontos fortes e pontos fracos do processo atual.

Abaixo são apresentados os tópicos que foram abordados no roteiro da entrevista e uma breve descrição Cada tópico tem por objetivo atender um dos objetivos. O roteiro na íntegra segue como anexo B deste trabalho.

- **Processo atual:** obter informações das formas de entrada de demandas de manutenção evolutiva, a organização da equipe durante o desenvolvimento e a entrega em ambiente de produção. Pensando em processo no conceito geral, foram identificadas as entradas, processamentos e saídas.

- **Ferramentas e artefatos:** identificar as ferramentas e artefatos utilizados no processo de desenvolvimento.

- **Percepção da equipe:** identificar os pontos fortes, fracos , problemas enfrentados e desejos de melhoria.

- **Indicadores e medição:** Identificar a forma de controle e medição utilizado pela equipe. Se são utilizados indicadores, metas e a forma como essas informações são atualizadas e utilizadas.

Os resultados obtidos através das entrevistas são apresentados de forma resumida na tabela 12, onde na coluna “Objetivo” são informados o objetivo em identificação numérica e na coluna “Resultado apurado” uma breve descrição das principais informações obtidas.

Tabela 12: Resultados apurados nas entrevistas

Objetivo	Resultado apurado
1	Capturado detalhes do processo atual sobre as formas de contato com o cliente, as formas que as demandas chegam, as características das manutenções executadas
2	Identificadas o uso das ferramentas Redmine, Jdeveloper, Django e Python. Como artefatos gerados são a solicitação de manutenção, a lista de atividades, as informações da atividades, o código fonte, o diagrama ER.
3	Fortes: Conhecimento da equipe no sistema, comunicação entre a equipe técnica, entrosamento da equipe técnica. Fracos: qualidade da documentação gerada, envolvimento do cliente, comunicação com o cliente.
4	Não existem indicadores e nem medição no processo atual. A única forma de controle é pelo prazo final estipulado para o cliente (Comissão).

4.2.3 Observação

O objetivo da técnica de observação foi capturar detalhes do processo atual e das características da equipe que não foram identificados no questionário e entrevistas. A aplicação da observação foi realizada em uma demanda de manutenção evolutiva de baixa complexidade e consequentemente de rápida execução. Esta demanda chegou através de *e-mail* de um professor sugerindo uma mudança em uma funcionalidade existente. O analista de sistemas avaliou a solicitação e por considerar de pouco impacto não foi consultado o parecer da comissão.

Esse acompanhamento aconteceu durante três semanas onde foram realizadas conversas diárias com o analista de sistemas e os programadores. Esse período de tempo ultrapassou o ciclo da manutenção específica que foi acompanhada, mas foi prolongado o período da observação para pode capturar detalhes das características da equipe e da forma de trabalho.

No mesmo ambiente físico, junto a equipe foi realizado o acompanhando desde a entrada da demanda até a entrega em produção. Durante o acompanhamento foi acompanhado o registro de informações através da ferramenta utilizada pela equipe, Redmine, acompanhado com o desenvolvedor a alteração do código fonte em alguns momentos.

Através da aplicação da observação foi possível entender melhor a forma de trabalho da equipe e como as coisas acontecem no dia-a-dia.

Abaixo segue considerações sobre alguns aspectos que foram observados durante a aplicação da observação que são relevantes:

- **Comunicação entre equipe técnica:** observado que a comunicação é um ponto forte da equipe. Fisicamente próximos (um ao lado do outro) a qualquer momento dúvidas e dificuldades são resolvidas na hora. Na figura 6 é apresentado o leiaute das mesas de trabalho da equipe.

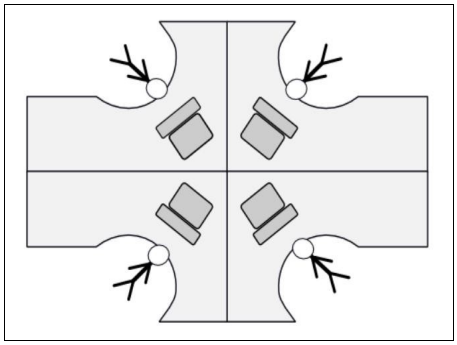


Figura 6: Leiaute do ambiente de trabalho da equipe

- **Comunicação com cliente:** Quando precisam de definições por parte do usuário, como o cliente é representado por uma comissão e esta comissão possui um representante, é tentado contato telefônico ou reunião, mas foi observado que existe problema de disponibilidade de agenda desses participantes da comissão. No caso da manutenção específica, não foi realizado nenhum contato com o cliente durante o processo, somente após a liberação em produção.

- **Documentação gerada:** A equipe utiliza a ferramenta Redmine para registrar as atividades e todas as informações do processo de desenvolvimento. As atividades são criadas pelo analista de sistemas e atribuídas ao(s) desenvolvedor(es). Nesta documentação as informações referentes as tarefas executadas. Identificado que informações como “Data prevista”, “Percentual terminado”, “Tempo gasto” não são informadas.

  Na figura 7 é apresentado um “recorte” da documentação do acompanhamento realizado, onde o nome dos membros da equipe foram preservados e substituídos pelos seus papéis na equipe (analista de sistemas e desenvolvedor). A documentação na íntegra segue como anexo C deste trabalho.

Figura 7: Acompanhamento da documentação registrada pela equipe

- **Ferramentas utilizadas:** As ferramentas utilizadas pela equipe são: Jdeveloper para modelagem do diagrama Entidade-Relacionamento (ER), Python como linguagem de programação, e Django como *framework* de desenvolvimento.

- **Testes:** Os programadores realizam os testes individuais de seus programas, esses testes são manuais, não são escritos antes da codificação. Após os testes dos programadores, o analista de sistema realiza novos testes de negócio e integração. Estes testes são sempre realizados em uma base de desenvolvimento, onde demais equipes estão realizando suas manutenções.

- **Liberação em ambiente de produção:** as alterações são disponibilizadas em base de produção e alguns usuários são comunicados sobre as novas funcionalidades, afim de testar. Se esses usuários identificarem alguma inconsistência as alterações necessárias são executadas.

4.2.4 Modelo do processo atual

Com os resultados obtidos através da aplicação do questionário, entrevistas e

observação foram levantadas as informações do processo atual utilizado pela equipe estudada, o que proporcionou a criação do modelo de processo, conforme sugerido por Sommerville (2007).

A equipe é pequena, composta por um analista de sistemas e dois programadores, e é responsável pela manutenção evolutiva de um sistema *web* com orientação a objetos, onde são desenvolvidas novas funcionalidades do *software* atual assim como alterações de requisitos já existentes, onde a arquitetura atual do sistema é mantida e são criados novos componentes quando necessários. As demandas de manutenção variam de baixa a alta complexidade e

A modelagem do processo foi realizada através da metodologia *Business Process Modeling* (BPM) e o levantamento das informações ocorreu através de entrevistas estruturadas com a equipe específica. O processo foi desenhado utilizando a notação *Business Process Modeling Notation* (BPMN), conforme OMG (2010) .

Na figura 8 é ilustrado o processo atual em BPMN, que se inicia pela atividade de 'Registrar solicitação', seguida pela atividade de 'Analisar Impacto', após a análise de impacto a solicitação é aprovada ou rejeitada. Se aprovada, segue para as atividades 'Registrar atividades', 'Criar modelo ER', 'Codificar', 'Realizar teste unitário', 'Realizar teste de sistema' e por fim 'Liberar em ambiente de produção'. Em paralelo aos testes são atualizadas as informações na ferramenta Redmine.

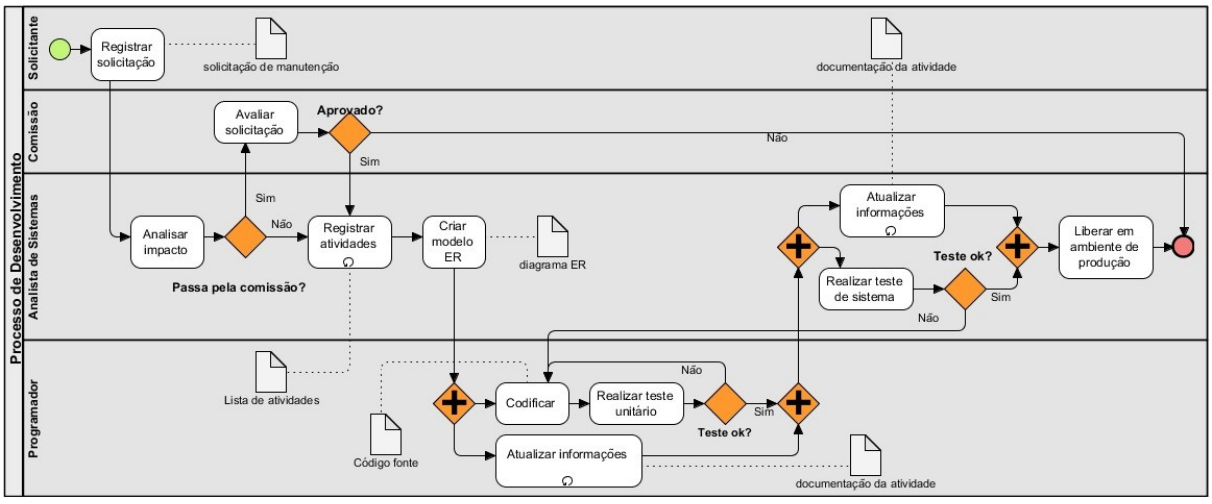


Figura 8: Modelo do processo atual

Avaliando as atividades do processo atual como um todo podemos dividir o processo em três fases: Análise, Desenvolvimento e Produção.

Na Fase de Análise acontecem as atividades “Registrar solicitação”, “Analisar Impacto”, “Registrar atividades” e “Criar modelo ER”. Essa última atividade é opcional, pois nem todas as manutenções necessitam de modificações no ER. Tem como marco a aprovação ou não da solicitação, e os trabalhadores envolvidos são o Solicitante (usuário), Analista de Sistemas e Programadores. Os artefatos gerados nessa fase são o registro da solicitação de manutenção e a lista de atividades registradas na ferramenta Redmine.

Na fase de desenvolvimento acontecem as atividades de “Codificar”, “Realizar teste unitário”, “Realizar teste de sistema”. Em para lelo a equipe vai atualizando as informações das atividades realizadas na ferramenta Redmine, conforme pode ser observado na figura 7 (pág.47). Nesta fase o marco é o(s) programas(s) testado(s) e funcionais prontos para liberação em produção. Os trabalhadores envolvidos nesta fase são o analista de sistemas e os desenvolvedores. E geram como artefato o código fonte e a documentação das atividades atualizada no Redmine.

E por fim, na fase de Produção, os programas são liberados em ambiente de produção e somente alguns usuários são comunicados para testar. Estes usuários utilizam (testam) as novas funcionalidades e dão *feedback* para o analista de sistema. Quando o usuário está satisfeito e aprova, a solicitação de manutenção é dada como encerrada. O marco desta fase é a aprovação dos usuários chaves que testam as funcionalidades. Os trabalhadores envolvidos são o Analista de sistemas, desenvolvedores e usuário. Nesta fase o artefato gerado é a atualização das informações no Redmine.

4.3 Considerações finais

Neste capítulo foram apresentados os resultados dos estágios de medição e análise do processo atual, conforme metodologia de customização utilizada neste trabalho e sugerida por Sommerville (2007).

No estágio de medição do processo foram definidos os objetivos da customização e para cada objetivo foram definidas as questões e métricas. Essas métricas foram coletadas na aplicação do estudo de caso, cujo resultado é apresentado no capítulo 7.

No estágio de análise do processo atual foram aplicadas algumas técnicas sugeridas por Sommerville (2007) como questionário, entrevistas e observação. Através dos resultados obtidos com a aplicação dessas técnicas foi possível gerar o modelo do processo atual.

Seguindo a metodologia de customização sugerida por Sommerville (2007) que foi adotada neste trabalho, as informações obtidas na análise do processo atual serviram de base para identificar os aprimoramentos a serem introduzidos no processo que estão descritos no próximo capítulo, que apresenta o estágio de mudança de processo.

5 . APRIMORAMENTO DO PROCESSO

O ciclo de aprimoramento de processo sugerido por Sommerville (2007) que foi adotado neste trabalho é composto pelos estágios de medição, análise e mudança, onde os dois primeiros já foram apresentados no capítulo anterior. O estágio de mudança de processo é composto por cinco fases: identificar aprimoramentos, priorizar aprimoramentos, introduzir mudança de processo, treinar a mudança de processo, e ajustar as mudanças de processo. O referencial teórico de cada fase foi apresentados no capítulo 2, seção 2.2 (página 26).

Neste capítulo é apresentada a fase "identificar aprimoramentos". A fase "priorizar aprimoramentos" não foi executada neste trabalho já que a customização de processo sugerida foi planejada e executada em uma única vez. Essa fase se faz necessária quando as mudanças são muitas e devem ocorrer de forma incremental, dessa forma priorizando as melhorias e implantando-as conforme a priorização. As demais fases são apresentadas no capítulo 7 que relata o estudo de caso que foi aplicado.

O capítulo é composto por 3 seções, onde na seção 5.1 são apresentados os aprimoramentos identificados os quais guiaram a proposta do novo modelo de processo que foi proposto.

Na seção 5.2 é apresentada a proposta de customização do processo de desenvolvimento de *software*, introduzindo algumas práticas dos métodos ágeis de desenvolvimento estudados e apresentados no capítulo 3. Essa proposta de customização tem como escopo o processo de manutenção evolutiva de *software*, que foi descrito nos objetivos deste trabalho, na seção 1.3. Dessa forma deve ser considerado que o sistema já está em uso, o que minimiza muito as atividades relacionadas a definição de arquitetura e componentes do *software*.

Como o objetivo do trabalho é propor uma metodologia de desenvolvimento de *software*, é importante lembrar a definição do termo "metodologia" que está sendo utilizada neste trabalho. A definição utilizada é a de Pressman (2006), onde define que a metodologia de desenvolvimento de *software* engloba o processo, os métodos e as ferramentas. O processo é definido como um *framework* para as tarefas que são necessárias para a construção de *software* de alta qualidade. Os métodos fornecem a "técnica" de como fazer para construir *softwares*. E as ferramentas são mecanismos automatizados de apoio ao processo e aos métodos.

E por fim, na seção 5.3 são apresentadas as considerações finais do capítulo.

5.1 Identificar e priorizar aprimoramentos do processo

A primeira fase do estágio de mudança é identificar os aprimoramentos, que consiste em utilizar os dados da análise do processo para definir os aprimoramentos de processo necessários para alcançar os objetivos da customização.

Os aprimoramentos identificados foram:

- **Padronizar o processo:** padronizando as atividades, artefatos, ferramentas, papéis e responsabilidades permitirá que a equipe tenha diretrizes do que fazer e do como fazer. Com isso existirá um modelo formal do processo padronizado, e caso entre novos membros na equipe isso pode auxiliar essa pessoa na ambientação. Com o processo padronizado a velocidade da equipe tende a aumentar, visto que cada membro terá maior clareza de seu papel e suas responsabilidades.
- **Melhorar a comunicação com o cliente:** esta melhoria pode ajudar na busca por um maior envolvimento do cliente. A ideia é ter o cliente acompanhando e fazendo parte do processo para que seja mais comprometido.
- **Melhorar a definição dos requisitos com a participação do cliente:** este aprimoramento vai proporcionar comprometimento do cliente, envolvimento na definição dos requisitos, dessa forma diminuindo o risco de não conformidades.
- **Melhorar a documentação gerada durante o processo:** esse aprimoramento irá auxiliar a equipe na rastreabilidade das modificações, assim como pode proporcionar um "*help*" melhor para o usuário final.

Esses aprimoramentos foram definidos baseados nos objetivos da customização e em conjunto com a equipe que foi objeto de estudo deste trabalho. Segundo Sommerville (2007), como resultado da fase de identificação de aprimoramentos deve ser gerado um novo modelo de processo, o qual será apresentado na próxima seção.

5.2 O novo modelo de processo de desenvolvimento

Partindo dos aprimoramentos identificados e apresentados na seção anterior, foi

realizado um estudo de quais práticas ágeis se adequariam e poderiam ser aplicadas na equipe estudada, para dessa forma chegar à um modelo de processo customizado baseado nas práticas ágeis que permitam alcançar os objetivos da customização.

A proposta do novo modelo de processo está baseado na premissa de um processo iterativo e incremental com práticas ágeis de desenvolvimento de *software*. O novo modelo de processo é composto por três fases: **Planejamento**, **Desenvolvimento** e **Encerramento** que possibilitam a iteratividade. Na figura 9 são apresentadas as fases do processo proposto e os artefatos gerados em cada fase.

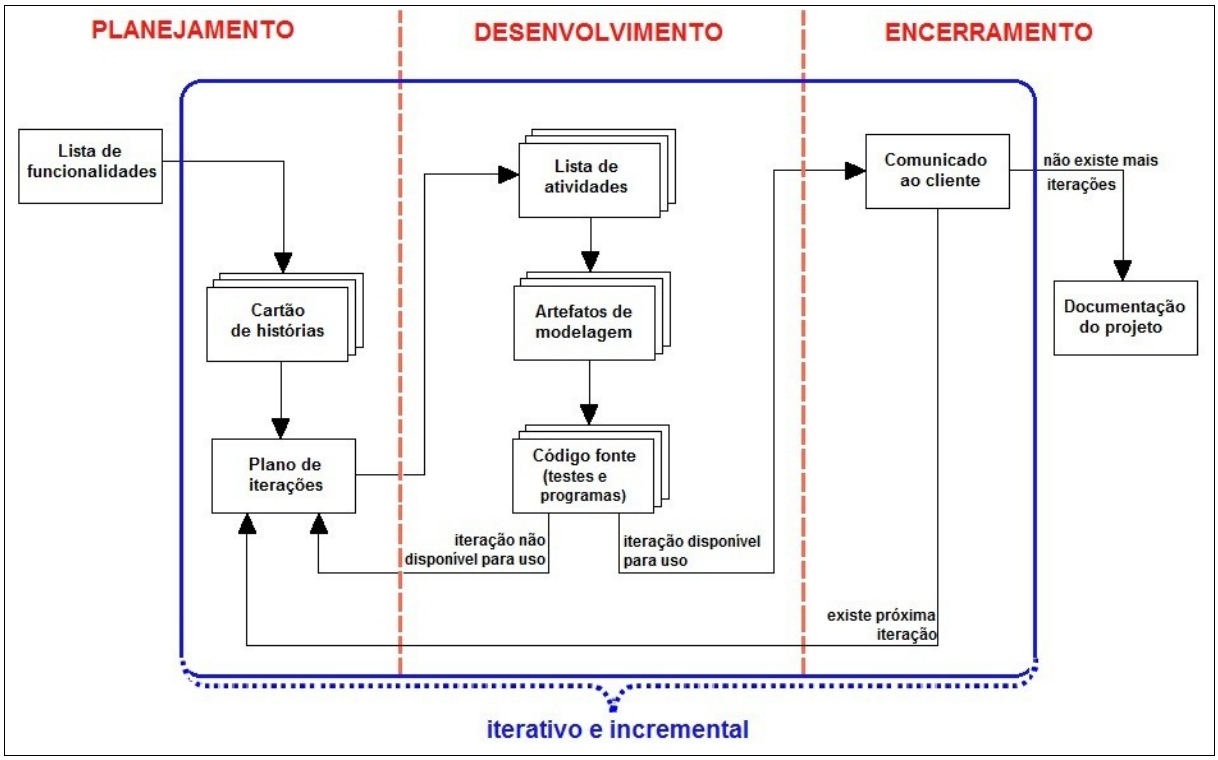


Figura 9: Fases e artefatos do novo modelo de processo

Abaixo estão detalhadas as três fases onde são apresentados os papéis envolvidos em cada fase, as práticas ágeis adotadas, e as atividades e artefatos gerados. Esses dois últimos representados na notação BPMN conforme OMG (2010), que representa a sequência de atividades executadas, os artefatos gerados por cada atividade e os papéis executores.

5.2.1 Fase de Planejamento

O principal objetivo da fase de planejamento é obter as informações necessárias junto

ao cliente para se ter a ideia da real necessidade e do escopo, para dessa forma poder planejar e estimar as iterações. Na fase de planejamento são executadas as atividades "Registrar a solicitação", "Realizar reunião inicial", "Estimar tempo para funcionalidades", "Escrever histórias" e "Planejar iterações".

Os papéis adotado nessa fase são:

- **Equipe Desenvolvimento:** corresponde aos membros da equipe técnica sendo composta pelo Analista de Sistemas, Programadores e Coordenador da equipe técnica.
- **Equipe Cliente:** composta pelos membros do cliente, que correspondem à Comissão Interdisciplinar ou seu representante ou demais usuários envolvidos no projeto.

Na figura 10 é apresentado o processo executado na fase de planejamento representando em BPMN, que recebe como entrada uma solicitação de manutenção evolutiva e gera como saída o Plano de Iterações.

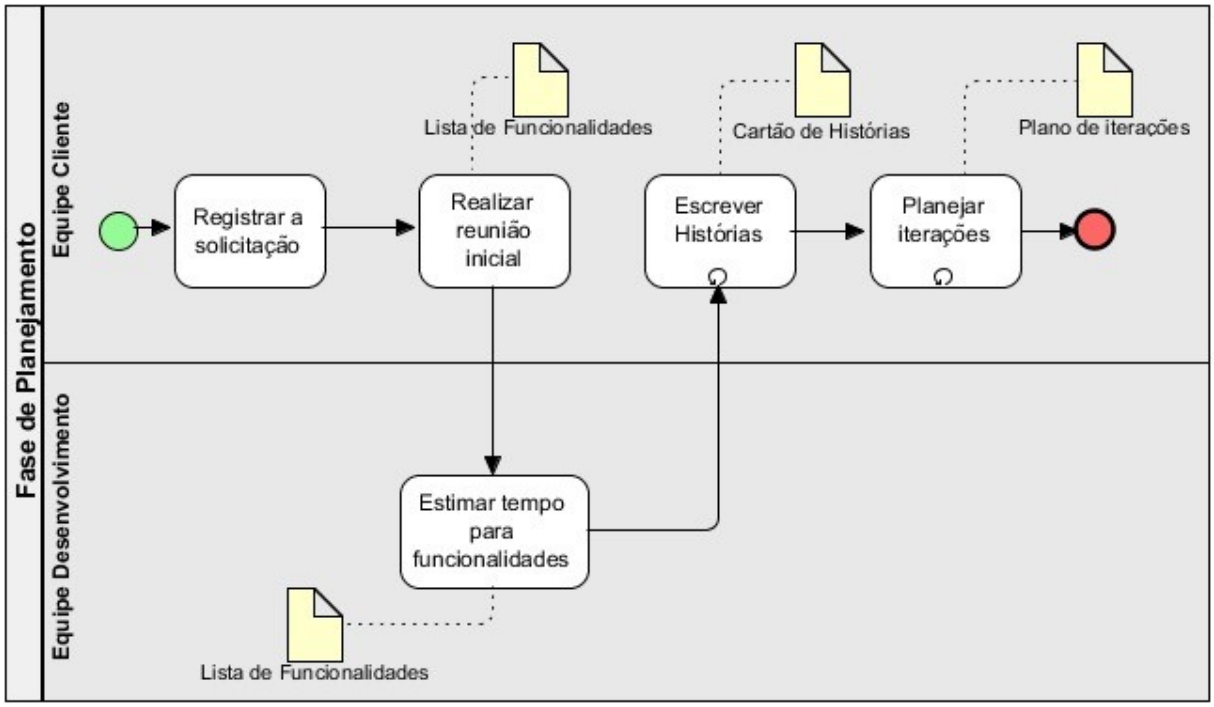


Figura 10: Processo executado na fase de planejamento

a) Atividade Registrar a solicitação

Esta atividade corresponde à formalização da solicitação de alteração no sistema executada pelo Cliente, que no caso da equipe estudada corresponde à Comissão

interdisciplinar ou seu representante. Esta solicitação pode ser registrada por *e-mail*, por contato telefônico, através de uma reunião, enfim, é o primeiro contato do cliente solicitando a alteração de sistema que pode corresponder a novas funcionalidades ou alterações em funcionalidades já existentes.

b) Atividade Realizar reunião inicial

Essa atividade compreende a reunião inicial entre a equipe cliente e a equipe de desenvolvimento onde o cliente apresenta suas necessidades e expectativas. Essa atividade gera como artefato a "Lista de Funcionalidades", que consiste em uma lista priorizada de todas as funcionalidades (requisitos) que o sistema deve atender. A priorização é realizada pelo cliente, que em atividade posterior deverá escrever uma história para cada funcionalidade. O objetivo da priorização feita pelo cliente é priorizar as funcionalidades que tem mais valor de negócio para o cliente.

Neste artefato são informados para cada funcionalidade as seguintes colunas:

- Id: um identificador numérico e sequencial;
- Descrição: uma breve descrição da funcionalidade;
- Prioridade cliente: uma prioridade classificada pelo cliente em Alta, Média e Baixa;
- Tempo estimado (hs): a estimativa de esforço (tempo em horas) para a entrega da funcionalidade. Esta estimativa é realizada pela equipe de desenvolvimento na atividade seguinte;
- Responsável: o nome da pessoa responsável por determinada funcionalidade. Esta pessoa deverá ser membro da equipe de desenvolvimento e é definida pela equipe de desenvolvimento;

Na figura 11 é apresentado uma sugestão do artefato "Lista de Funcionalidades", adaptado do *Product Backlog*, do Scrum.

Lista de Funcionalidades				
Id	**Descrição**	**Prioridade cliente**	**Tempo estimado (hs)**	**Responsável**
1	Cadastrar usuários	Alta	30	Maria
2	Cadastrar itens	Alta	30	Maria
3	…	Alta	…	…
4	…	Alta	…	…
5	Gerar orçamentos	Média	30	João
6	Gerar pedido	Média	30	Maria
7	Emitir nota fiscal	Baixa	60	Pedro
8	Gerar relatórios	Baixa	30	João

Figura 11: Exemplo do artefato Lista de Funcionalidades

Não foi sugerido a abordagem de riscos nesse artefato por essa informação não ser relevante para a equipe em questão, conforme pode ser observado na análise do processo atual apresentada no capitulo anterior.

c) Atividade Estimar tempo para funcionalidades

Nesta atividade a equipe de desenvolvimento realiza a estimativa de tempo para cada funcionalidade, atualizando as informações do artefato "Lista de Funcionalidades", na coluna "Tempo estimado (hs)", conforme apresentado na tabela 11. Esta estimativa é empírica, baseada na experiência da equipe de desenvolvimento.

d) Atividade Escrever histórias

Esta atividade corresponde a geração das histórias escritas pelo cliente, adaptado das *user stories* em XP, onde Beck (2004) define que para cada funcionalidade deverá ser escrita uma *user story*. O objetivo das histórias escritas pelo cliente é descrever as informações a respeito da funcionalidade, como o cliente espera que o sistema se comporte assim como os requisitos do teste de aceitação, que serão executados pelo analista de sistema durante a fase de desenvolvimento. Nessa atividade está se buscando o comprometimento do cliente e sua participação no processo, na etapa de levantamento de requisitos.

Nesta atividade é utilizado o artefato "Cartão de histórias" que deve ser preenchido considerando as seguintes orientações:

- nos campos "Id" e "Descrição" as informações devem ser as mesmas informadas no artefato "Lista de funcionalidades", visto que o cartão de histórias tem por objetivo detalhar a funcionalidade priorizada e estimada no artefato citado.
- no campo "Data criação" deverá ser informado a data de criação da história.
- no campo "Cliente" deverá ser informado o nome do membro da equipe do cliente que escreveu a histórias.
- no campo "História" deve ser informado pelo cliente de forma breve uma explicação para a funcionalidade que está sendo detalhada, contendo todas as informações que o cliente julgar importantes e essenciais.
- no campo "Critérios de aceitação" o cliente deve descrever os critérios de aceitação para a referida história. Estes critérios de aceitação serão considerados na execução dos testes que serão realizados pelo analista de sistemas na fase de

desenvolvimento.

- o campo "Testes de Sistema" deverá ser preenchido pelo analista de sistema quando for executada a atividade "Realizar testes de sistema", a qual está descrita na página 79.

Na figura 12 é demonstrado um exemplo do artefato "Cartão de Histórias" proposto adaptado de XP (BECK, 2004).

Cartão de Histórias

Id: 2 **Descrição:** Cadastrar itens

Data criação: 01/01/00 **Cliente:** João da Silva

História:

Informar o código, descrição, unidade de medida, grupo de estoque, almoxarifado padrão.
Permitir realizar alterações nas informações do cadastro, exceto código.
Permitir imprimir listagem de itens ordenados pelo código ou descrição.

Critérios de Aceitação:

Cadastrar vários itens sem repetir o código.
Imprimir um relatório ordenado por código e outro por descrição.
Alterar um cadastro de item. Não deve permitir alterar código.

Testes de Sistema:

Data:	Resultado:
15/01/00	Erro ao alterar item.
16/01/00	Executado com sucesso.

Figura 12: Exemplo do artefato Cartão de Histórias

e) Atividade Planejar iterações

Essa atividade corresponde a realização de uma reunião com a equipe cliente e a equipe de desenvolvimento para a definição das iterações e quais funcionalidades serão entregues em cada iteração. É necessário definir a duração da iteração, o que deve ser entre 2 a 4 semanas[14], conforme definição das equipes em conjunto. Considerando a duração da iteração e as estimativas das funcionalidades o Plano de Iterações é construído e poderá sofrer alterações em respostas às mudanças que poderão ocorrer durante as próximas fases.

Nesta atividade é gerado o artefato "Plano de Iterações" que tem por objetivo agrupar as funcionalidades que serão entregues em cada iteração assim como definir os períodos de trabalho de cada iteração. O artefato possui as seguintes colunas:

14 Baseado em XP e Scrum

- Iteração: informar um sequencial numérico que será o identificador da iteração;
- Data início: data de início de trabalho da iteração
- Data fim: data final de entrega da iteração;
- Funcionalidades - Id: informar o identificador da funcionalidade que está contida na determinada iteração;
- Funcionalidades - Descrição: informar a descrição da funcionalidade que está contida na determinada iteração;

Na figura 13 é apresentado um modelo para o artefato Plano de Iterações.

Plano de Iterações				
Iteração	Data início	Da fim	Funcionalidades	
			Id	Descrição
1	01/03/00	30/03/00	1	Cadastro de Usuários
			2	Cadastro de Itens
			3	…
			4	…
2	01/04/00	30/04/00	5	Gerar orçamentos
			6	Gerar pedido
3	01/05/00	30/05/00	7	Emitir NF
			8	Gerar relatórios
			9	Consultar auditorias

Figura 13: Exemplo do artefato Plano de Iterações

Essa atividade foi adaptada da reunião de planejamento da *sprint* do Scrum, que segundo Schwaber e Sutherland (2009), a reunião de planejamento da *sprint* é o momento no qual a iteração é planejada.

5.2.2 Fase de Desenvolvimento

É nessa fase que ocorre o maior trabalho de desenvolvimento compreendendo a modelagem (análise e projeto), implementação (programação) e testes. O objetivo da fase desenvolvimento é entregar as funcionalidades planejadas para a iteração corrente. O processo que é executado na fase de desenvolvimento é cíclico, ou seja, executado por iteração de acordo com o Plano de Iterações. As atividades executadas nessa fase são “Registrar atividades”, “Realizar modelagem”, “Codificar”, “Realizar testes unitários”, “Realizar testes de sistema” e “Apresentar para cliente”.

Os papéis que trabalham nessa fase são:

- **Cliente:** corresponde à Equipe do cliente já citada na fase de planejamento, ou à um membro da equipe cliente.
- **Analista de sistemas:** é membro da equipe de desenvolvimento, citada na fase de planejamento.
- **Programador:** é membro da equipe de desenvolvimento, citada na fase de planejamento.

Na figura 14 é representado em BPMN o processo executado na fase de desenvolvimento.

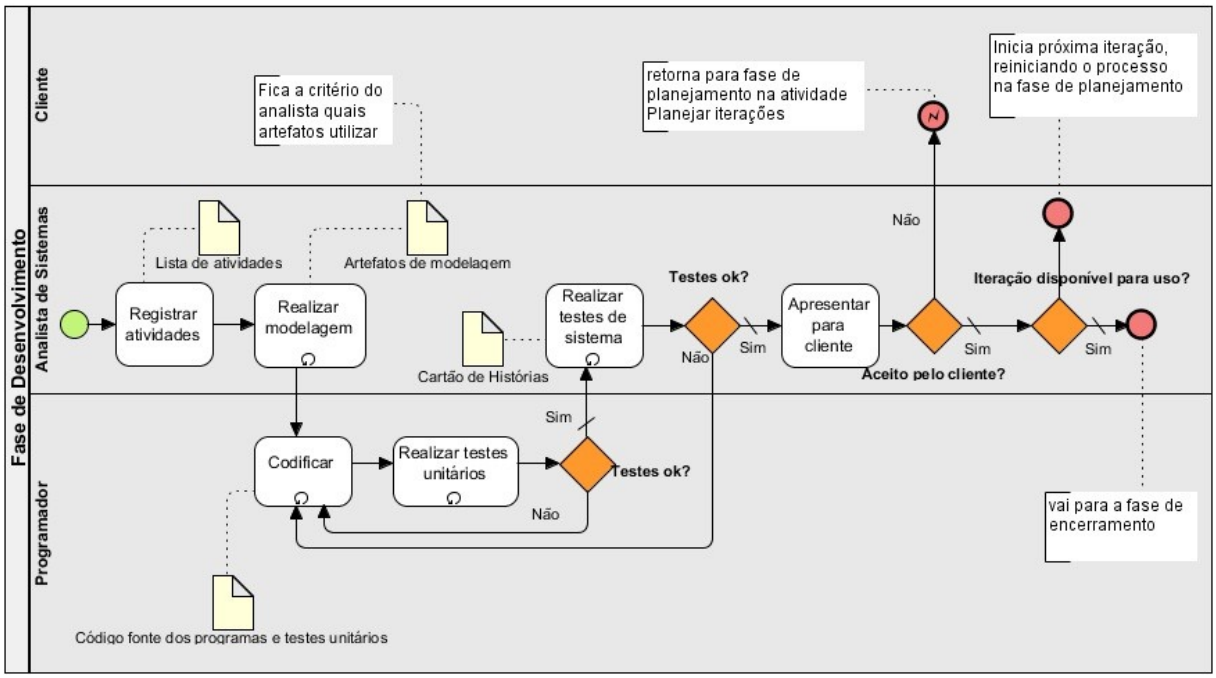


Figura 14: Processo executado na fase de desenvolvimento

a) Atividade Registrar atividades

Essa atividade corresponde ao registro das atividades necessárias para executar as histórias escritas para as funcionalidades da iteração corrente. O analista de sistemas deve ler as histórias e detalhar as atividades necessárias para a implementação da determinada história.

Ao registrar as atividades o analista de sistemas já irá atribuir as atividades aos executores, que poderão ser o próprio analista de sistemas ou os programadores. Essa definição das atribuições é realizada em conjunto com todos os membros da equipe de desenvolvimento. Nessa atividade é gerado o artefato "Lista de Atividades" que tem por objetivo descrever as atividades necessária para a execução de cada história,

consequentemente gerando as atividades a serem executadas na iteração corrente.

As informações que a "Lista de Atividades" deve conter são:

- No campo "História": informar o identificador e descrição da história.
- No campo "Estimado": deve ser informado o total de horas estimado para a história.
- Na coluna "Id": informar o identificador para a atividade, deve ser numérico e sequencial.
- Na coluna "Descrição": Informar a descrição da atividade.
- Na coluna "Responsável": informar o membro da equipe de desenvolvimento responsável pela atividades.
- Na coluna "P (hs)": informar as horas previstas para a atividade.
- Na coluna "R(hs)": informar as horas realizadas para a atividade.
- Na coluna "Situação": informar a situação atual da atividade entre as opções: "Não iniciado", "Em andamento", "Finalizado".
- Na coluna "Comentários": inserir a data e o comentário desejado.

A intenção é que este artefato seja atualizado diariamente, para que reflita o real andamento das atividades. Na figura 15 é apresentado uma sugestão para o artefato "Lista de Atividades".

Lista de Atividades

História: 2 – Cadastrar itens **Estimado:** 30hs

Id	Descrição	Respon-sável	P (hs)	R (hs)	Situação	Comentários
1	Definir estrutura (diagrama de classes)	João	8	5	Finalizado	05/01/00- executado esboço do diagrama em papel
2	Criar modelo físico (ER)	João	8	5	Em andamento	05/01/00- iniciado ER na ferramenta Jdeveloper
3	Criar protótipo de tela	Pedro	2		Não iniciado	

Figura 15: Exemplo do artefato Lista de Atividades

Esta atividade foi adaptada de XP, onde Beck(2004) sugere que as histórias sejam transformadas em tarefas gerando os cartões de tarefas e o comprometimento com os responsáveis pelas tarefas.

Através das informações de horas previstas e realizadas atualizadas no artefato "Lista

de Atividades" é possível gerar um relatório que permita realizar um acompanhamento do trabalho restante ao longo do tempo. Baseado nos gráficos *burndown* de Scrum, sugere-se a utilização do gráfico *burndown* como artefato para realizar esse acompanhamento.

Neste gráfico, a altura indica a quantidade de tarefas (em horas) da iteração (ou *Sprint Backlog* em Scrum) não completadas, e o comprimento são os dias. Com isto, é possível visualizar facilmente se um trabalho está tendo progresso, completando as tarefas, enquanto observa-se que as colunas do gráfico vão caindo em sua altura. Na figura 16 é apresentado um exemplo de gráfico *burndown* gerado a partir das informações de horas restantes de trabalho por dia da iteração. No eixo Y são apresentadas as horas restantes e no eixo X as datas.

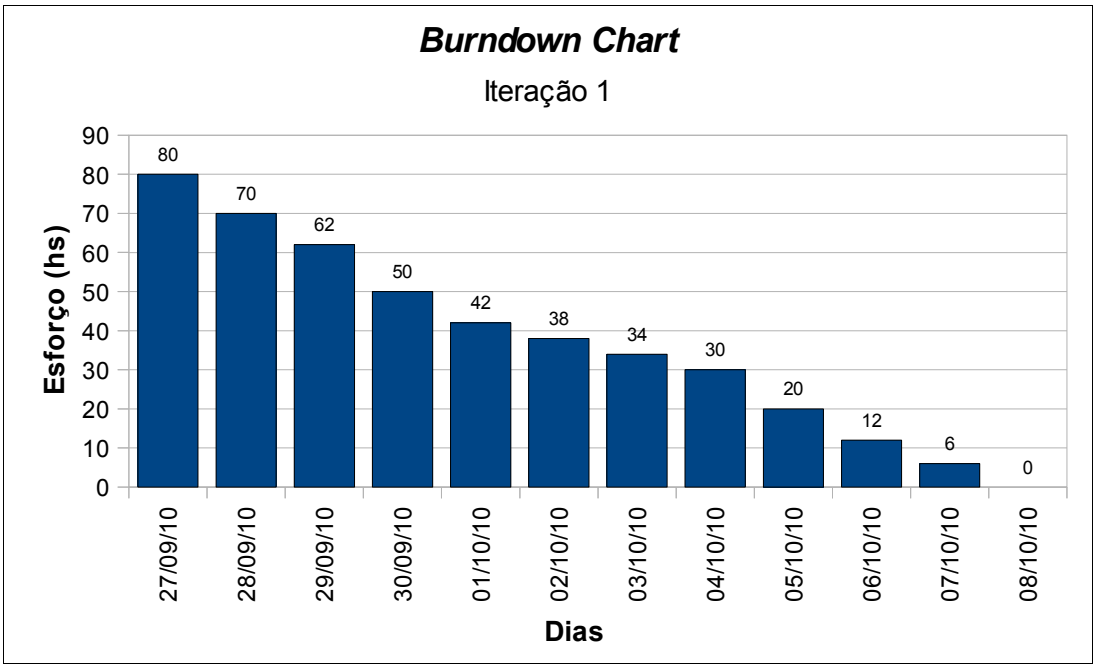


Figura 16: Exemplo do artefato Burndown da Iteração

Os dados que alimentam os gráficos *burndown* podem ser atualizados a cada reunião diária realizada pela equipe, onde a equipe poderá acompanhar o seu progresso.

b) Atividade Realizar modelagem

Esta atividade é facultativa, pois depende da complexidade dos requisitos e do conhecimento do analista para ser definido quais os artefatos serão úteis na modelagem. O termo "modelagem" compreende a análise e projeto do sistema, sendo conduzida pelo analista de sistemas mas podendo ser realizada em conjunto com os programadores.

Por se tratar de sistemas web com orientação a objetos, é sugerido a utilização da

UML (*Unified Modeling Language*), também sugerido por Ambler (2004) e Bezerra (2007). Bezerra (2007) afirma que a UML define uma notação padrão que pode ser utilizada no desenvolvimento de *softwares* orientados a objetos e apresenta exemplos práticos da utilização da UML para análise e projeto de sistemas orientados a objetos.

O objetivo desta atividade é gerar uma documentação de conteúdo simples e que satisfaça as necessidades da equipe (desenvolvimento e cliente). Segundo Ambler (2004), a modelagem é simples quando se comunica tudo o que deve ser comunicado, não possuir informação duplicada e o menor número de elementos possíveis.

E sobre a utilização de ferramentas, como a proposta é agilidade e simplicidade, a equipe deve utilizar uma ferramenta que supra as necessidades do projeto, independente de ser um desenho feito a lápis numa folha de papel, ou em uma ferramenta CASE (*Computer Aided Software Engineering*) (AMBLER, 2004).

c) Atividade Codificar

Essa atividade compreende a codificação dos testes unitários e dos programas, gerando como artefato o código fonte dos testes e programas. Quem executa esta atividade é o programador. É sugerido que quando for requisitos de alta complexidade a programação seja em pares, sendo uma definição em conjunto pela da equipe. É sugerido que sejam escritos primeiro os testes unitários e depois os programas, pois de acordo com Beck (2004), dessa forma o programador irá analisar a melhor forma de codificar os programas.

Na execução dessa atividades são sugeridas as práticas de XP: seguir um padrão de codificação, a refatoração, a propriedade coletiva do código. Essas práticas devem ser utilizadas conforme necessidades da equipe.

Por ser um processo de manutenção, ocorre de um programador realizar a manutenção em funcionalidades implementadas por outro, dessa forma já é direcionado para a prática de refatoração e propriedade coletiva do código.

d) Atividade Realizar testes unitários

Nessa atividade os programadores executam os testes unitários que foram escritos na atividade anterior, “Codificar”. O objetivo dos testes unitários é garantir que o programa está executando a(s) funcionalidade(s) de acordo com o planejado e esperado pelo cliente. Através dos testes unitários o programador já tem um *feedback* imediato sobre o código gerado.

Quando identificados erros ou o programa não executar conforme o esperado, o programador deve corrigir os erros ou desvios de comportamento dos programas gerados, consequentemente submetendo o novo código novamente ao teste unitário. Portanto, essa atividade é cíclica, só devendo seguir adiante quando o resultado dos testes unitários for positivo.

e) Atividade Realizar testes de sistema

Essa atividade é executada pelo analista de sistemas e corresponde a execução dos testes de aceitação que devem atender os critérios de aceitação escritos pelo cliente no artefato "Cartão de Histórias". Através dessa atividade também são executados os testes de integração com os demais programas e sistemas. Os resultados dos testes devem ser registrados no artefato "Cartão de Histórias" no campo "Testes de Sistema", conforme figura 12 exibida na página 73.

Assim como acontece com os testes unitários, essa atividade também é cíclica, ou seja, se o resultado dos testes de sistema não for satisfatório, o analista de sistemas devolve os programas para o programador rever o código e corrigir as inconsistências.

f) Apresentar para cliente

Essa atividade corresponde a realização de uma reunião com o cliente para ser apresentado o resultado do que foi produzido na iteração. Essa atividade está baseada na Reunião de Revisão da *Sprint*, aplicada em *Scrum*.

A reunião não deve durar mais de 4 horas, e além de apresentar para o cliente o que foi produzido, é o momento da equipe de desenvolvimento expor os problemas enfrentados e como foram resolvidos, e toda a equipe discutir o que foi bom e o que deve ser melhorado. Ao final da reunião o cliente deve dar um feedback na iteração finalizada, dessa forma aceitando ou não a iteração.

Se o cliente não aceitar a iteração, deve expor os problemas identificados e a execução das atividades deve retornar para a fase de planejamento, na atividade "Planejar iterações", onde será atualizado o Plano de iterações para contemplar as mudanças necessárias para resolver os problemas apontados.

Porém, se a iteração for aceita pelo cliente e a iteração estiver disponível para uso, a fase de desenvolvimento é finalizada e iniciada a fase de encerramento. E se a iteração não

estiver disponível para uso, as atividades são direcionadas para a próxima iteração, que se inicia pela atividade "Planejar iterações" onde será revisado o Plano de iterações e dar início a próxima iteração.

Todas as atividades executadas na fase de desenvolvimento são cíclicas, ocorrem por iteração. Durante toda a fase de desenvolvimento deverão ser realizadas reuniões diárias entre a equipe técnica. Essa prática está baseada em XP e Scrum, e tem por objetivo promover a comunicação entre a equipe técnica e a identificação de problemas ou dificuldades que os membros da equipe estão enfrentando. Conforme sugerido em XP e Scrum, essa reunião deve ser breve, de 15 a 20 minutos, e os membros da equipe devem responder a três questões:

- O que você realizou desde a última reunião diária?
- O que você vai fazer antes da próxima reunião diária? e
- Quais obstáculos estão em seu caminho?

Segundo Schwaber e Sutherland (2009), as Reuniões Diárias melhoram a comunicação, eliminam outras reuniões, identificam e removem impedimentos para o desenvolvimento, ressaltam e promovem a tomada rápida de decisões e melhoram o nível de conhecimento de todos acerca do projeto.

5.2.3 Fase de Encerramento

A fase de encerramento se inicia a partir da aceitação da iteração por parte do cliente, e essa estiver disponível para uso. O objetivo dessa fase é disponibilizar as funcionalidades em ambiente de produção para o cliente começar utilizar e dar *feedbacks*. Com o *feedback* do cliente é possível melhorar as próximas iterações. Segundo Ambler (2004), entrar em produção significa disponibilizar os sistema para uso no ambiente real de trabalho do cliente.

Os papéis que estão envolvidos nessa fase são:

- **Equipe Desenvolvimento**: formada pelo analista de sistemas e programadores.
- **Equipe Cliente:** como já citado nas fases anteriores, representa os membros da Comissão interdisciplinar, ou seu representante.

Na fase de encerramento são executada as atividades "Entregar incremento", "Utilizar sistema e dar feedback", "Iniciar próxima iteração" e "Encerrar projeto".

Na figura 17 é representado, em BPMN, o processo executado na fase de

encerramento.

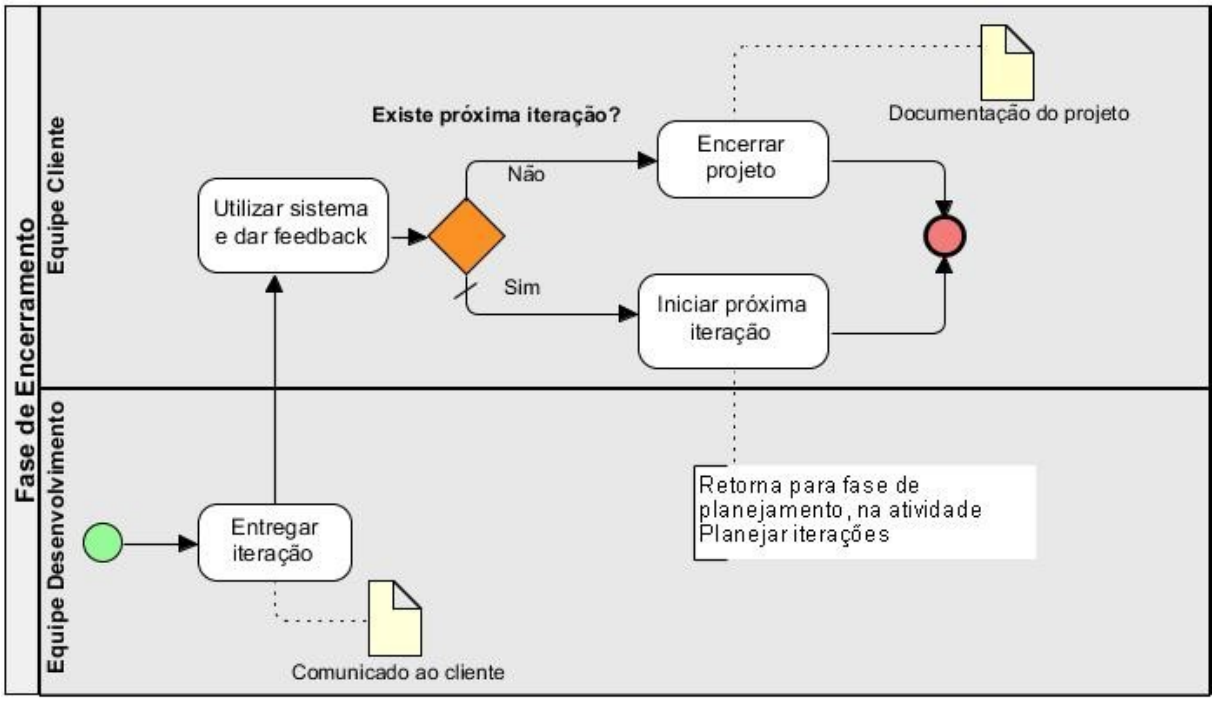


Figura 17: Processo executado na fase de encerramento

a) Atividade Entregar iteração

Essa atividade corresponde a disponibilizar as funcionalidades produzidas na iteração para uso, em ambiente de produção. É sugerido como artefato dessa atividade o "Comunicado ao cliente", que tem por objetivo informar aos interessados que as novas funcionalidades estão disponíveis para uso. Este artefato pode ser um *e-mail*, ou uma notícia no portal utilizado pelos usuários, ou a critério da equipe. A estrutura do conteúdo pode ser padronizada independente da ferramenta utilizada. É interessante que possua o título e o corpo da mensagem padrão, variando apenas o nome das funcionalidades e datas. Assim os usuários já sabem do que se trata quando recebido o comunicado. Na figura 18 é apresentado uma sugestão para esse artefato.

Prezados usuários do Sistema **<nome_do_sistema>**,

Informamos que a partir de **<data>** estão disponíveis as novas funcionalidades descritas abaixo. Qualquer dúvida entre em contato com o suporte pelo telefone **<num_telefone>.**

- <lista_funcionalidades>

Atenciosamente,
Equipe de desenvolvimento

Figura 18: Sugestão de estrutura para artefato Comunicado ao cliente

As palavras que estão em negrito e entre os símbolos "< >" devem ser substituídas conforme descrito abaixo:

- <nome_do_sistema>: deve ser substituído pelo nome do sistema que sofreu manutenção.
- <data>: deve ser substituído pela data de liberação para uso em produção.
- <lista_funcionalidades>: deve ser substituído por uma listagem com o nome de todas as funcionalidades novas ou alteradas.

Ao final da entrega de cada iteração é sugerido que a equipe de desenvolvimento realize uma reunião somente com seus membros (analista de sistemas e programadores) para avaliar a iteração entregue, os problemas enfrentados como foram resolvidos, as coisas boas, o aprendizado e o que pode ser melhorado no processo de desenvolvimento, ferramentas e práticas para a próxima iteração. Esta reunião é uma adaptação da Reunião de Retrospectiva, de Scrum, onde Schwaber e Sutherland (2009) definem que nessa reunião o objetivo é inspecionar como correu a última *Sprint* em se tratando de pessoas, das relações entre elas, dos processos e das ferramentas. A inspeção deve identificar e priorizar os principais itens que correram bem e aqueles que, se feitos de modo diferente, poderiam ter deixado as coisas ainda melhores.

b) Atividade Utilizar sistema e dar *feedback*

Essa atividade consiste em o cliente utilizar as funcionalidades entregues e dar um *feedback* para a equipe de desenvolvimento. O *feedback* deve abordar aspectos positivos e negativos. Quando ocorrer feedback negativo, nas reuniões de planejamento das iterações devem ser avaliados para tomar ações necessárias para minimizar ou anular esses impactos negativos. Se essas ações implicarem em alteração do escopo das iterações, o plano de iterações deve ser atualizado, refletindo as alterações realizadas.

Se após essa atividade ainda existirem iterações, a próxima atividade a ser executada é "Iniciar próxima iteração", caso não existam mais iterações a próxima atividade é "Encerrar projeto".

c) Atividade Iniciar próxima iteração

Essa atividade corresponde em reiniciar o processo na fase de planejamento para desenvolver a próxima iteração. Por padrão esta atividade remete para a atividade "Planejar

iterações" na fase de planejamento, onde a próxima iteração é iniciada.

d) Encerrar projeto

Essa atividade corresponde ao fim do projeto, isso significa que todas as funcionalidades definidas na "Lista de Funcionalidades" foram entregues de acordo com o Plano de Iterações, que o cliente já está utilizando as funcionalidades em ambiente de produção e está satisfeito.

Nessa atividade é sugerido a utilização do artefato "Documento do Projeto" que tem por objetivo conter algumas informações relevantes das alterações realizadas pela manutenção executada. Esse artefato proposto é baseado em XP, que segundo Beck (2004), deve conter de 5 a 10 páginas e as informações poderão ser úteis para próximas manutenções.

Assim como no artefato "Comunicado ao cliente", neste artefato "Documento do projeto" também é sugerido uma estrutura do documento, que pode ser adaptado pela equipe conforme o projeto. A forma de armazenamento e divulgação do artefato também fica a critério da equipe, que pode optar por imprimir e arquivar em papel, arquivar em meio digital, ou incluir em alguma ferramenta que é utilizada pela equipe. O importante é realizar um breve registro do projeto e os objetivos da manutenção que foi realizada. Abaixo são descritas as informações a serem registradas nesse artefato.

- Projeto: informar o nome do projeto
- Objetivo: informar com breve descrição o objetivo do projeto, a necessidade que gerou a demanda de manutenção evolutiva.
- Data início e Data fim: informar as datas de início e fim do projeto.
- Equipe cliente: informar o nome dos membros que participaram da equipe cliente durante a execução do projeto.
- Equipe desenvolvimento: informar o nome dos membros que participaram da equipe de desenvolvimento durante a execução do projeto.
- Lista de Funcionalidades entregues: informar a descrição das funcionalidades entregues.
- Informações adicionais: espaço aberto para informações que são relevantes ao projeto.
- É sugerido anexar os artefatos Lista de Funcionalidades e Cartões de Histórias, mas essa definição foca a critério da equipe pois depende dos meios que serão

armazenados os artefatos.

Na figura 19 é apresentada um esboço da estrutura sugerida para o artefato "Documento do projeto".

DOCUMENTO DE PROJETO

Projeto: _____

Objetivo: _____

_____

_____

Data início: _____ **Data fim:** _____

Equipe Cliente **Equipe Desenvolvimento**

_____ _____

_____ _____

_____ _____

Lista de Funcionalidades entregues:

_____

_____

Informações adicionais: _____

_____

_____

Obs.: poderá ser anexado os artefatos Lista de Funcionalidades e Cartões de Histórias

Figura 19: Estrutura do artefato Documento do Projeto

Ao finalizar a atividade "Encerrar projeto", o projeto considera-se concluído e a equipe estará disponível para assumir próximas demandas de manutenção, com mais conhecimento e experiência que foram adquiridos na execução das iterações do projeto em questão.

5.3 Considerações finais

Neste capítulo foi apresentada a fase "identificar aprimoramentos" do estágio mudança de processo conforme o ciclo de aprimoramento sugerido por Sommerville (2007). Nessa fase foram apresentados os aprimoramentos identificados os quais guiaram a proposta do novo modelo de processo que foi proposto.

A proposta de customização do processo que foi apresentada foi baseada nos

aprimoramentos identificados e apresentados na seção 5.1 e com a introdução de práticas sugeridas nos métodos ágeis de desenvolvimento XP e Scrum, sempre perseguindo os objetivos definidos na fase de medição (GQM).

O processo que foi customizado é um processo de desenvolvimento de *software* utilizado em manutenções evolutivas de um sistema web desenvolvido com orientação a objetos. O objetivo da customização do processo de manutenção é propor uma melhoria do processo atual que foi estudado e apresentado no capítulo 4 introduzindo práticas sugeridas nos métodos ágeis de desenvolvimento XP e Scrum, sempre perseguindo os objetivos definidos na fase de medição (GQM).

Assim como os métodos ágeis, o novo modelo de processo proposto foi criado baseado nos princípios do Manifesto Ágil. A premissa foi propor um processo iterativo e incremental, que permita ser aplicado em equipes com características semelhantes com a equipe que foi estudada, composta por analistas de sistemas e programadores que realizam manutenção evolutiva em sistemas *web* com orientação a objetos.

No novo modelo de processo foram propostas algumas práticas sugeridas nas metodologias XP e Scrum, adaptando-as para as características da equipe e da instituição do estudo de caso.

Na tabela 13 são apresentadas as atividades e artefatos gerados no processo proposto com sua fundamentação nos métodos ágeis de forma consolidada. Nas colunas são informadas as metodologias ágeis e nas linhas as atividades e artefatos gerados por fase.

Na coluna “O que” é informado se é atividade ou artefato, e na coluna “Descrição” é informada a descrição da atividade ou artefato.

Nas colunas “XP”, “Scrum” e “MA” é inserido um “X” quando deseja identificar a origem da prática adotada ou adaptada.

Na coluna “Adaptação das praticas ágeis” é descrito o nome da prática ágil que a atividade ou artefato está baseada ou foi adaptada.

Na última linha da tabela é apresentada a prática “Reuniões Diárias” que não aparece dentro de nenhuma das fases por ocorrer em todas elas.

Tabela 13: Resumo das atividades e artefatos gerados em cada fase do novo processo proposto *versus* práticas ágeis

	O que	Descrição	Métodos Ágeis			Adaptação das práticas ágeis:
			XP	Scrum	MA	
Fase Planejamento	Atividade	Realizar reunião inicial	X	X		Jogo do Planejamento Cliente Presente
		Estimar funcionalidades	X	X		Jogo do Planejamento
		Escrever Estórias	X			Jogo do Planejamento Cliente Presente
		Planejar iterações	X	X		Jogo do Planejamento Reunião de Planejamento da Sprint
	Artefato	Lista de Funcionalidades		X		*Product Backlog*
		Cartão de Histórias	X			Cartões de histórias *(User stories*)
		Plano de iterações		X		*Sprint Backlog*
Fase Desenvolvimento	Atividade	Registrar atividades	X			Jogo do Planejamento
		Realizar modelagem	X		X	Projeto simples
		Codificar	X			Refatoração Programação em pares Propriedade coletiva Padrões de codificação
		Realizar testes unitários	X	X		Testes
		Realizar testes de sistema	X	X		Integração contínua
		Apresentar para cliente		X		Reunião de Revisão da *Sprint*
	Artefato	Lista de atividades	X			Cartão de tarefas(*Task card*)
		Artefatos de Modelagem			X	Projetos simples Aplicar o artefato correto Criar Conteúdo simples Utilizar ferramentas simples
		Código fonte	X	X		Iteração *Sprint*
		Gráficos *burdown*		X		*Burndown Chart*
Fase Encerramento	Atividade	Entregar iteração	X	X		Entregas frequentes Iterativo e incremental Reunião de retrospectiva da *Sprint*
		Utilizar sistema e dar feedback	X	X		Cliente presente Iterativo e incremental
		Iniciar próxima iteração				Criação do autor
		Encerrar projeto	X			Fase Morte (XP) *PostGame* (Scrum)
	Artefato	Comunicado ao cliente				Criação do autor
		Documentação do projeto	X			Fase Morte (XP) *PostGame* (Scrum)
	Prática	Reuniões Diárias	X	X		Reuniões diárias

6 . AVALIAÇÃO DE FERRAMENTAS PARA O PROCESSO PROPOSTO

Conforme já citado em capítulos anteriores, Pressman (2006) define as ferramentas como mecanismos automatizados de apoio ao processo e aos métodos que compõem a metodologia de desenvolvimento de *software*. Portanto, para apoiar a metodologia de desenvolvimento proposta neste trabalho é necessária a seleção de uma ferramenta que atenda as necessidades do novo processo customizado, assim como as características da equipe estudada.

Este capítulo tem por objetivo apresentar o processo de avaliação e a escolha da ferramenta selecionada para apoiar a metodologia de desenvolvimento proposta neste trabalho.

O capítulo é composto por duas seções, onde na seção 6.1 é descrito o processo de avaliação executado e a seção 6.2 apresenta as considerações finais deste capítulo.

6.1 Processo de avaliação

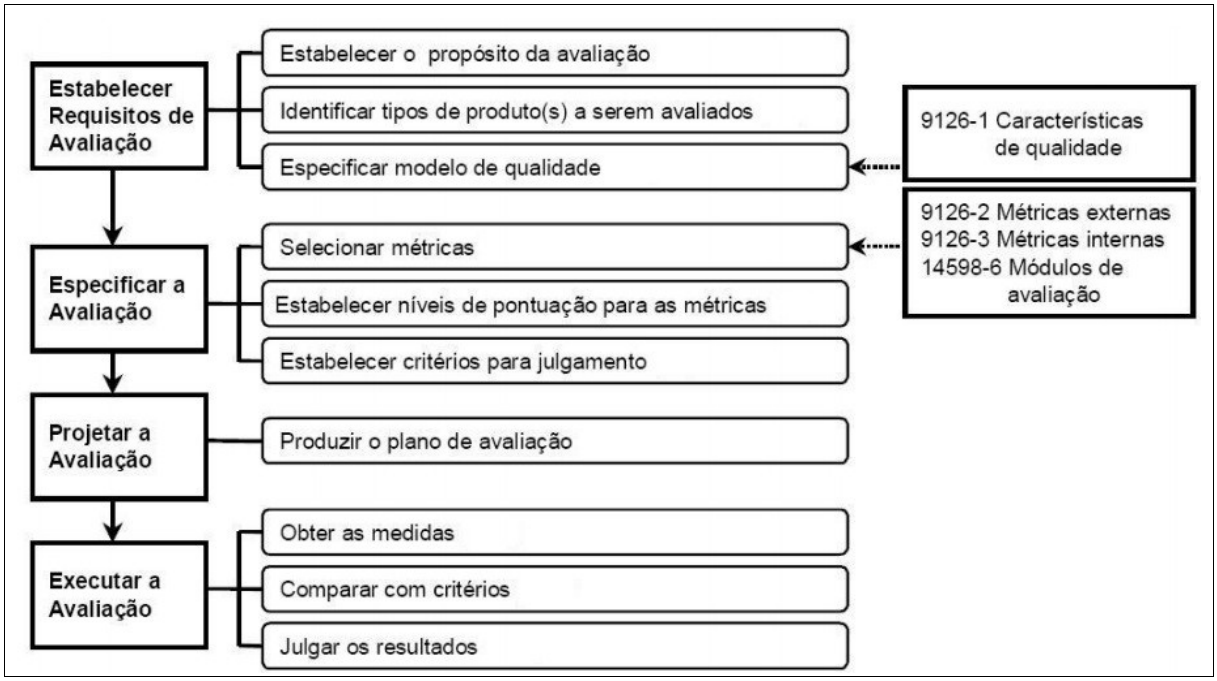
Esta seção apresentada o processo de avaliação que foi executado para a seleção de uma ferramenta de apoio à metodologia proposta neste trabalho.

Para executar o processo de avaliação fez-se necessário um estudo das normas de qualidade dos produtos de *software* das séries NBR ISO/IEC 9126 e NBR ISO/IEC 14598 que proporcionaram o conhecimento para a definição de um modelo de qualidade e o processo de avaliação de *software*. O resumo do estudo dessas normas segue como anexo D deste trabalho.

O processo de avaliação executado é baseado no processo descrito na norma NBR ISO/IEC 14598-1 e adaptado para as necessidades do contexto deste trabalho.

A norma NBR ISO/IEC 14598-1 fornece uma estrutura para avaliar a qualidade de *software*, define um modelo de processo de avaliação genérico e estabelece os requisitos para a medição e avaliação (ABNT, 2001).

O processo de avaliação proposto pela norma NBR ISO/IEC 14598-1 é apresentado na figura 20. Contém quatro etapas, distribuídas em dez atividades, e fornece requisitos gerais de avaliação.



Fonte: ABNT (2001)

Figura 20: Processo de avaliação segundo a norma NBR ISO/IEC 14598-1

A equipe de desenvolvimento, objeto de estudo deste trabalho, foi envolvida em algumas etapas da avaliação, já que a ferramenta selecionada será utilizada pela equipe em questão.

A avaliação está divida em quatro etapas assim como sugerido na norma NBR ISO/IEC 14598-1: Estabelecer requisitos de avaliação, Especificar a avaliação, Projetar a avaliação e Executar a avaliação. A execução de cada uma dessas etapas é descrita nas subseções a seguir assim como os resultados obtidos.

6.1.1 Estabelecer requisitos de avaliação

Nesta etapa são definidos os requisitos gerais da avaliação como o objetivo, tipos de produtos a serem avaliados e o modelo de qualidade que foi utilizado.

O objetivo da avaliação é selecionar uma ferramenta de apoio à análise, modelagem e gerenciamento de projetos de desenvolvimento de sistemas baseados nos princípios ágeis. Essa ferramenta será utilizada no novo processo de desenvolvimento proposto neste trabalho, desas forma fazendo parte da metodologia proposta para a equipe que é objeto de estudo deste trabalho.

Os tipos de produtos avaliados são *softwares* de gerenciamento de projetos destinados a apoiar as metodologias ágeis. Conforme sugerido na norma NBR ISO/IEC 14598-1, buscou-se por opinião especializada para a seleção das ferramentas a serem avaliadas. Contudo, as políticas da instituição e as características da equipe também foram levadas em consideração. Se utilizou como base uma comparação de ferramentas *open source* de gestão ágil de projetos publicada na Linux.com[15]. Nesse artigo também são citadas as ferramentas pagas mais conhecidas e líderes no mercado.

Foram selecionadas para participar da avaliação oito ferramentas, onde três são pagas (Rational Team Concert, ScrumWorks e VersionOne) e cinco *open source* (Agilefant, Agilo, ExplainPMT, IceScrum e Xplanner-Plus).

O modelo de qualidade utilizado foi baseado no modelo de qualidade da norma NBR ISO/IEC 9126-1 conforme sugerido pelas normas estudadas[16] e também por Rocha, Maldonado e Weber (2001). Algumas das características selecionadas para a avaliação foram: funcionalidade, usabilidade e confiabilidade.

6.1.2 Especificar a avaliação

Nesta etapa são definidas as métricas, os níveis de pontuação, a escala e os critérios de julgamento.

Foram definidas vinte e uma métricas, distribuídas em dez requisitos funcionais e onze não funcionais. Requisitos funcionais são definidos por Pressman (2006) como declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Já os requisitos não funcionais são definidos por Sommerville (2007) como sendo restrições sobre os serviços ou as funções oferecidos pelo sistema. Em geral se relacionam com padrões de qualidade, confiabilidade, desempenho, usabilidade e eficiência.

Os requisitos funcionais foram definidos baseados nas características e necessidades do novo processo de desenvolvimento proposto no capítulo 5, da equipe objeto de estudo deste trabalho e do projeto do estudo de caso aplicado. Os requisitos não funcionais foram definidos baseados no modelo de qualidade sugerido na norma ISO/IEC 9126 (ABNT, 2003)

15 Disponível em http://www.linux.com/news/software/applications/67269-comparing-open-source-agile-project-management-tools.

16 O resumo do estudo das normas NBR ISO/IEC 9126 e 14598 segue no anexo D deste trabalho.

considerando as características e necessidades da equipe.

Na tabela 14 são listados todos os requisitos funcionais (RF) e requisitos não funcionais (RNF) definidos para a avaliação. Na coluna "Cód." são exibidos os códigos dos requisitos onde para os requisitos funcionais se utilizou "RN" e para os não funcionais "RNF". Para ambos o código é complementado por um sequencial numérico. Na coluna "Descrição" é exibido o nome do requisito.

Tabela 14: Requisitos da avaliação

Cód.	Descrição
Requisitos funcionais	
RF_01	Permitir controle por projetos
RF_02	Possuir iterações ou sprints
RF_03	Possuir cartão de histórias
RF_04	Possuir cartão de tarefas
RF_05	Possuir controle de horas trabalhadas por recurso
RF_06	Possuir gráficos de acompanhamento
RF_07	Possuir consultas de acompanhamento
RF_08	Possuir relatórios de acompanhamento
RF_09	Possuir cadastro de equipes (recursos)
RF_10	Possuir mural virtual de lembretes (planejamento de atividades)
Requisitos Não funcionais	
RNF_01	Ser um software livre (free)
RNF_02	Possuir versão web
RNF_03	Permitir que seja utilizado nos navegadores Firefox e Internet Explorer
RNF_04	Utilizar banco de dados free
RNF_05	Possuir versão com idioma português
RNF_06	Possuir controle de acesso por usuários no sistema
RNF_07	Possuir cadastro de permissões por projeto
RNF_08	Possuir manutenção e atualizações
RNF_09	Possuir interface amigável
RNF_10	Possuir facilidade na operação
RNF_11	Deve se comportar de maneira estável (ausência de erros)
RNF_12	Possuir mensagens compreensíveis

A seguir estão descritos cada um dos requisitos.

- **RF_01 Permitir controle por projeto:** Este requisito funcional foi definido com base na necessidade de controle de trabalho por projeto. Espera-se que a ferramenta permita registro de atividades, recursos e datas por projeto. Nesse contexto, projeto é uma demanda de trabalho com data de início e fim e contém um conjunto de iterações ou *sprints*.

- **RF_02 Possuir iterações ou *sprints*:** Este requisito funcional está baseado na característica iterativa e incremental do processo de desenvolvimento proposto e nas práticas ágeis. O objetivo é inserir e manipular as informações do artefato "Plano de Iterações", exibido na figura 13, no cadastro de iterações ou *sprints* da ferramenta.

- **RF_03 Possuir cartão de histórias:** Este requisito funcional também está baseado nas características do processo de desenvolvimento proposto. O objetivo deste requisito é que a ferramenta permita criar, editar e gerenciar os cartões de histórias, dessa forma substituindo o artefato "Cartão de história" exibido na figura 12.

- **RF_04 Possuir cartão de tarefas:** Assim como os requisitos anteriores, este requisito funcional também está baseado nas características do processo de desenvolvimento proposto. O objetivo é substituir o artefato "Lista de atividades" exibido na figura 13. Uma tarefa deverá pertencer ou estar associada à uma história. Deverá ser possível criar, editar e gerenciar as tarefas, incluindo as informações de horas previstas por tarefa.

- **RF_05 Possuir controle de horas trabalhadas por recurso:** O objetivo deste requisito é permitir que seja registradas as horas realizadas em cada tarefa para que seja possível realizar o acompanhamento do trabalho restante. Através da informação de horas trabalhas será possível realizar o acompanhamento da atividade, história, iteração e consequentemente do projeto. Essa informação também alimentará os gráficos gerados pelo sistema.

- **RF_06 Possuir gráficos de acompanhamento:** Esse requisito foi definido com o objetivo de substituir o artefato do gráfico *burndown*, figura 16. O gráfico deverá ser gerado a partir das informações de horas restantes de trabalho da iteração. No eixo x deverão ser apresentadas as horas restantes e no eixo y são demonstrados as datas (dias). A ferramenta deve possibilitar gerar gráficos por iteração e por projeto.

- **RF_07 Possuir consultas de acompanhamento:** A ferramenta deve possuir consultas de acompanhamento por projeto, iterações e recursos.

- **RF_08 Possuir relatórios de acompanhamento:** A ferramenta deve possuir relatórios de acompanhamento por projeto, iterações e recursos. São considerados relatórios as consultas que podem ser impressas em impressora e/ou arquivo.

- **RF_09 Possuir cadastro de equipes:** Esse requisito refere-se a necessidade de cadastrar equipes de trabalho por projeto, ou seja, alocar recursos para cada projeto. Uma equipe pode ser composta por um ou vários recursos e o mesmo recurso pode pertencer a equipes diferente.

- **RF_10 Possuir mural virtual de lembretes (planejamento de atividades):** O objetivo desse requisito é que a ferramenta forneça em uma única tela os lembretes de atividades que estão atribuídas para determinada pessoa que se logar no sistema. Exemplo: se o recurso "a" se logar no sistema, deverá ser exibido uma painel com todas as demandas atribuídas ao recurso "a".

- **RNF_01 Ser um *software* livre (*free*):** Este requisito não funcional foi definido em função das políticas da instituição que incentivam e tem preferência por utilização de ferramentas *free* (gratuitas).

- **RNF_02 Possuir versão *web*:** Esse requisito foi definido por solicitação da equipe e refere-se a forma de acesso ao sistema ser via *web* não necessitando de instalações individuais em cada computador cliente.

- **RNF_03 Permitir que seja utilizado nos navegadores *Firefox* e *Internet Explorer*:** Este requisito foi mapeado devido às políticas da instituição que sugerem a utilização dessas duas opções de *browser* e também porque alguns membros da equipe utilizam *Firefox* e outros *Internet Explorer.*

- **RNF_04 Usar banco de dados *free*:** Assim como o requisito RNF_01, este requisito também foi definido em razão das políticas da instituição.

- **RNF_05 Possuir versão com idioma português:** Esse requisito foi definido por ser de preferência da equipe, mas podendo ser também em idioma inglês.

- **RNF_06 Possuir controle de acesso por usuário no sistema:** Este é um

requisito considerado fundamental por se tratar de segurança. O esperado da ferramenta é que seja possível cadastrar usuários e para cada usuário uma senha de acesso, para dessa forma cada membro da equipe acesse a ferramenta utilizando o seu usuário (*login*).

- **RNF_07 Possuir cadastro de permissões por projeto:** Assim como o requisito anterior, também refere-se a segurança, porém com importância menor. O esperado da ferramenta é que permita relacionar usuários à um determinado projeto e atribuindo permissões no referido projeto.

- **RNF_08 Possuir interface amigável:** Este requisito foi definido por ser considerado importante pela equipe de desenvolvimento, objeto de estudo deste trabalho. A medida dessa métrica é subjetiva a cada avaliador.

- **RNF_09 Possuir facilidade na operação:** Este requisito está relacionado à usabilidade do sistema e foi definido por ser considerado uma variável relevante na adaptação da equipe ao novo processo proposto. A medida dessa métrica também é subjetiva a cada avaliador. F

- **RNF_10 Deve se comportar de maneira estável (ausência de erros):** Este requisito foi definido por ser considerado essencial pela equipe e tem por objetivo verificar a confiabilidade do *software*.

- **RNF_11 Possuir mensagens compreensíveis:** Este requisito foi definido em razão da usabilidade da ferramenta. A medida dessa métrica também é subjetiva a cada avaliador.

Foi definida uma escala nominal de 0 a 2 e dividida em duas categorias em relação ao nível de satisfação, “Satisfatório” e “Insatisfatório”. A nota '0' corresponde à ausência do requisito, a nota '1' corresponde à presença parcial do requisito e a nota “2” corresponde a presença total do requisito. Portanto deverá ser atribuída a nota “0” quando a ferramenta não oferecer o requisito que está sendo avaliado, nota “1” quando atender parcialmente e nota “2” quando atender plenamente.

A figura 21 demonstra a escala e as categorias definidas.

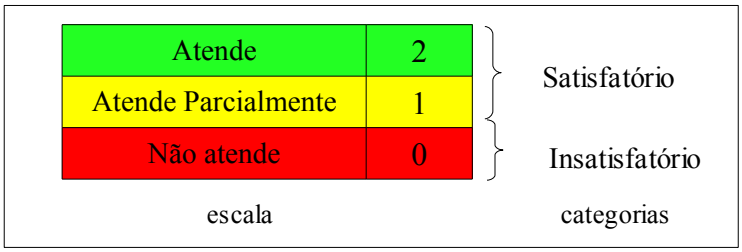


Figura 21: Escala de pontuação e categorias

Foram definidos pesos de acordo com a importância de cada requisito para compor o cálculo da pontuação. A importância "Baixa" possui peso "1", a "Média" peso "2" e a "Alta" peso "3". Dessa forma a pontuação do requisito de prioridade alta será mais significativa do que a pontuação de um requisito de importância baixa.

Na tabela 15 são exibidos os pesos e as importância nas colunas "Valor do Peso" e "Importância", respectivamente.

Tabela 15: Pesos e importância

Valor do Peso	Importância
1	Baixa
2	Média
3	Alta

Os critérios para determinação da importância dos requisitos são os seguintes:

- **Importância Baixa:** o requisito é considerado com importância baixa quando não se aplica às características da equipe do estudo de caso deste trabalho;
- **Importância Média**: o requisito é considerado com importância média quando a sua ausência não compromete a utilização do novo processo proposto, ou seja, a equipe consegue executar o processo sem o determinado requisito.
- **Importância Alta:** o requisito é considerado com importância alta quando não pode faltar, ou seja, a sua ausência compromete a execução do novo processo proposto. A ferramenta a ser selecionada deverá atender a todos os requisitos de importância alta.

Para cada requisito foi atribuído um peso de acordo com a sua importância, conforme exibido na tabela 16. Foi utilizada a tabela 14 e adicionada a coluna "Peso" onde descreve o peso definido para cada requisito.

Tabela 16: Pesos de cada requisito

Requisitos		Peso
Requisitos funcionais		
RF_01	Permitir controle por projetos	3
RF_02	Possuir iterações ou sprints	3
RF_03	Possuir cartão de histórias	3
RF_04	Possuir cartão de tarefas	3
RF_05	Possuir controle de horas trabalhadas por recurso	3
RF_06	Possuir gráficos de acompanhamento	2
RF_07	Possuir consultas de acompanhamento	3
RF_08	Possuir relatórios de acompanhamento	2
RF_09	Possuir cadastro de equipes (recursos)	1
RF_10	Possuir mural virtual de lembretes (planejamento de atividades)	2
Requisitos Não funcionais		
RNF_01	Ser um software livre (*free*)	3
RNF_02	Possuir versão web	3
RNF_03	Permitir que seja utilizado nos navegadores Firefox e Internet Explorer	3
RNF_04	Utilizar banco de dados free	3
RNF_05	Possuir versão com idioma português	2
RNF_06	Possuir controle de acesso por usuários no sistema	3
RNF_07	Possuir cadastro de permissões por projeto	2
RNF_08	Possuir interface amigável	3
RNF_09	Possuir facilidade na operação	3
RNF_10	Deve se comportar de maneira estável (ausência de erros)	3
RNF_11	Possuir mensagens compreensíveis	2

6.1.3 Projetar a avaliação

Nesta etapa é criado um plano de avaliação contendo as informações necessárias para orientar os testes na avaliação. Foram definidos, junto com a equipe, dois casos reais que foram utilizados para avaliar todas as ferramentas.

Foram definidos dois projetos para serem utilizados como roteiro nas avaliações, que são: “Ava no Cetec” e “Blog do Ava”. Para cada projeto foi criado um documento consolidando as informações necessárias para guiar os teste. Esses documentos são exibidos nas figuras 22 e 23, respectivamente.

No projeto “Ava no Cetec” foi definida uma equipe composta pelos papéis coordenador de projeto, analista de sistemas e programador. O projeto compreende uma lista de três funcionalidades a serem entregues em duas iterações com durações de duas semanas cada. Cada iteração possui histórias e cada história tarefas atribuídas a algum recurso.

Na figura 22 é exibido o documento com as informações consolidadas do projeto 1 - Ava no Cetec que foi utilizado na avaliação das ferramentas.

Projeto 1: Ava no Cetec

Equipe: Daiane - Coordenador do projeto
João - Analista de Sistemas
Maria - Programador

Lista de Funcionalidades:
- Lista de Participantes
- Mural
- Fórum

Iteração 1 (24/09 - 01/10)
- Lista de Participantes
- Mural

História 1: ***Mural de recados por turma***
Descrição: Cada turma deverá possuir um mural de recados onde todos os participantes podem postar. Participantes são professores e alunos.
Teste de aceitação:
- acessar ambiente com perfil de professor e postar notícia no mural.
- acessar ambiente com perfil de aluno e postar notícia no mural.
- consultar as notícias postadas.

Tarefa 1.1: Implementar os métodos. **Responsável:** João
Tarefa 1.2: Implementar interface. **Responsável:** Maria

História 2: ***Lista de Participantes***
Descrição: Cada turma deve possuir uma lista de participantes que corresponde aos alunos e professor.
Teste de aceitação:
- cadastrar participantes
- consultar lista de participantes

Tarefa 2.1: Implementar os métodos. **Responsável:** João
Tarefa 2.2: Implementar interface. **Responsável:** Maria

Iteração 2 (04/10 – 15/10)
- Fórum

História 1: ***Fórum da disciplina***
Descrição: Cada turma deverá possuir um fórum onde somente o professoe pode criar tópicos e todos os participantes podem responder. Participantes são professores e alunos.
Teste de aceitação:
- professor criar fórum
- alunos responder fórum

Tarefa 1.1: Implementar os métodos. **Responsável:** João
Tarefa 1.2: Implementar interface. **Responsável:** Maria

Figura 22: Documento do plano de avaliação do Projeto 1

O segundo projeto, chamado pela equipe de "Blog do Ava", possui a mesma equipe utilizada no projeto 1- "Ava no Cetec". É composto por uma única funcionalidade e uma

única iteração. O objetivo desse segundo projeto no plano de avaliação é poder avaliar o requisito RF_01(Permitir controle por projetos).

Na figura 23 é apresentado o documento criado com as informações consolidadas do projeto 2 -"Blog do Ava".

Projeto 2:	Blog do Ava
Equipe:	Daiane - Coordenador do projeto João - Analista de Sistemas Maria - Programador
Lista de Funcionalidades:	
- Adaptar o Blog para postar imagens	
Iteração 1 (22/10 – 05/11)	
- Adaptar o Blog para postar imagens	
História 1:	***Postar imagens no Blog***
Descrição:	O blog deve possuir galeria de imagens e permitir post com arquivos de imagens. **Teste de aceitação:** - acessar blog e postar uma imagem na galeria
Tarefa 1.1:	Implementar os métodos. **Responsável:** João
Tarefa 1.2:	Implementar interface. **Responsável:** Maria

Figura 23: Documento do plano de avaliação do Projeto 1

6.1.4 Executar a avaliação

Nesta etapa foi executada a avaliação propriamente dita de cada uma das ferramentas, onde cada uma foi instalada, configurada e testada de acordo com o plano de avaliação. Foram aplicadas as métricas a cada ferramenta obtendo como resultado valores de pontuação de cada requisito avaliado. Por fim esses valores foram totalizados por ferramenta, dessa forma obtendo uma pontuação para cada ferramenta avaliada.

Após a aplicação das métricas as notas foram totalizadas por ferramenta e as três ferramentas com melhor nota foram apresentadas para a equipe de desenvolvimento, objeto de estudo deste trabalho, a qual emitiu um parecer. As notas não foram absolutas na definição da ferramenta, a opinião da equipe também foi levada em consideração.

A seguir é apresentada a avaliação de cada uma das ferramentas com uma síntese das informações mais relevantes, suas principais características, pontos fortes e pontos fracos, a nota obtida na avaliação e comentários gerais sobre a avaliação.

a) Agilefant

Agilefant é uma ferramenta *open source* cujo desenvolvimento é coordenado pelo projeto de pesquisa ATMAN conduzido pelo grupo de pesquisa de processo de *software* (*Software Process Research Group*) da Universidade de Tecnologia de Helsinki (Helsinki *University of Technology*), Finlândia[17]. O *download* da ferramenta foi realizado através do site oficial cujo endereço é http://www.agilefant.org/, onde contém toda documentação de suporte e instalação.

Os testes foram realizados na versão 2.0 e a instalação foi executada sem dificuldades, onde a documentação de suporte à instalação foi suficiente. A avaliação foi realizada com sucesso pois a ferramenta é bastante intuitiva e a documentação do produto foi pouco utilizada. Os conceitos suportados pela ferramenta são de projetos com múltiplas iterações e essas contém histórias. As histórias possuem as tarefas associadas ao recurso responsável, que registra as horas trabalhadas. Suporta o cadastro de usuários e equipes, porém não suporta permissões a nível de projeto. Uma das características interessante é o painel de trabalho, que é exibido na página inicial ao se logar na ferramenta.

Na figura 24 é exibido uma captura de tela do painel de trabalho por recurso.

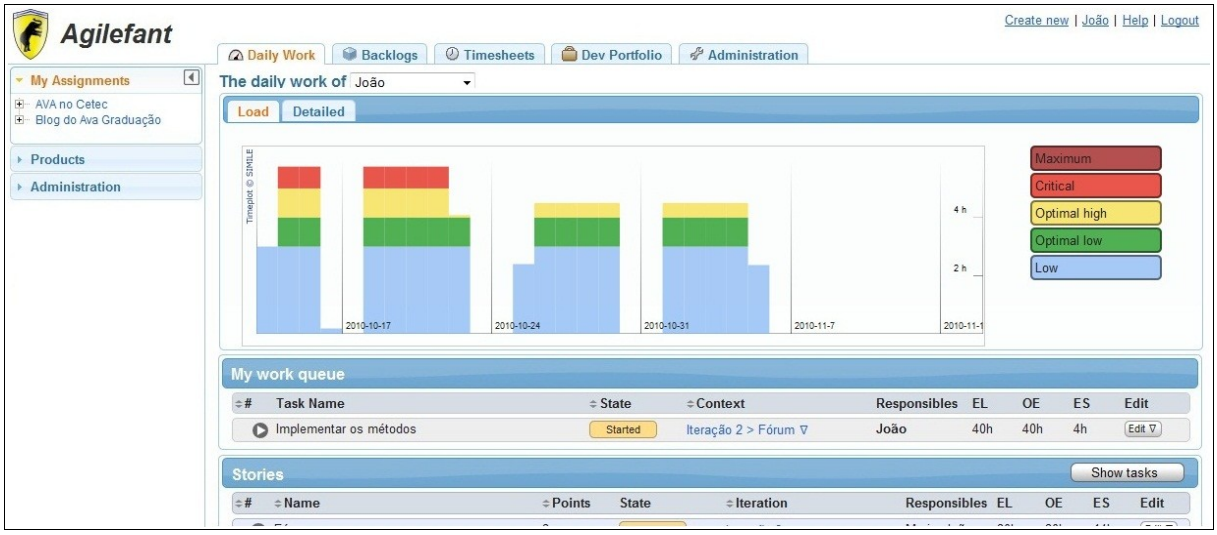


Figura 24: Captura de tela do painel de trabalho diário

Agilefant foi a única ferramenta *open source* que atendeu com nota máxima todos os requisitos funcionais. Atendeu plenamente a nove dos dez requisitos funcionais, e atendeu parcialmente o requisito RF_09, pois não possui muitas opções de relatórios no sistema e gera somente em *excel* (.xls). Sua avaliação pontuou em 98, o que representa 89,1% sobre o total

17 http://www.helsinki.fi/university/

da pontuação máxima possível.

b) Agilo

Agilo é uma ferramenta *open source* oferecida pela empresa Agile42 com sede em Berlim e filiais na Europa e América do Norte. A ferramenta Agilo é uma versão com menos recursos que a Agilo Pro, que é comercializada., e está disponível para download em http://www.agile42.com/cms/pages/agilo/.

Agilo é uma ferramenta bastante flexível para apoiar a metodologia Scrum. Possui uma limitação no conceito de projetos, visto que permite cadastrar somente um único projeto. A ferramenta é baseada na criação de itens de suporte que se classificam em "Requisitos", "Histórias" e "Tarefas" e permite fazer referências entre as tarefas e através dessas referências manter os relacionamentos.

Um aspecto falho foi no conceito de "tarefas", pois não permite relacionar tarefas à histórias, somente à iteração. Permite cadastrar marcos, iterações, recursos, equipes. Agilo possui um quadro de tarefas bastante intuitivo e com o recurso de "arrastar e soltar". Possui bastante opções na administração de cadastros de usuário, permissões, equipes, e demais cadastros em geral. Na figura 25 é exibida a captura de tela da área de administração de cadastros oferecida pela ferramenta Agilo.

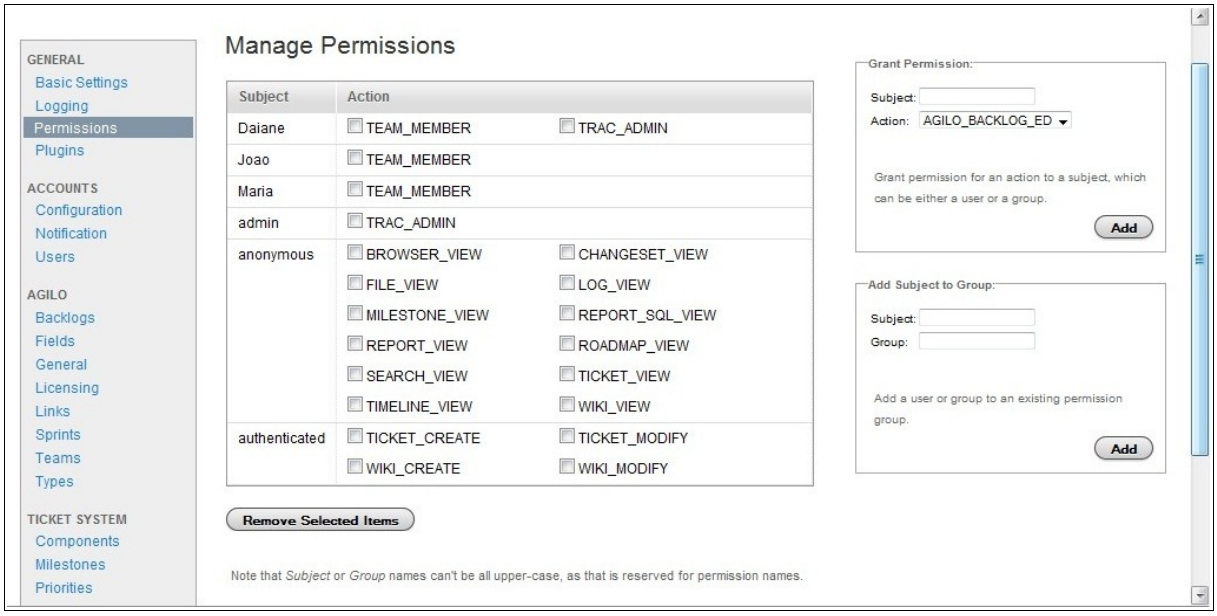


Figura 25: Captura de tela do cadastro de permissões

O resultado da avaliação da ferramenta Agilo foi 71 pontos, onde 33 pontos foram obtidos nos requisitos funcionais e 38 nos requisitos não funcionais. Totalizou com quatro

requisitos nota 0, sete requisitos com nota 1 e onze com nota 2. Esse resultado representa 64,5% sobre o total de pontos possíveis.

c) Explainpmt

Explainpmt é uma ferramenta *open source* que foi desenvolvida para dar suporte a metodologia de desenvolvimento XP. Está disponível para *download* em https://github.com/explainpmt/explainpmt. Possui uma versão demo disponível no endereço http://www.andrewtrommer.com/explainpmt/users/login, onde pode ser acessado com o usuário "admin" e senha "admin".

A ferramenta usa os conceitos de projetos, *releases*, iterações, cartão de histórias, tarefas e testes de aceitação. É possível cadastrar vários projetos, onde cada projeto pode possuir várias iterações. O projeto possui somente um *backlog* que possui as histórias associadas. As histórias possuem tarefas e testes de aceitação e são agrupadas em uma *release*.

Possui como pontos positivos a interface bastante intuitiva e os testes de aceitação. Já como pontos negativos constam a ausência da estimativa e controle de horas, permite somente uma iteração vigente, não tem painel de atividades, não possui gráfico *burndown* por iteração, somente por projeto. Também não possui documentação e suporte ao produto.

Na figura 26 é exibida a tela de consulta de uma história, onde pode ser observadas as tarefas e testes de aceitação associados à história.

Figura 26: Captura de tela da consulta de uma história

Na avaliação realizada, Explainpmt apresentou quatro requisitos com nota 0, seis requisitos com nota 1 e onze com nota 2. A pontuação final totalizou em 76 pontos, representando 69,1% sobre a pontuação máxima possível.

d) IceScrum

IceScrum é uma ferramenta *open source* destinada ao gerenciamento de projetos Scrum. Através de seu site é possível acessar uma versão demo pelo endereço http://www.icescrum.org/demo/. A versão testada e avaliada foi a R2#15.1 liberada em maio/2010.

A ferramenta suporta os conceitos proposto por Scrum, onde o mais alto nível é o de produto também chamado de projeto, sendo possível criar vários projetos. Cada projeto possui um único *backlog*, que corresponde ao conjunto de funcionalidades, histórias, defeitos e histórias técnicas. O projeto é dividido em *sprints* que são agrupadas em uma *release* e compostas por histórias. A ferramenta também possui o conceito de impedimentos do produto. Como permite somente uma único *sprint* ativo não é indicado para grandes projetos que possuem mais de uma equipe trabalhando em diversos *sprints* ao mesmo tempo. Possui a visualização do backlog da *sprint* como um quadro de tarefas onde se é possível "arrastar e soltar" as tarefas, se assemelhando à um quadro físico de manipulação dos *post-it* coloridos.

A ferramenta não suporta cadastro de permissões ou de equipes, os usuários de IceScrum poderão assumir os papéis disponíveis que são P*roduct Owner*, *Scrum Master*, T*eam Member*, *Stakeholder*. Pelo papel do usuário a ferramenta libera determinadas ações no sistema.

De uma forma geral, a ferramenta se destacou mais nos requisitos funcionais por possuir bem definidas as características de Scrum. Porém foram identificadas algumas deficiências em termos de usabilidade e padronização das funcionalidades, como por exemplo o "arrastar e soltar" que só funciona em alguns quadros do sistema. Algumas das opções do sistema são acessadas somente pelo do menu de contexto através do clique com o botão direito do mouse, o que não é intuitivo.

Na figura 27 é apresentado uma tela da consulta do *backlog* do produto na ferramenta IceSrum.

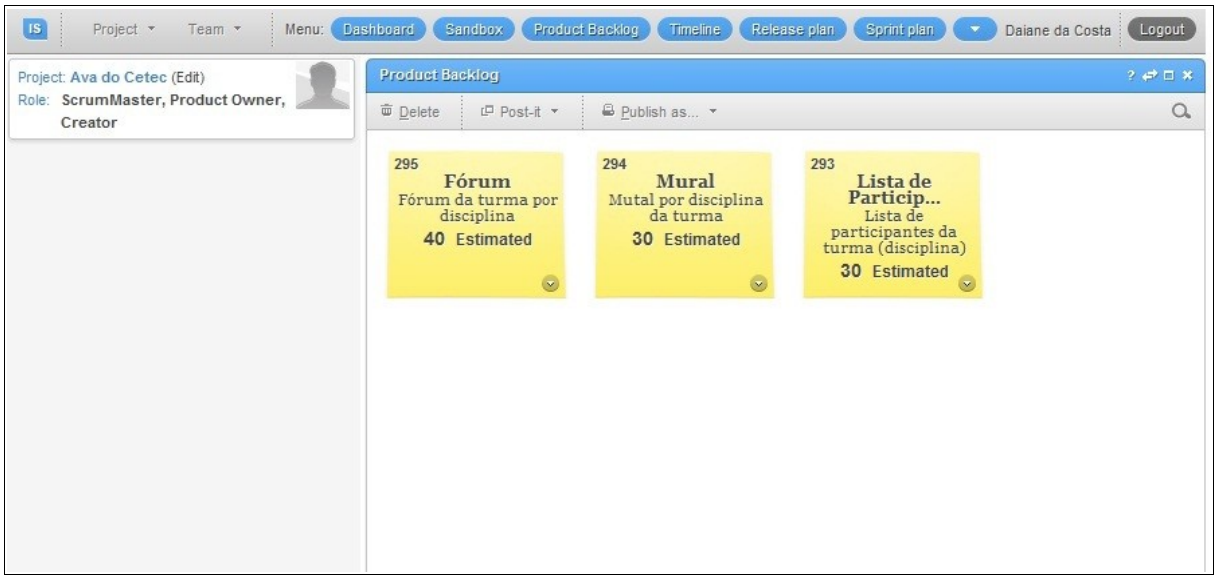


Figura 27: Captura de tela do *product backlog*

Na avaliação realizada, IceScrum apresentou três requisitos com nota 0, cinco requisitos com nota 1 e treze com nota 2. A pontuação final totalizou em 85, o que representa 77,3% sobre o total da pontuação máxima possível.

e) Rational Team Concert

Rational Team Concert é uma ferramenta da suíte *Rational* da IBM. Segundo IBM (2010), foi projetada para dar suporte ao desenvolvimento ágil auxiliando na colaboração entre as equipes. Por ser uma ferramenta comercializada, a avaliação foi realizada em uma versão experimental de 60 dias da versão 2.0.0.2 disponível no site da IBM[18].

No site da IBM foi encontrada toda a documentação para suporte e instalação da ferramenta. Rational Team Concert possui uma excelente documentação e pode ser acessada através do menu "Ajuda" localizado no canto superior direito nas telas do sistema.

Pelos testes realizados pode ser observado que a ferramenta é bastante completa. Rational Team Concert suporta os conceitos de projetos, *releases*, *sprints*, histórias, tarefas, defeitos, testes de aceitação, papéis do usuário, gráficos, consultas que podem ser customizadas e relatórios diversos. As estimativas de histórias são possíveis somente por pontos, as horas são estimadas somente em tarefas. As tarefas não são relacionadas à uma história, somente à uma *sprint*. Pode ser utilizada pela *web* ou com instalação local (cliente/servidor) sendo que os recursos na versão *web* são mais limitados. Os painéis de

18 http://www.ibm.com/expressadvantage/br/catalogo/software_rational_team_concert_express.phtml

tarefas e acompanhamento estão disponíveis somente pela *web*.

Por ser bastante robusta não foi possível realizar testes sem a leitura da documentação, pois existem muitas configurações iniciais e opções disponíveis, o que tornou a ferramenta complexa para as necessidades deste trabalho.

A ferramenta possui uma interface bastante intuitiva e de boa usabilidade. Também possui forte característica de personalização (consultas, relatórios, painéis) e administração (segurança e permissões em diversos níveis).

Na figura 28 é demonstrada a tela de cadastro de tarefas da ferramenta Rational Team Concert.

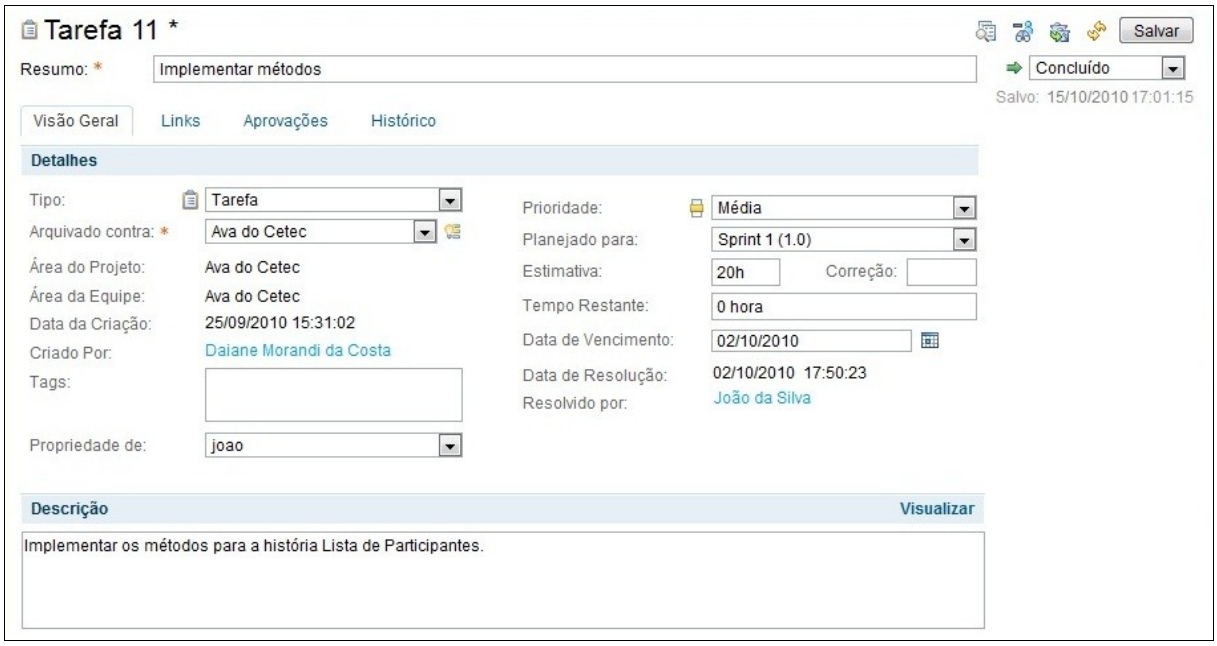


Figura 28: Captura de tela do cadastro de tarefa

Na avaliação realizada a ferramenta Rational Team Concert apresentou um requisito com nota 0, três requisitos com nota 1 e dezessete com nota 2. A pontuação final totalizou em 97, atingindo 88,2% do total de pontos possíveis.

f) ScrumWorks

ScrumWorks é uma ferramenta desenvolvida e comercializada pela empresa Danube Technologies, Inc. Fundada em 2000, fornece *software* e treinamento focada exclusivamente no método Scrum de desenvolvimento ágil. Na sua carteira de clientes consta a Oracle, Intel, Adobe, Sun Microsystems entre outros que podem ser conferidos no site da empresa pelo

endereço http://www.danube.com/company/customers.

Por ser uma ferramenta comercial os testes realizados foram em uma versão trial, ScrumWorks Basic versão 1.8.4, com uma limitação bastante expressiva de recursos conforme pode ser observado em http://www.danube.com/scrumworks, em relação à versão comercializada, a ScrumWorks Pro.

Através do site foi possível encontrar toda a documentação necessária para a instalação e utilização do produto.

A ferramenta suporta os conceitos de projetos, *sprints*, *product backlog*, histórias, tarefas cadastro de equipes além de consultas, relatórios e gráficos.

Durante os testes realizados na avaliação verificou-se boa usabilidade e uma interface bastante intuitiva onde não foi necessário muita consulta à documentação. A versão testada, *Basic*, não possui controle de horas trabalhadas por recursos, controle de segurança e permissões.

Na figura 29 é apresentado a tela de consulta do *product backlog* do projeto, que está dividido em *sprints* onde para cada *sprint* lista suas histórias e atividades.

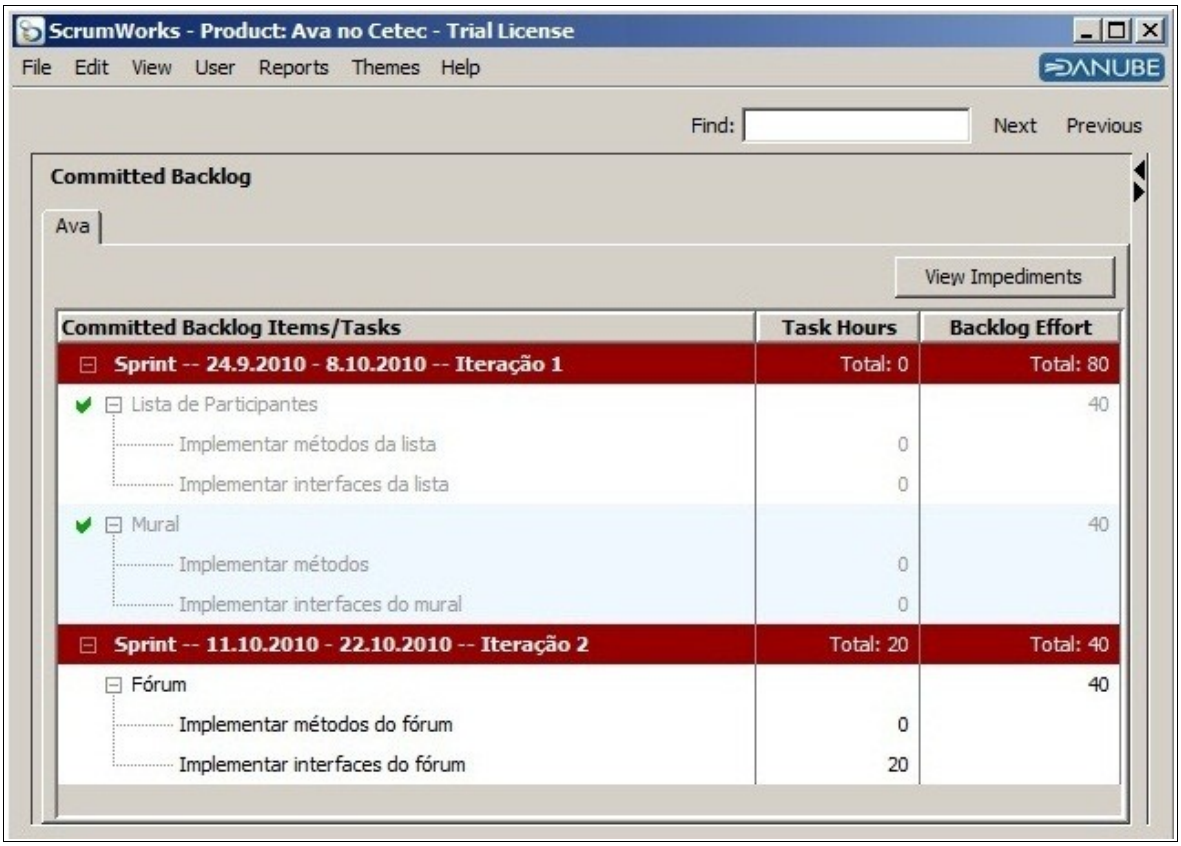


Figura 29: Captura de tela da consulta do *product backlog*

Na avaliação realizada a ferramenta ScrumWorks, versão *Basic*, apresentou cinco requisitos com nota 0, três requisitos com nota 1 e treze com nota 2. A pontuação final totalizou em 79 representando 71,8% sobre o total da pontuação possível. A limitação de recursos da versão *Basic* (trial) em relação à versão comercial, *Pro*, pode ter prejudicado a avaliação, mas entendeu-se prudente pontuar de acordo com os recursos que foram possíveis avaliar na versão *trial* disponibilizada pela Danube.

g) VersionOne

VersionOne é uma ferramenta de gerenciamento de projetos desenvolvida para empresas que fazem uso das metodologias ágeis como Scrum e XP. Desenvolvida e comercializada pela empresa VersionOne, Inc. que atua no mercado desde 2002 possui como clientes AMD, Cisco, Toshiba, Motorola, Nokia, Siemens, entre outros conforme divulgado em http://www.versionone.com/Customers/.

Por ser uma ferramenta comercial foi testada a versão VersionOne *Enterprise trial* (30dias) disponibilizada no site da VersionOne. No site está publicado toda documentação de instalação e suporte ao produto, o que auxiliou nos testes da ferramenta.

A ferramenta suporta os conceitos de projetos, *releases*, iterações, histórias, tarefas, testes, equipes, defeitos, além de uma diversidade de consultas e gráficos em diferentes níveis.

Na figura 30 é exibida a tela inicial de trabalho ao se logar no sistema. Neste exemplo da figura, está posicionado na aba "*My work*" e são exibidas as histórias e tarefas atribuídas ao usuário "joao". Para acessar a história basta clicar em cima, pois possui *link* que direciona para o cadastro de histórias, e assim para as demais opções do sistema.

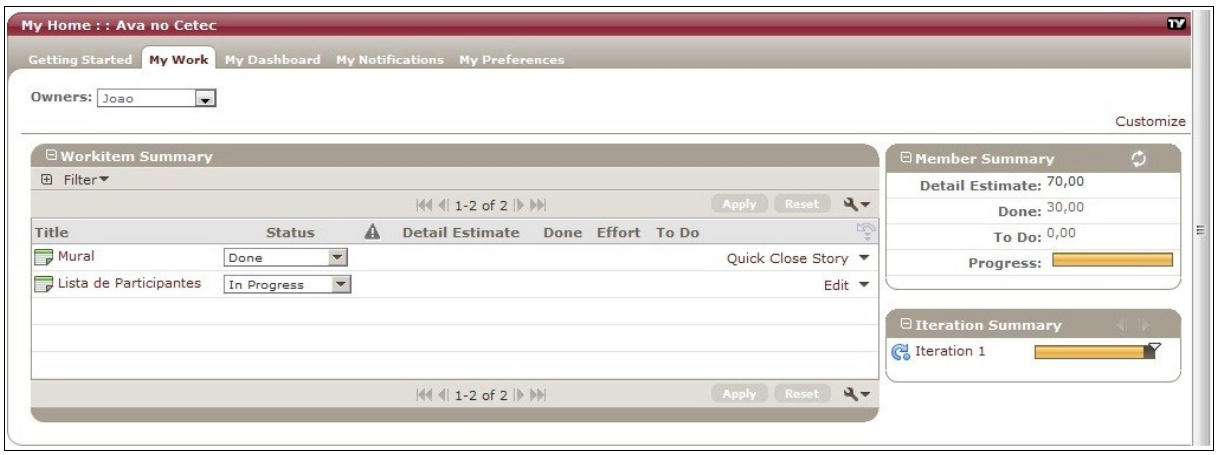


Figura 30: Captura de tela do painel de trabalho por recurso

Na figura 31 é apresentada uma das telas de consulta da iteração, chamada de "*dashboard*" onde apresenta os gráficos burndown da iteração e o acumulado de fluxo de trabalho por situação. Da metade da tela para baixo são exibidos os painéis por situação e com o recurso de "arrastar e soltar". Na figura está posicionado na aba "*Storyboard*" onde são exibidas as histórias por situação.

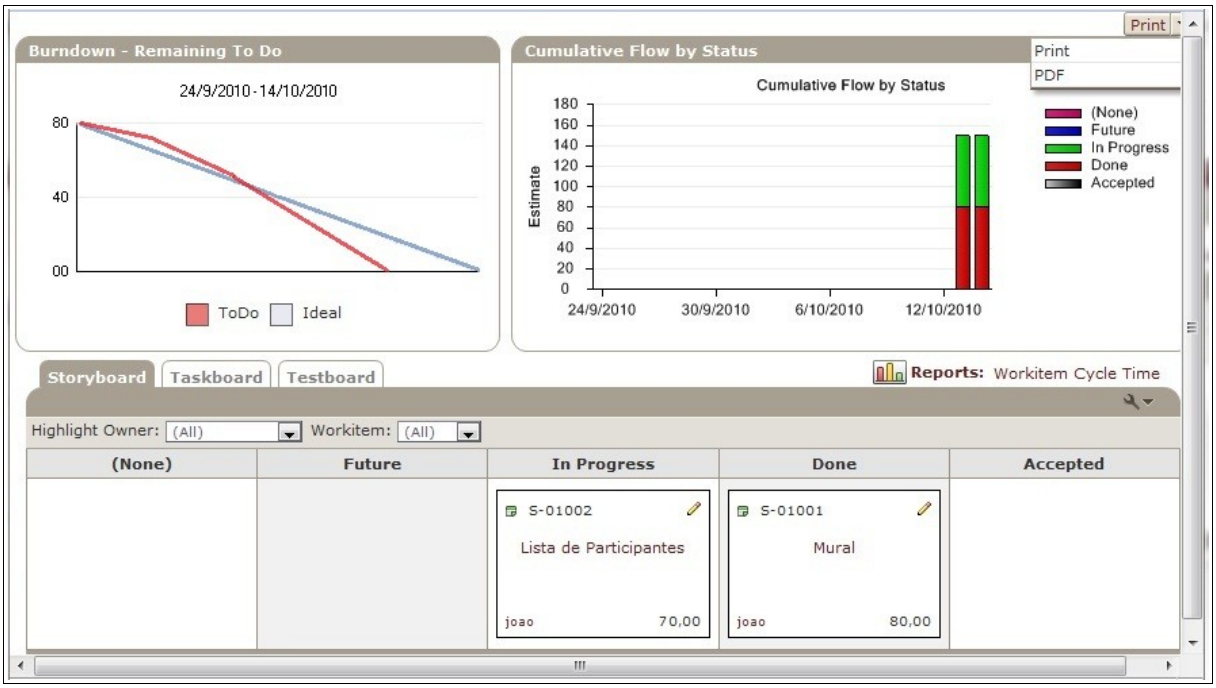


Figura 31: Captura de tela do *dashboard* da iteração

Na avaliação, a ferramenta VersionOne atendeu todos os requisitos funcionais com nota máxima. Nos não funcionais recebeu duas notas 0, uma nota 1 e oito notas 2. A pontuação final totalizou em 97, representando 88,2% sobre o total da pontuação possível.

h) Xplanner-Plus

Xplanner-Plus é uma ferramenta *open source* desenvolvida para apoiar projetos que utilizam metodologias ágeis, principalmente XP. É possível acessar uma versão de demonstração pelo endereço http://demo.xplannerplus.org/xplanner/do/logout. Xplanner-Plus é uma evolução da ferramenta Xplanner. Foram adicionadas algumas funcionalidades e melhorada a interface e navegabilidade dentro da ferramenta.

No site de Xplanner-Plus[19] está disponível a documentação para instalação. Não foi

19http://xplanner-plus.sourceforge.net/

localizado documentação de suporte, apenas fóruns de discussão. A versão que foi testada e avaliada foi 1.0b3 de 26/10/2009.

Apesar de não possuir documentação os testes ocorrem com sucesso já que a ferramenta é bastante simples e intuitiva. O mais alto nível são projetos e a ferramenta permite vários projetos. Cada projeto possui iterações e essas possuem histórias associadas. Para cada história são criadas tarefas, e as estimativas são realizadas em horas. A ferramenta não possui cadastro de equipes e as permissões são dadas a nível de projeto podendo ser Nenhum, Visualização, Edição, e Administração. Possui consultas e gráficos e poucas opções de relatórios, os quais apresentaram erros nos testes. A ferramenta não possui o conceito de *releases*, p*roduct backlog* nem quadro / painel de tarefas.

Na figura 32 é apresentado uma tela de consulta de história onde pode ser observado as tarefas associadas a essa história. Para acessar as tarefas basta clicar no nome da tarefa que possui um link que direciona para a tarefa em questão. Na coluna "*Action*" possui os ícones para as possíveis ações nas tarefas, que são: Editar, Excluir, Mover e Registrar horas trabalhadas.

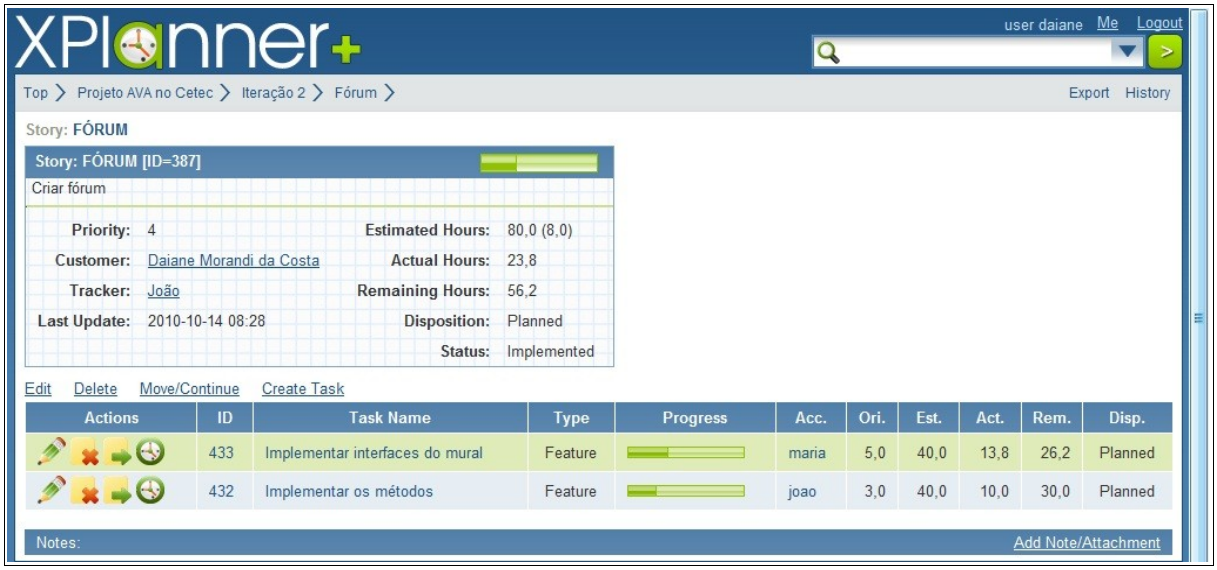


Figura 32: Captura de tela da consulta da história

Na avaliação realizada Xplanner-Plus atendeu oito dos dez requisitos funcionais e nove dos onze não funcionais. Ao todo pontuou quatro requisitos com nota 0, dois com nota 1 e quinze com nota 2. A pontuação final totalizou em 90, representando 81,8% sobre o total de pontos possíveis.

6.1.5 Conclusão da avaliação

Após concluída a execução das avaliações, as notas foram tabuladas em uma planilha eletrônica onde as pontuações foram consolidadas e comparadas para, então, realizar o julgamento, que é a etapa final do processo de avaliação. A planilha com a tabulação das notas segue na íntegra como anexo E deste trabalho. A pontuação máxima possível é de 110 pontos, caso todos os requisitos sejam totalmente atendidos.

Na tabela 17 são exibidas as pontuações de cada requisito e ferramenta. Na coluna "Requisitos" são descritos os requisitos com seus códigos e descrições. Na coluna "Peso" são exibidos os pesos de cada requisito. Na coluna "P" é exibida a pontuação do requisito em cada ferramenta, que é o resultado da multiplicação do peso pela nota. Na última linha da tabela é exibida a pontuação total de cada ferramenta, que é o resultado da soma das pontuações dos requisitos.

Tabela 17: Pontuações da avaliação

Requisitos		Peso	Agilefant	Agilo	Explainpmt	IceScrum	Rational TC	ScrumWorks	VersionOne	Xplanner
			P	P	P	P	P	P	P	P
Requisitos funcionais										
RF_01	Permitir controle por projetos	3	6	0	6	6	6	6	6	6
RF_02	Possuir iterações ou sprints	3	6	6	6	6	6	6	6	6
RF_03	Possuir cartão de histórias	3	6	3	6	6	6	3	6	6
RF_04	Possuir cartão de tarefas	3	6	3	3	3	3	3	6	6
RF_05	Possuir controle de horas trabalhadas por recurso	3	6	3	0	0	6	0	6	6
RF_06	Possuir gráficos de acompanhamento	2	4	2	2	4	4	4	4	4
RF_07	Possuir consultas de acompanhamento	3	6	6	3	6	6	6	6	6
RF_08	Possuir relatórios de acompanhamento	2	2	4	0	4	4	4	4	2
RF_09	Possuir cadastro de equipes (recursos)	1	2	2	2	0	1	1	2	0
RF_10	Possuir mural virtual de lembretes (planejamento de atividades)	2	4	4	0	2	4	0	4	0
Requisitos Não funcionais										
RNF_01	Ser um softw are livre (free)	3	6	6	6	6	0	0	0	6
RNF_02	Possuir versão w eb	3	6	6	6	6	6	6	6	6
RNF_03	Permitir que seja utilizado nos navegadores Firefox e Internet Explorer	3	6	6	6	6	6	6	6	6
RNF_04	Utilizar banco de dados free	3	6	6	6	6	6	6	6	6
RNF_05	Possuir versão com idioma português	2	0	0	0	0	4	0	0	0
RNF_06	Possuir controle de acesso por usuários no sistema	3	6	6	6	6	6	6	6	6
RNF_07	Possuir cadastro de permissões por projeto	2	0	0	2	2	4	0	4	4
RNF_08	Possuir interface amigável	3	6	3	3	3	6	6	6	6
RNF_09	Possuir facilidade na operação	3	6	0	3	3	3	6	3	6
RNF_10	Deve se comportar de maneira estável (ausência de erros)	3	6	3	6	6	6	6	6	0
RNF_11	Possuir mensagens compreensíveis	2	2	2	4	4	4	4	4	2
	Pontuação Total		**98**	**71**	**76**	**85**	**97**	**79**	**97**	**90**

A maior pontuação foi 98 da ferramenta Agilefant, depois com 97 pontos ficaram as ferramentas Rational Team Concert e VersionOne. Na sequência com 90 pontos ficou a Xplanner-Plus e 85 a IceScrum, após com 79 pontos ficou a ScrumWorks. E por fim, as ferramentas Explainpmt e Agilo com 76 e 71 pontos, respectivamente.

Na figura 33 é apresentado o gráfico com as notas de cada ferramenta avaliada. No eixo X são exibidas as pontuações e no eixo Y as ferramentas.

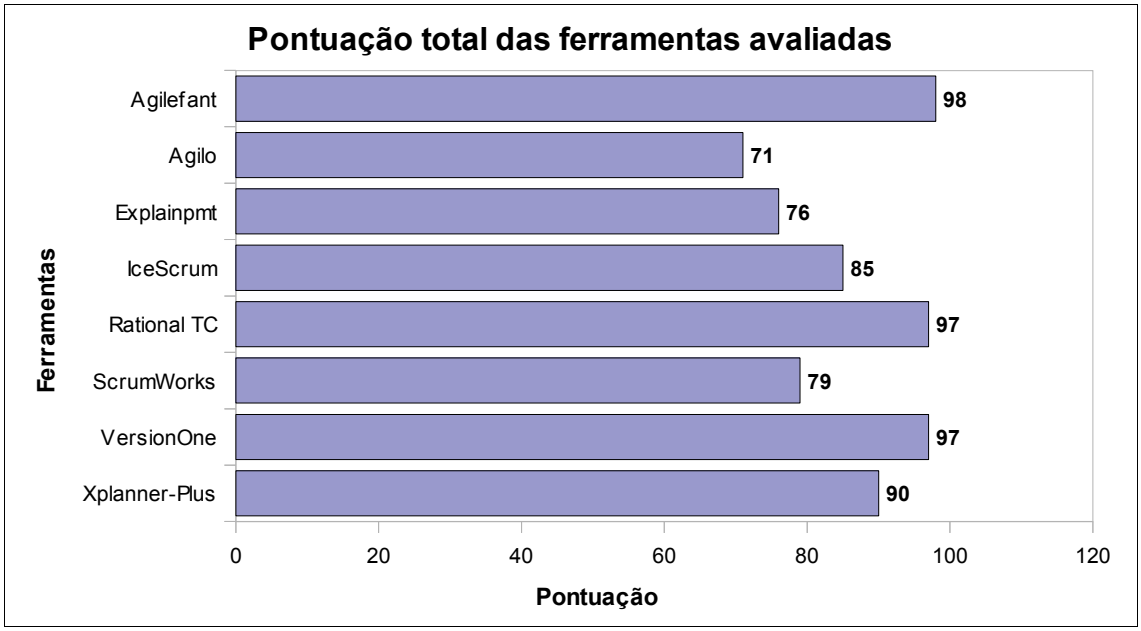


Figura 33: Pontuação total das ferramentas avaliadas

Foram tabuladas em separado somente as notas das ferramentas *open source*, conforme pode ser observado no gráfico exibido na figura 34. No eixo X estão descritas as pontuações e no eixo Y as ferramentas.

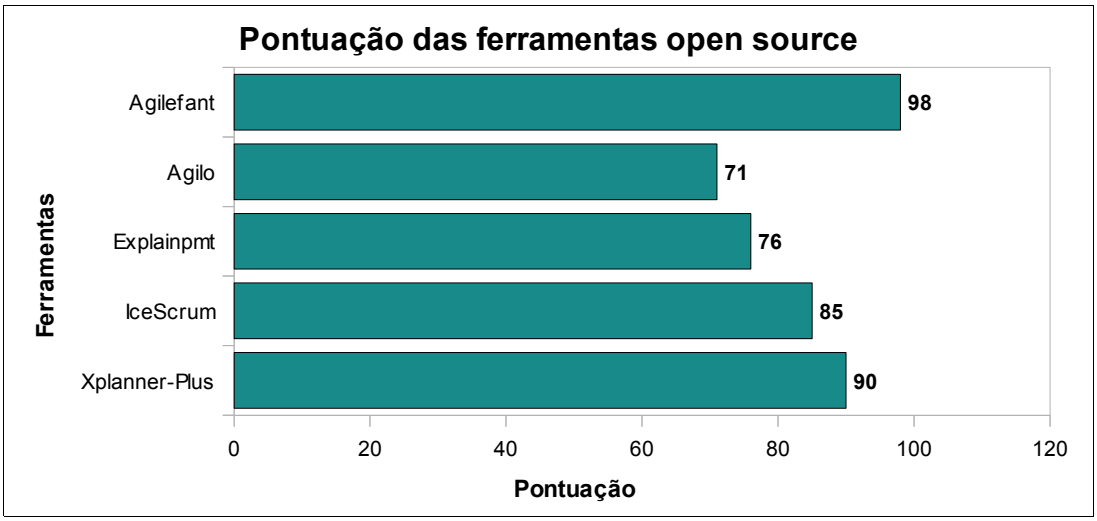


Figura 34: Pontuação total das ferramentas *open source*

O resultado das avaliações foi apresentado, através de uma reunião, para a equipe de desenvolvimento, objeto de estudo deste trabalho, onde definiu que deverá ser selecionada uma ferramenta *open source*, ou seja, sem custos.

Nessa reunião foram apresentadas as três ferramentas *open source* que obtiveram maior pontuação na avaliação e realizados breves testes para que a equipe pudesse emitir um parecer.

Foram apresentadas para a equipe as ferramentas Agilefant, Xplanner-Plus e IceScrum e analisadas as notas de cada requisito. A equipe entende que os requisitos funcionais são de grande importância para o processo customizado que foi proposto devido suas características ágeis de desenvolvimento.

Por esse motivo foi avaliado cada requisito funcional e não funcional em cada uma das três ferramentas *free* de maior pontuação. Na figura 35 é apresentado um gráfico com as pontuações de cada ferramenta em requisitos funcionais e não funcionais. No eixo X estão descritas as pontuações e no eixo Y as ferramentas.

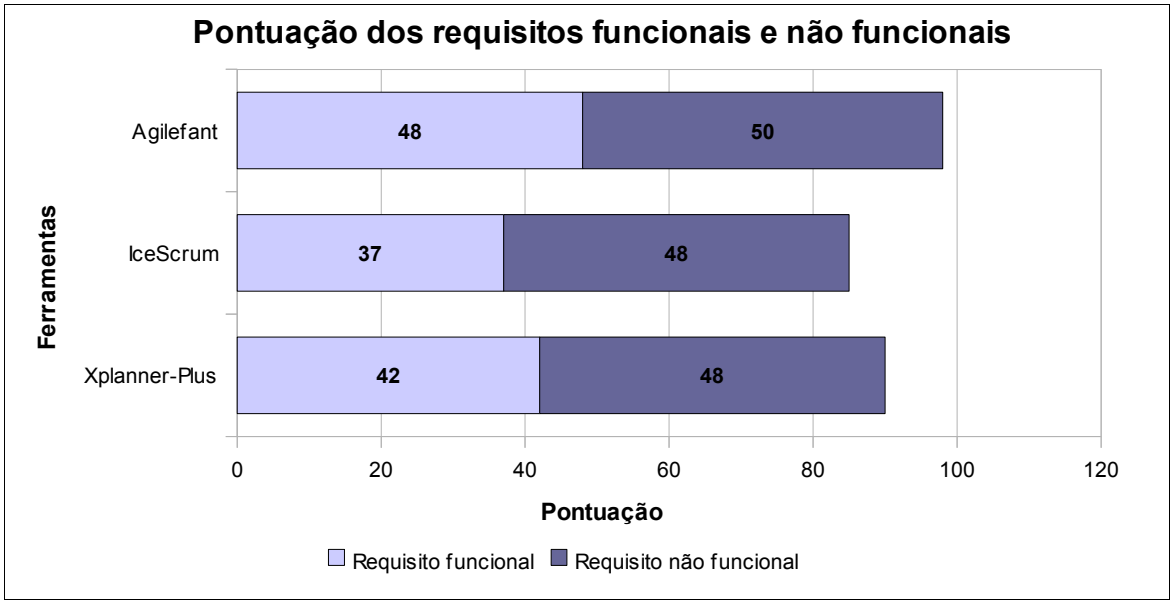


Figura 35: Pontuação dos requisitos funcionais e não funcionais

Como pode ser observado no gráfico acima, a ferramenta Agilefant possui a maior nota nos requisitos funcionais e não funcionais. O parecer da equipe de desenvolvimento também foi a favor de Agilefant, o que conduziu à conclusão da avaliação indicando a Agilefant como a ferramenta selecionada. A nota não foi absoluta na conclusão, o parecer da equipe também foi considerado nesta etapa de conclusão da avaliação de ferramentas.

6.2 Considerações finais

Este capítulo apresentou o processo de avaliação executado para a seleção de uma ferramenta para apoiar a metodologia de desenvolvimento proposta neste trabalho.

Foram estudadas as séries de normas NBR ISO/IEC 9126 e NBR ISO/IEC 14598 que proporcionaram o conhecimento para a definição de um modelo de qualidade e o processo de avaliação de *software* que foi aplicado. O resumo do estudo das normas segue como anexo D deste trabalho, na página 148.

O processo de avaliação executado foi baseado na norma NBR ISO/IEC 14598 e na seção 6.2 foram apresentados os resultados de cada etapa do processo de avaliação. As etapas executadas foram Estabelecer requisitos de avaliação, Especificar a avaliação, Projetar a avaliação e Executar a avaliação.

Foram avaliadas oito ferramentas, onde cinco são *open source* e três comerciais. As ferramentas avaliadas foram Agilefant, Agilo, Explainpmt, IceScrum, Rational Team Concert, ScrumWorks, VersionOne e Xplanner-Plus. As três ferramentas com maior pontuação foram Agilefant, Rational Team Concert e VersionOne. Como o presente trabalho trata de customização de processo de desenvolvimento foram consideradas as políticas da instituição e o parecer da equipe que está sendo objeto de estudo deste trabalho. Por esse motivo as ferramentas comerciais foram descartadas, já que não seriam viáveis para a equipe em questão.

Dessa forma, foram apresentadas para a equipe as três ferramentas gratuitas (*free)* com maior pontuação na avaliação, Agilefant, Xplanner-Plus, IceScrum, que foram submetidas à uma breve apreciação da equipe que por fim emitiu um parecer a favor da ferramenta Agilefant.

Considerando o resultado da avaliação e o parecer da equipe de desenvolvimento, a avaliação foi concluída selecionando a ferramenta Agilefant para apoiar a metodologia de desenvolvimento proposta neste trabalho.

No próximo capítulo será apresentado um estudo de caso, no qual foi aplicado a metodologia de desenvolvimento proposta neste trabalho.

7 . ESTUDO DE CASO

A metodologia de desenvolvimento de *software* proposta neste trabalho pode ser avaliada através da aplicação de um estudo de caso. Segundo Wohlin *et. al.* (2000) um estudo de caso é conduzido para investigar uma única entidade de um fenômeno com um espaço de tempo específico. Permite coletar evidências para comprovar a validade da hipótese (WAZLAWICK, 2009).

Este capítulo descreve o estudo de caso realizado neste trabalho e está divido em 3 seções, onde na seção 7.1 é apresentada a caracterização do estudo de caso, a seção 7.2 descreve a aplicação da metodologia proposta neste trabalho e por fim, a seção 7.3 apresenta os resultados obtidos e as considerações finais do capítulo.

No estudo de caso foram aplicadas as fases "introduzir mudança de processo", "treinar equipe" e "ajustar mudanças de processo" do estágio de mudança conforme o ciclo de aprimoramento de processo sugerido por Sommerville (2007). Os resultados foram avaliados através das métricas definidas na fase de medição do processo através da abordagem GQM (*Goal-Question-Metric*).

Na fase "treinar equipe" aconteceu o treinamento da equipe de desenvolvimento que participa do processo que foi customizado. A equipe, formada por um analista de sistemas e um programador, utilizou a metodologia proposta em uma demanda de manutenção evolutiva de *software* por um período de quatro semanas. Nesse treinamento foram apresentados o novo processo, os artefatos e a ferramenta Agilefant.

Na fase "introduzir mudança de processo" o novo processo foi colocado em prática. Foi nessa fase que o estudo de caso foi aplicado.

E por fim, na fase de "ajuste de mudanças" foi realizado o acompanhamento das mudanças introduzidas. Sommerville (2007) sugere que essa fase dure por alguns meses para que ocorram as modificações necessárias e a equipe esteja satisfeita com o novo processo. Como não houve tempo hábil para realizar o acompanhamento durante meses, essa fase não foi concluída.

7.1 Caracterização do estudo de caso

A aplicação do estudo de caso ocorreu com uma equipe de desenvolvimento de

software que trabalha na Universidade de Caxias do Sul (UCS). A equipe apresenta as características definidas no objetivo deste trabalho, descrito no capítulo 1: pequena, composta por um analista de sistemas e um desenvolvedor que realizam manutenção evolutiva em alguns *softwares web* da UCS.

O *software* que a equipe realiza manutenção é o Ambiente Virtual de Aprendizagem (AVA) utilizado pelos alunos e professores da graduação e pós graduação da UCS. Utilizada para fins acadêmicos como apoio ao ensino, a ferramenta possui diversas funcionalidades que permitem a interação entre os professores e alunos como a troca de mensagens, mural de recados, lista de participantes, informações da disciplina, cronograma da disciplina, acervo da turma, *webfólio*, fórum, entre outros.

O estudo de caso foi aplicado em uma demanda de manutenção evolutiva do sistema AVA, onde a solicitação partiu da Coordenadoria Administrativa da Escola de Ensino Médio e Profissionalizante (CETEC). A solicitação foi de disponibilizar o AVA para os alunos e professores do Cetec de acordo com as suas características de ensino, que em algumas situações são diferentes da graduação e pós graduação. Por exemplo, o conceito de turma, que para o Cetec é anual e para a graduação é semestral.

Dentro dessas características foi aplicado o estudo de caso, o qual os resultados são descritos na próxima seção.

7.2 Aplicação da metodologia proposta

Nesta seção são apresentados os resultados obtidos na aplicação do estudo de caso. Os resultados serão apresentados de acordo com a execução do processo proposto, apresentado no capítulo 5, que compreende três fases: planejamento, desenvolvimento e encerramento.

O projeto foi denominado de “Ava Cetec” e a duração das iterações foi definida em 2 (duas) semanas por iteração. A equipe cliente é representada pela professora coordenadora administrativa do Cetec, e a equipe de desenvolvimento é comporta por um analista de sistemas e um programador. A carga horária semanal dedicada ao projeto pela equipe de desenvolvimento foi de 25 horas por recurso, considerando uma média de 5 horas diárias, já que a equipe realiza outras atividades de suporte à ferramenta .

7.2.1 Fase de planejamento

A fase de planejamento é composta pelas atividades “Registrar solicitação”, “Realizar reunião inicial”, “Estimar tempo para funcionalidades”, “Escrever histórias” e “Planejar iterações”.

a) Registrar solicitação

A fase de planejamento se iniciou com a solicitação de manutenção reportada pela professora coordenadora administrativa do Cetec em final de setembro/2010 através de uma ligação telefônica. A partir desta solicitação, foi agendada uma reunião inicial com a equipe cliente e a equipe de desenvolvimento.

b) Realizar reunião inicial

Na reunião inicial, a equipe cliente apresentou as suas expectativas e as funcionalidades do Ava que solicitam para o Cetec. Para cada funcionalidade foi definida uma prioridade conforme a necessidade do cliente. Nessa atividade foi gerado o artefato “Lista de Funcionalidades” que apresenta as funcionalidades priorizadas pelo cliente e os responsáveis. A priorização das funcionalidade se classificam em Alta, Média e Baixa. Onde o que foi definido pelo cliente como prioridade Alta é o que o cliente entende que deva ser entregue primeiro. O responsável por todas as funcionalidades é o analista de sistemas da equipe de desenvolvimento.

O artefato é apresentado na tabela 18, onde a coluna “Tempo estimado (hs)” está vazia, pois a estimativa de tempo foi realizada pela equipe de desenvolvimento em uma atividade posterior a reunião inicial.

Tabela 18: Lista de Funcionalidades do projeto Ava Cetec

Lista de Funcionalidades				
Id	**Descrição**	**Prioridade cliente**	**Tempo estimado (hs)**	**Responsável**
1	Lista de Participantes	Alta		Fabiano
2	Mural de recados	Alta		Fabiano
3	Fórum	Média		Fabiano
4	Mensagens	Média		Fabiano
5	Webfólio	Média		Fabiano
6	Histórico de acessos	Baixa		Fabiano

Ainda na reunião inicial a equipe de desenvolvimento apresentou o processo de desenvolvimento para a equipe cliente, para que todos tivessem conhecimento das atividades que compõem o processo bem como o papel e a responsabilidade de cada um. Encerrada a reunião inicial, a próxima atividade foi uma reunião somente com a equipe de desenvolvimento para realizar uma estiva de tempo das funcionalidades solicitadas pelo cliente.

c) Estimar tempo para as funcionalidades

O analista de sistemas e o programador avaliaram juntos as funcionalidades solicitadas pelo cliente e estimaram tempo (em horas) para cada uma. A estimativa foi empírica, baseada na experiência da equipe de desenvolvimento.

Durante essa atividade identificaram que no artefato Lista de Funcionalidades estava descrito somente as funcionalidades descritas pelo cliente, ou aquelas que tem valor para o mesmo. Porém existem algumas atividades técnicas relacionadas à estrutura da aplicação e banco de dados que devem ser consideradas na estimativa de tempo do projeto, que não estão relacionadas a cada uma das funcionalidades em si, mas ao projeto como um todo.

Portanto foi incluída uma nova funcionalidade no artefato Lista de Funcionalidades chamada "Disponibilizar novo ambiente Ava para Cetec" que compreende as tarefas de preparação da estrutura da aplicação e banco de dados para o novo ambiente que deverá ser criado dentro do AVA para o Cetec. Na tabela 19 é apresentado o artefato Lista de Funcionalidades com as informações atualizadas pela equipe de desenvolvimento

Tabela 19: Artefato Lista de Funcionalidades atualizado

Lista de Funcionalidades				
Id	**Descrição**	**Prioridade cliente**	**Tempo estimado (hs)**	**Responsável**
1	Disponibilizar novo ambiente Ava para Cetec	Alta	54	Fabiano
2	Lista de Participantes	Alta	8	Fabiano
3	Mural de recados	Alta	8	Fabiano
4	Fórum	Média	8	Fabiano
5	Mensagens	Média	12	Fabiano
6	Webfólio	Média	24	Fabiano
7	Histórico de acessos	Baixa	8	Fabiano

d) Escrever histórias

Dando continuidade ao processo na fase de planejamento, a próxima atividade

executada foi “Escrever histórias”. Para cada funcionalidade foi criada uma história com uma breve descrição, responsável e horas estimadas.

As histórias foram registradas na ferramenta Agilefant pelo analista de sistemas, e não pelo cliente como foi proposto no processo. Essa foi uma decisão da equipe de desenvolvimento por entender que, como trata-se da disponibilização das funcionalidades já existentes do Ava não implica em mudanças nessas funcionalidades. Ou seja, o projeto não contempla mudanças de comportamento das funcionalidades atuais do Ava, e sim a liberação das mesmas em um novo ambiente criado para o Cetec.

Como a ferramenta controla os pontos por história, foi determinado o seguinte padrão: Para funcionalidades de prioridade “Alta” utilizou-se de 3 pontos, para prioridade “Média” 2 pontos e prioridade “Baixa” 1 ponto.

As histórias foram criadas sem informar a iteração, já que essa informação foi definida na próxima atividade do processo, “Planejar iterações”. Após a definição do Plano de Iterações do projeto, as histórias foram atualizadas adicionado as iterações correspondentes.

Na figura 36 é apresentada a consulta da história “Lista de Participantes”.

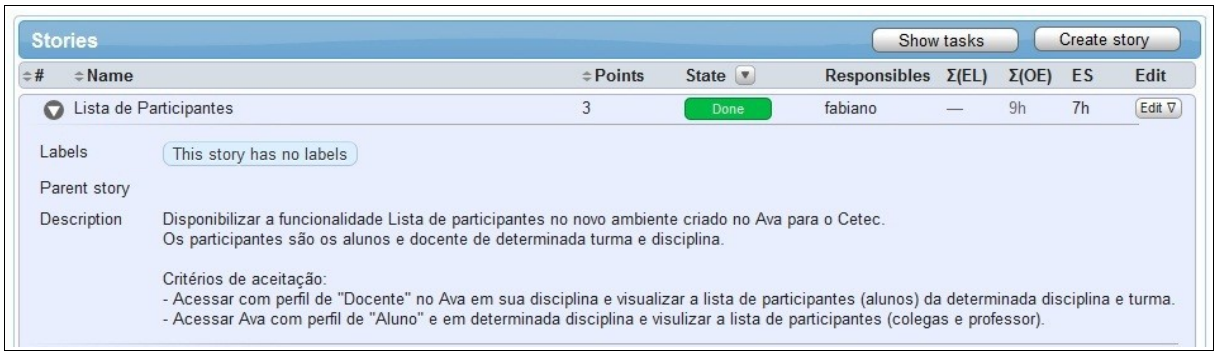


Figura 36: Captura de tela da consulta de uma história

e) Planejar Iterações

Após a equipe de desenvolvimento concluir as estimativas de tempo e o cadastro das histórias foi agendada um nova reunião com a equipe cliente para essas estimativas serem apresentadas.

Nessa reunião o grupo avaliou as estimativas de tempo e definiu a quantidade de iterações e quais funcionalidades seriam entregues em cada iteração. Essa definição foi baseada na avaliação das estimativas de horas das funcionalidades *versus* carga horária semanal da equipe e a duração da iteração, que é de duas semanas.

O resultado dessa reunião foi o plano de iterações, que é apresentado na tabela 20.

Tabela 20: Plano de Iterações do projeto Ava Cetec

Plano de Iterações				
Iteração	Data início	Data fim	Funcionalidades	
			Id	Descrição
1	11/10/10	22/10/10	1	Disponibilizar novo ambiente Ava para Cetec
			2	Lista de Participantes
2	25/10/10	05/11/10	3	Mural de recados
			4	Fórum
			5	Mensagens
			6	Webfólio
			7	Histórico de acessos

Após definido o plano de iterações do projeto, as informações foram registradas na ferramenta Agilefant. Na figura 37 é exibida a consulta do projeto e das iterações na ferramenta Agilefant.

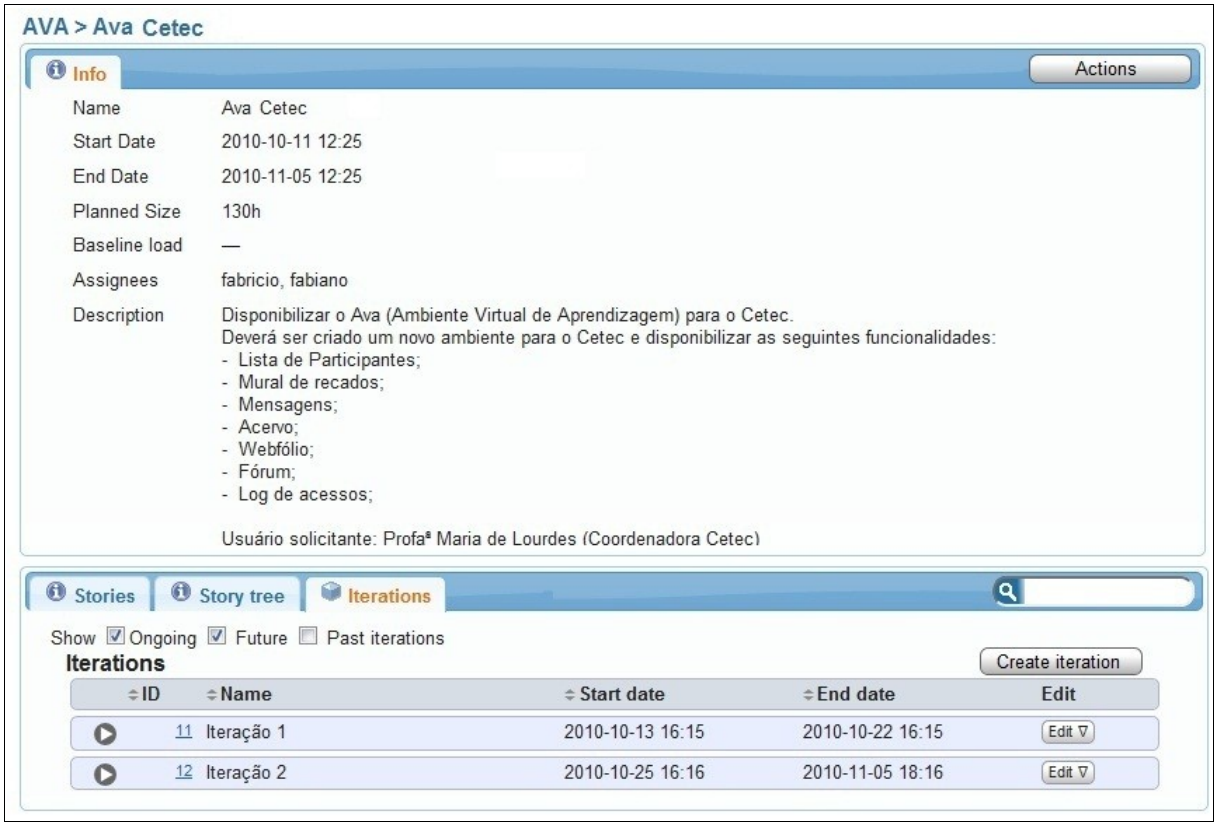


Figura 37: Consulta do projeto Ava Cetec e suas iterações

Com o Plano de Iterações definido, foram atualizadas as histórias incluindo a informação da iteração a qual pertencem. Na figura 38 é apresentada a listagem das histórias cadastradas para cada iteração do projeto Ava Cetec na ferramenta Agilefant.

Figura 38: Histórias cadastradas para o projeto

A fase de planejamento foi encerrada com a criação do Plano de Iterações. Os artefatos gerados nessa fase serão utilizados na fase de desenvolvimento para a execução das atividades.

7.2.2 Fase de desenvolvimento

Foi nessa fase que aconteceu o maior trabalho de desenvolvimento compreendendo a análise, codificação e testes. A fase de desenvolvimento é cíclica, ou seja, ocorre por iteração e compreende as atividades "Registrar atividades", "Realizar modelagem", "Codificar", "Realizar testes unitários", "Realizar testes de sistema" e "Apresentar para cliente".

a) Registrar atividades

A fase de desenvolvimento se iniciou pela atividade "Registrar atividades", que correspondeu ao registro de tarefas para cada história na ferramenta Agilefant.

Uma história pode possuir várias tarefas, onde a soma das horas estimadas e realizadas das tarefas refletem nas horas da história. As possíveis situações de uma tarefa e história são: Não iniciada (*Not Started*), Iniciada (*Started*), Pendente (*Pending*), Bloqueada (*Blocked*), Pronto (*Ready*) e Concluído (*Done*). O padrão adotado pela equipe foi de utilizar as situações Não iniciada, Iniciada, Pendente, Pronto e Concluído com os seguintes critérios:

- **Não iniciada:** utilizada para as histórias e tarefas novas e ainda não iniciadas.
- **Iniciada:** utilizada quando o analista ou programador iniciar a história/tarefa.
- **Pendente:** utilizada quando uma história/tarefa já iniciada estiver pendente com alguma coisa, por exemplo: uma definição por parte do cliente.

- **Pronto:** utilizada quando uma história/tarefa estiver com o desenvolvimento e testes concluídos, aguardando a aprovação do cliente.
- **Concluído:** utilizada quando o cliente aprovou.

Na figura 39 é apresentada uma história com suas tarefas registradas na ferramenta Agilefant.

Figura 39: Consulta das tarefas de uma história

b) Realizar modelagem

Dando continuidade no processo, fase de desenvolvimento, a próxima atividade executada foi "Realizar modelagem". Como proposto neste trabalho, esta atividade é facultativa pois depende da complexidade dos requisitos e da necessidade do analista de sistemas. Para esse projeto o artefato gerado foi o Diagrama ER (Entidade-Relacionamento), o qual refere-se as alterações no banco de dados. Por questões de sigilo e solicitação da equipe, este artefato não será apresentado neste trabalho.

c) Codificar

Após a modelagem, iniciou-se a atividade "Codificar", que corresponde a confecção dos códigos fonte dos programas. As horas de trabalho das atividades de codificação foram registradas na ferramenta Agilefant nas tarefas registradas para tal. Na ferramenta as horas de trabalho são chamadas de "*effort spent*" , em português "esforço despendido" e devem ser registradas por dia. O registro pode ser retroativo, ou seja, registrar horas de trabalho para dias anteriores.

Nas figuras 40 e 41 são exibidas as telas de registro de horas trabalhadas. Basta acessar o botão [*Edit*] da tarefa desejada e selecionar a opção "*Log effort*", conforme mostra a figura 40.

Figura 40: Acessando a tarefa para registrar as horas trabalhadas

O registro de horas trabalhadas é realizado informando a quantidade de horas, a data e o comentário desejado, e salva clicando no botão [OK], conforme mostra a figura 41. O usuário o sistema busca do usuário logado no momento da ação.

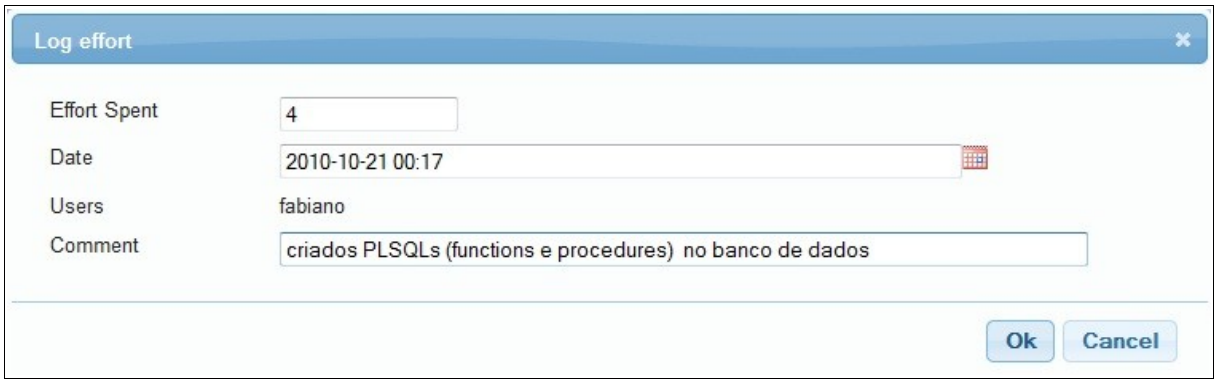


Figura 41: Registro de horas trabalhadas

d) Realizar testes unitários

Os testes unitários não foram automatizados (codificados) por opção da equipe de desenvolvimento. Mas durante todo o ciclo de desenvolvimento os testes unitários foram executados de forma manual pelo desenvolvedor.

Quando identificados erros ou desvios de comportamento, o programador corrigiu os programas, submetendo o novo código novamente ao teste unitário. Portanto, essa atividade foi cíclica, só seguindo adiante no processo quando o resultado dos testes unitários foi positivo.

e) Realizar testes de sistema

Os testes de sistema foram executados em todas as histórias entregues. Na ferramenta Agilenfat foi registrada uma tarefa chamada “Testes” para cada história. Nessa tarefa foram registrados os testes de sistema, que foram guiados pelos critérios de aceitação descritos na história. Quando detectado algum erro foi registrado na tarefa e atualizado a situação para “Pendente” (*Pending*) até que o programa fosse corrigido e testado novamente. Quando o teste foi executado com sucesso a situação da tarefa foi atualizada para “Pronto” (*Ready*).

Na figura 42 é exibida uma tela de consulta de uma tarefa de testes da história “Lista de Participantes”, onde é possível observar o histórico das horas trabalhadas com os comentários do analista de sistemas.

Figura 42: Consulta do histórico da tarefa "Testes"

As atividades de codificação e testes se repetiram para cada história das Iterações 1 e 2, e diariamente a equipe se reunia para discutir brevemente sobre o andamento do trabalho. As reuniões diárias sugeridas no processo proposto aconteceram mas de forma bem informal e sem horários definidos. Geralmente no turno da manhã, mas sempre que necessário o analista e o programador conversavam já que estão sentados um ao lado do outro.

O acompanhamento do trabalho foi realizado através de consultas pela ferramenta Agilefant. Durante a fase de desenvolvimento a equipe não sentiu necessidade de relatórios impressos.

Na figura 43 é apresentada uma tela de consulta do acompanhamento da Iteração 1, antes de seu término. Essa é uma visão geral da Iteração, onde é possível acessar outras consultas do sistema por histórias ou tarefas com maiores detalhes.

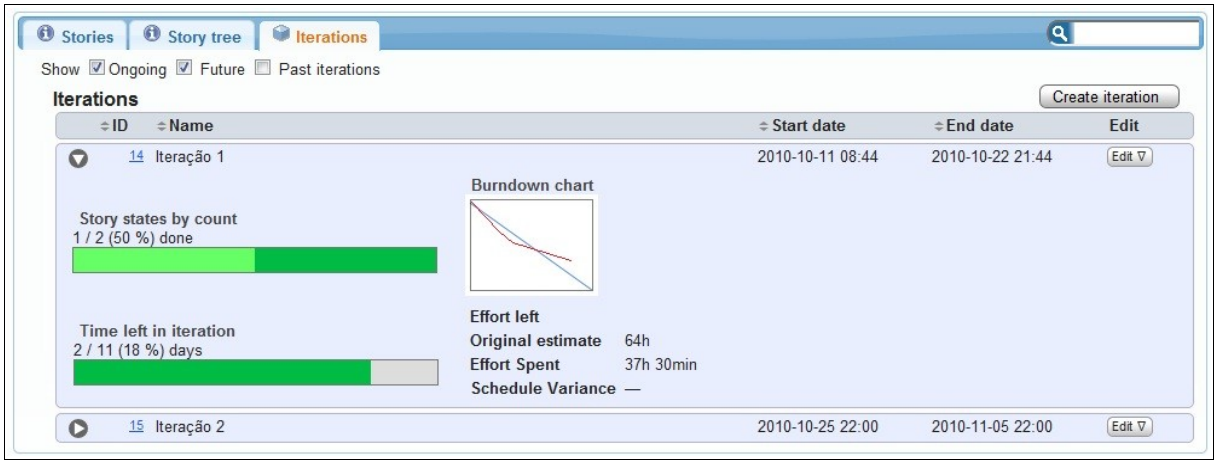


Figura 43: Consulta de acompanhamento da Iteração 1

f) Apresentar para cliente

Ao finalizar as histórias da Iteração 1, foi agendada uma reunião com a equipe cliente, para apresentar o que foi produzido nas duas semanas de trabalho da primeira iteração. Essa reunião foi bem informal, no próprio local de trabalho do usuário chave e quem conduziu foi o analista de sistemas que realizou uma apresentação das funcionalidades concluídas na Iteração 1. As funcionalidades apresentadas foram homologadas e aceitas pela equipe cliente, representada pela Profª. Coordenadora Administrativa do Cetec. Com o aceite do cliente se encerrou a fase de desenvolvimento da Iteração 1 e iniciou-se a fase de encerramento, que está descrita na próxima subseção, 7.2.2 (página 140).

Por se tratar de um processo iterativo e incremental, a fase de desenvolvimento é cíclica enquanto houverem iterações para entregar. Desse modo, dando continuidade no processo de desenvolvimento, foi reiniciada a fase de desenvolvimento para a Iteração 2, que compreendeu a entrega de cinco histórias. Na figura 44 é apresentada a relação das histórias entregues na Iteração 2, que são: Mural de Recados, Fórum, Mensagens, *Webfólio*, Histórico de acessos.

Figura 44: Listagem das histórias da Iteração 2

Ao serem finalizadas as histórias da Iteração 2, assim como na Iteração 1, foi agendada uma reunião com equipe cliente para realizar uma demonstração das histórias concluídas na Iteração 2.

A reunião aconteceu novamente no local de trabalho da Profª. Coordenadora Administrativa do Cetec onde o analista de sistemas conduziu a apresentação em ambiente de desenvolvimento (base de testes). A entrega foi homologada e aceita pelo cliente, desse forma encerrando a fase de desenvolvimento da Iteração 2 e iniciando novamente a fase de encerramento, descrita na subseção 7.2.2 (página 140).

Serão exibidas algumas figuras com consultas realizadas na ferramenta durante a fase de desenvolvimento.

Na figura 45 é apresentado o gráfico burndown da iteração 2 consultado em 28/10/10 pelo analista de sistemas.

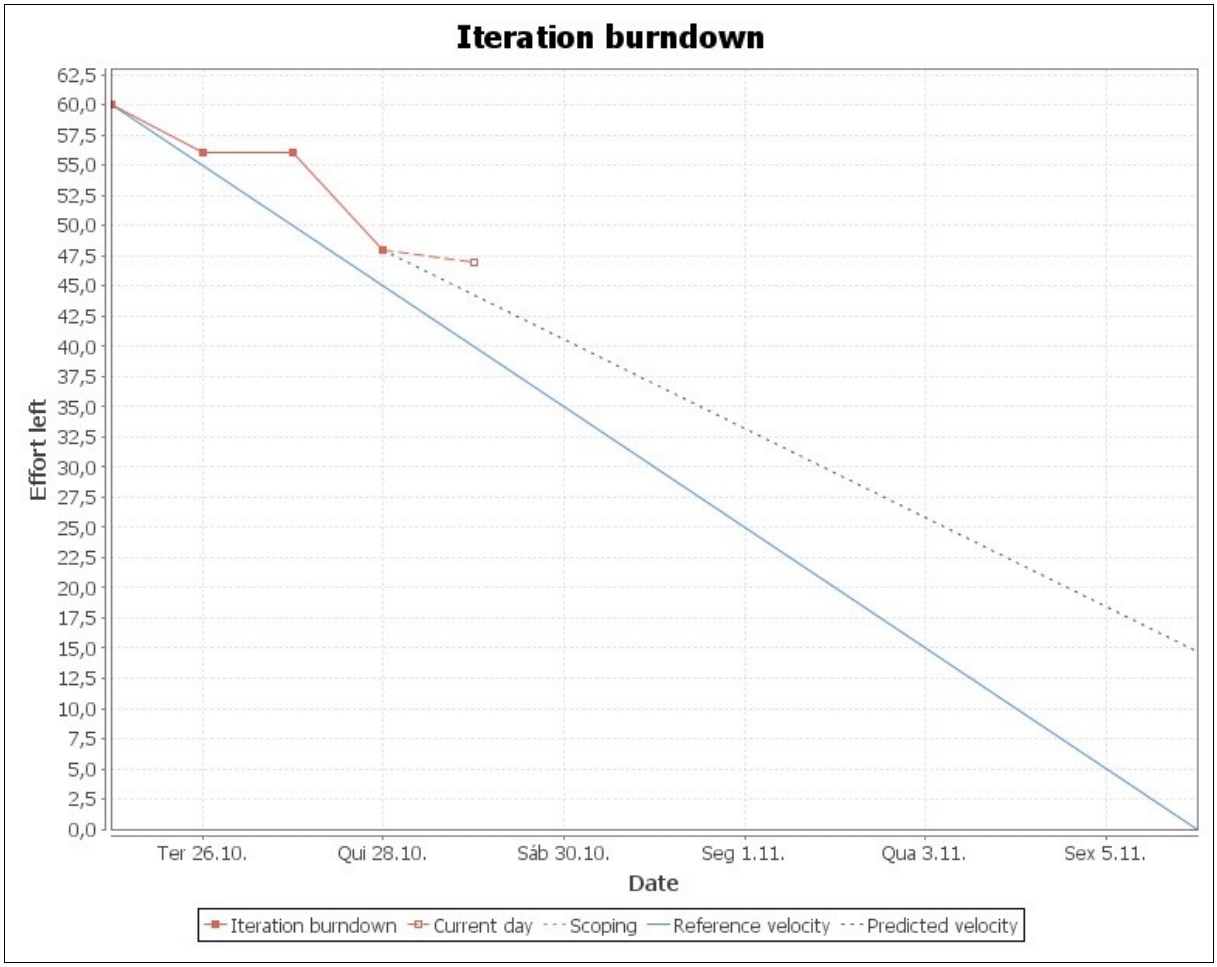


Figura 45: Gráfico *burndown* da iteração 2

Na figura 46 pode ser observado a tela de filtro para geração de relatórios.

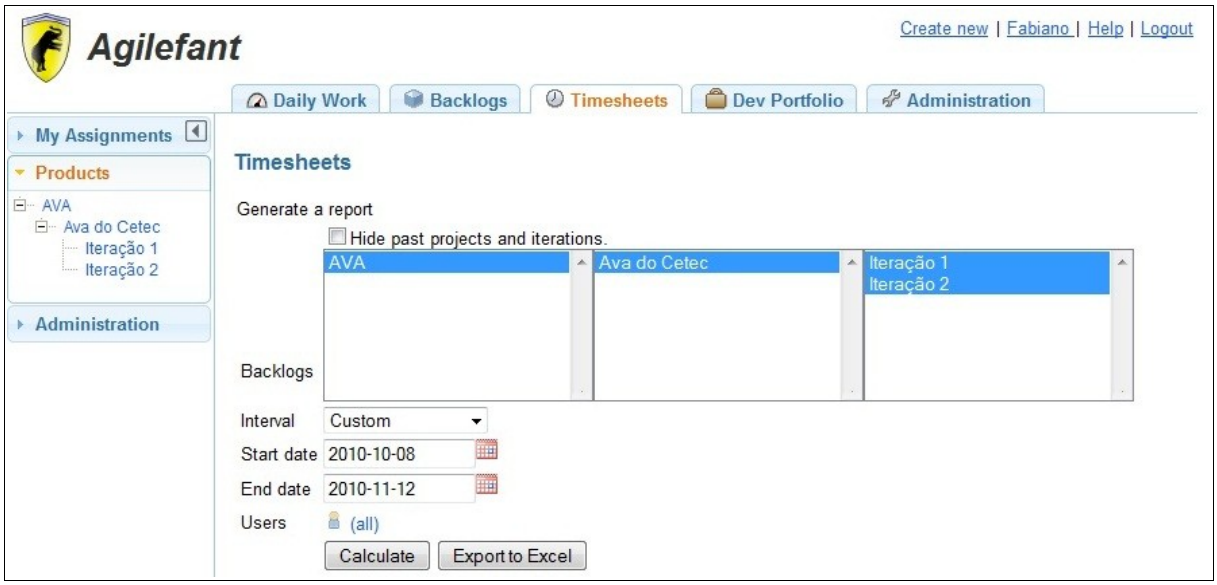


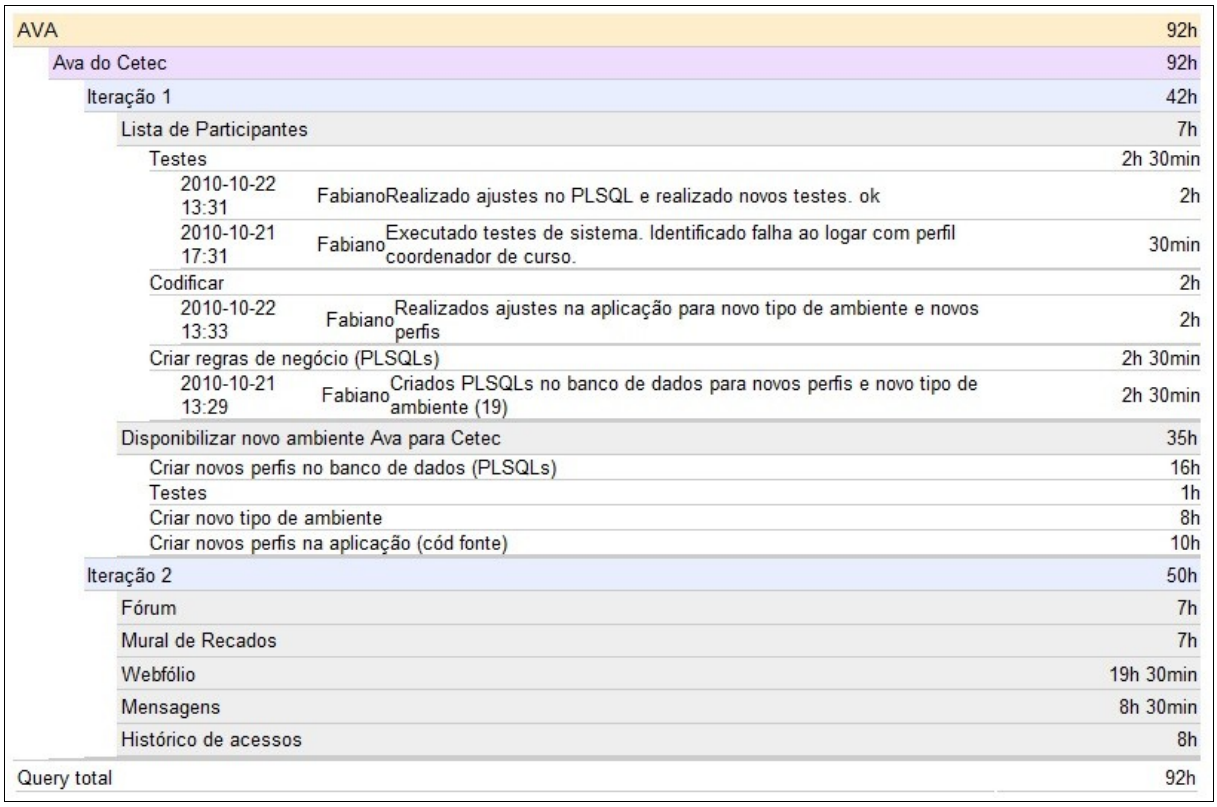
Figura 46: Tela de filtro para geração de relatórios

Na figura 47 é demonstrado o relatório gerado a partir do filtro apresentado na figura anterior, 46.

AVA	92h
Ava do Cetec	92h
Iteração 1	42h
Lista de Participantes	7h
Testes	2h 30min
Codificar	2h
Criar regras de negócio (PLSQLs)	2h 30min
Disponibilizar novo ambiente Ava para Cetec	35h
Criar novos perfis no banco de dados (PLSQLs)	16h
Testes	1h
Criar novo tipo de ambiente	8h
Criar novos perfis na aplicação (cód fonte)	10h
Iteração 2	50h
Fórum	7h
Mural de Recados	7h
Webfólio	19h 30min
Mensagens	8h 30min
Histórico de acessos	8h
Query total	92h

Figura 47: Relatório de horas trabalhadas do projeto Ava Cetec

O relatório pode ser expandido e/ou recolhido apenas clicando em cima da linha desejada. Na figura 48 é apresentado o mesmo relatório da figura 47, porém analítico.



AVA			92h
Ava do Cetec			92h
Iteração 1			42h
Lista de Participantes			7h
Testes			2h 30min
2010-10-22 13:31	Fabiano	Realizado ajustes no PLSQL e realizado novos testes. ok	2h
2010-10-21 17:31	Fabiano	Executado testes de sistema. Identificado falha ao logar com perfil coordenador de curso.	30min
Codificar			2h
2010-10-22 13:33	Fabiano	Realizados ajustes na aplicação para novo tipo de ambiente e novos perfis	2h
Criar regras de negócio (PLSQLs)			2h 30min
2010-10-21 13:29	Fabiano	Criados PLSQLs no banco de dados para novos perfis e novo tipo de ambiente (19)	2h 30min
Disponibilizar novo ambiente Ava para Cetec			35h
Criar novos perfis no banco de dados (PLSQLs)			16h
Testes			1h
Criar novo tipo de ambiente			8h
Criar novos perfis na aplicação (cód fonte)			10h
Iteração 2			50h
Fórum			7h
Mural de Recados			7h
Webfólio			19h 30min
Mensagens			8h 30min
Histórico de acessos			8h
Query total			92h

Figura 48: Relatório analítico de horas trabalhadas

7.2.3 Fase de encerramento

A fase de encerramento, assim como a de desenvolvimento é cíclica, ou seja, ocorre por iteração. É composta pelas atividades "Entregar iteração", "Utilizar sistema e dar feedback", "Iniciar próxima iteração" e "Encerrar projeto".

a) Entregar iteração

A fase de encerramento iniciou-se com a liberação das funcionalidades entregues na Iteração 1 em ambiente de produção após o aceite da equipe cliente realizado ainda na fase de desenvolvimento. O analista de sistemas liberou acesso ao novo tipo de ambiente do Ava criado para o Cetec em base de produção para a Profª. Coordenadora Administrativa do Cetec, que ficou responsável por utilizar a ferramenta e dar retorno para a equipe de desenvolvimento do Ava.

O analista de sistemas enviou para a equipe cliente um documento por *e-mail* formalizando a entrega da iteração. Nesse documento foi informada a data de liberação e os

dados de acesso na ferramenta e demais orientações gerais.

Na figura 49 é exibido um recorte do artefato "Comunicado ao cliente" enviado para a equipe cliente formalizando a entrega da iteração 1.

Informamos que a partir de **22/10/10** estão disponíveis para uso as funcionalidades da Iteração 1 do projeto Projeto Ava Cetec.

Para acessar a ferramenta Ava no ambiente do Cetec é necessário acessar o UCSVirtual com seu login e senha pessoal.

As funcionalidades entregues na Iteração 1 são:

- Disponibilizar novo ambiente Ava para Cetec
- Lista de Participantes

Solicitamos que nos mantenham informados sobre a utilização da ferramenta, bem como em situações de erros ou dúvidas.

Atenciosamente,
Equipe de desenvolvimento Ava

Figura 49: Artefato de comunicação ao cliente

Ao final de cada iteração a equipe de desenvolvimento se reuniu brevemente para avaliar como foi o andamento da iteração entregue sob os pontos de vista de processo, ferramentas e pessoas.

b) Utilizar sistema e dar *feedback*

Após a entrega da iteração 1 a usuária chave utilizou a ferramenta Ava no ambiente do Cetec por uma semana e reportou ao analista de sistemas suas dúvidas e sugestões de melhoria.

Ao concluir essa atividade na Iteração 1, a próxima atividade executada foi "Iniciar próxima iteração" pois ainda existia a iteração 2 para entregar. Quando essa atividade foi executada na iteração 2, a próxima atividade executada foi "Encerrar projeto".

c) Iniciar próxima iteração

Após a entrega da iteração 1 a equipe de desenvolvimento iniciou a fase de de desenvolvimento para a Iteração 2, que foi encerrada mediante o aceite da equipe cliente das funcionalidades apresentadas.

Assim como ocorreu com a Iteração 1, após o aceite do cliente iniciou-se a fase de encerramento para a Iteração 2, onde todas as funcionalidades foram liberadas em ambiente de produção estando disponível para uso. Por não haver mais iterações, após a atividade

"Utilizar sistema e dar feedback" a próxima e última atividade executada foi a "encerrar projeto".

d) Encerrar projeto

Após entregues todas as iterações do projeto, a ferramenta Ava estava disponível para uso do Cetec. A Profª. Coordenadora Administrativa do Cetec definiu que seria realizado um "piloto"[20] com alguns professores e turmas ainda este ano. O objetivo de executar um piloto é envolver um número reduzido de pessoas para avaliar a adaptação dos professores e alunos à nova ferramenta da escola.

O projeto "piloto" iniciou-se no mês de novembro e deve durar até o final do período letivo 2010 do Cetec, que se encerra em meados de dezembro/2010. Durante a execução do piloto as dúvida ou problema deverão ser reportados à equipe de desenvolvimento, para o analita de sistemas. A equipe cliente definiu que o Ava será implantado para todos os professores e alunos do Cetec a partir do período letivo de 2011.

Para iniciar a execução do piloto, o analista de sistemas liberou o acesso ao sistema AVA no ambiente Cetec somente aos professores designados pela Profª. Coordenadora Administrativa do Cetec. Foi realizada uma breve apresentação da ferramenta AVA para os professores participantes do piloto afim de apresentar a ferramenta e as funcionalidades disponíveis e orientações de uso. Essa apresentação foi ministrada por uma funcionária do NEAD (Núcleo de Educação à Distância) que é responsável pelos treinamentos dos professores no sistema AVA, com o apoio do analista de sistemas.

Após duas semanas de utilização da ferramenta pelos professores e alunos do piloto foi considerado encerrado o projeto Ava Cetec onde a documentação do projeto ficou concentrada e arquivada na ferramenta Agilefant.

Na subseção seguinte serão apresentados os resultados obtidos com a aplicação do estudo de caso relatado.

7.2.4 Medição do Processo

A avaliação do modelo proposto neste trabalho foi realizada através da aplicação do estudo de caso e da coleta das métricas definidas na fase de medição do processo.

20 Entende-se por piloto a simulação da utilização da ferramenta em ambiente real de trabalho porém com um número reduzido de pessoas, nesse caso professores e alunos.

Os objetivos definidos na fase de medição de processo orientaram as mudanças de processo que foram propostas. As métricas serviram de base para a avaliação do modelo proposto neste trabalho. Conforme sugerido por Sommerville (2007), após a implantação das mudanças de processo as métricas são colhidas com o intuito de responder às questões e por fim, verificar se os objetivos da customização do processo foram alcançados.

Para o estudo de caso que foi aplicado neste trabalho foram definidos dois objetivos: **aumentar a eficiência da equipe de desenvolvimento** e **aumentar a satisfação do cliente** seguindo a abordagem GQM, onde para cada objetivo foram definidas questões e para cada questão métricas, que foram apresentados no capítulo 4 (página 54).

Durante a aplicação do estudo de caso foram aplicadas as seis métricas definidas. Na tabela 21 são descritas as métricas:

Tabela 21: Métricas aplicadas no estudo de caso

	Métricas
1	Percentual da razão entre o número de funcionalidades entregues no prazo sobre o número de funcionalidades estimadas.
2	Percentual da razão entre o número de funcionalidades com erros detectados na fase de testes sobre o número de funcionalidades entregues.
3	Número de contatos formais entre o cliente e a equipe de desenvolvimento durante um ciclo de desenvolvimento.
4	Percentual da razão entre a quantidade de funcionalidades solicitadas depois da entrega sobre o número de funcionalidades entregues.
5	Percentual da razão entre o número de funcionalidades não conformes sobre o número de funcionalidades entregues.
6	Percentual da razão entre o número de funcionalidades com erros detectados pelo cliente sobre o número de funcionalidades entregues.

- **Métrica 1:** As informações para o cálculo da métrica 1 foram o total de funcionalidades entregues no prazo, 7 (sete), e o total de funcionalidades estimadas no projeto, também 7 (sete). A razão desses dois valores é igual a zero, o que corresponde a um resultado de 0%. O resultado dessa métrica tem a seguinte interpretação: quanto mais próximo de 0% (zero por cento) melhor.

- **Métrica 2:** As informações coletadas para calcular a métrica 2 foram o total de funcionalidades com erros detectados durante os testes realizados na fase de desenvolvimento e o total de funcionalidades entregues. O número total de funcionalidades com erros foram 2 (dois) e o total de funcionalidades entregues são 7 (sete). O percentual obtido pela razão desses números foi de 28,6%. O resultado dessa métrica deve ser interpretado como: quanto mais próximo de 0% (zero por

cento) melhor.

- **Métrica 3:** Essa medida não possui cálculo, é um número absoluto. O total de encontros formais realizados entre a equipe de desenvolvimento com o cliente em uma iteração foram 2 (dois). Um dos encontros foi agendado por iniciativa do cliente e outro pelo analista de sistemas. O resultado dessa métrica tem a seguinte interpretação: quanto mais melhor.

- **Métrica 4:** Durante o acompanhamento realizado após a entrega de todas as funcionalidades, foi registrada a necessidade de modificações na funcionalidade "Fórum". Essas modificações foram sugestões de mudanças somente para o ambiente do Cetec, que foram a restrição de *posts* pelos alunos e a permissão do professor bloquear e excluir mensagens. O total de funcionalidades que foram solicitadas modificações foram 1 (uma) onde o total de funcionalidades entregues foram 7 (sete). O percentual dessa razão é de 14,3%. O resultado dessa métrica tem a seguinte interpretação: quanto mais próximo de 0% (zero por cento) melhor.

- **Métrica 5:** Não foram relatas pela equipe cliente funcionalidades não conformes, todas as funcionalidades entreguem estavam de acordo com o solicitado. Ou seja, são 0 (zero) não conformidades divido por 7 (sete) que corresponde ao total de funcionalidades entregues. O percentual obtido dessa divisão é de 0%. O resultado dessa métrica tem a seguinte interpretação: quanto mais próximo de 0% (zero por cento) melhor.

- **Métrica 6:** O número de funcionalidades com erros detectados pelo cliente foram 2 (duas), um erro na funcionalidade "Mensagens" e outro na funcionalidade "Webfólio". Os erros foram reportados ao analista de sistemas e os mesmos já foram corrigidos em ambiente de produção. O percentual do resultado obtido através da divisão de 2 (total de funcionalidades com erros) por 7 (total de funcionalidades entregues) foi 28,6%. O resultado dessa métrica tem a seguinte interpretação: quanto mais próximo de 0% (zero por cento) melhor.

Na tabela 22 são apresentados os cálculos e resultados de cada métrica. Na coluna "Métrica" são descritas as métricas, na coluna "Cálculo" são apresentados os cálculos realizados e a coluna "Resultado" apresenta o resultado obtido para cada métrica.

Tabela 22: Resultados das métricas aplicadas

Métrica	Cálculo	Resultado
Métrica 1	7 / 7 = 0	**0%**
Métrica 2	2 / 7 = 0,286	**28,6%**
Métrica 3		**2**
Métrica 4	1 / 7 = 0,143	**14,3%**
Métrica 5	0 / 7 = 0	**0%**
Métrica 6	2 / 7 = 0,286	**28,6%**

Esses resultados correspondem à primeira medição do novo processo após a aplicação das mudanças. Conforme Sommerville (2007), o objetivo do estágio de medição é aprimorar as medidas de acordo com os objetivos da organização envolvida no aprimoramento de processo. Dessa forma, o ideal é que a medição seja realizada por várias vezes para que seja possível analisar e comparar os resultados.

Durante a realização deste trabalho, por questões de tempo, foi possível realizar somente uma medição. Analisando os resultados obtidos percebe-se que as métricas 2, 4 e 6 devem ser monitoradas e afim de reduzir os percentuais. As três métricas estão relacionadas às atividades de testes, o que indica que a fase de testes é o ponto crítico do processo e deve ser monitorado.

7.3 Considerações Finais

Este capítulo descreveu a aplicação de um estudo de caso com o objetivo de validar o modelo proposto neste trabalho. O estudo de caso foi aplicado em uma equipe de desenvolvimento que apresenta as características definidas no objetivo deste trabalho. Apesar de ter sido um estudo de caso relativamente pequeno e que durou aproximadamente 45 dias, seus objetivos foram atingidos.

Foi possível verificar que:

- A metodologia de desenvolvimento proposta se aplica a pequenas equipes de desenvolvimento, formadas por analista de sistemas e programadores, responsáveis por manutenção evolutiva em *softwares* específicos *web*;

- Relacionado às características ágeis, a metodologia proposta permitiu uma maior comunicação e colaboração entre os envolvidos no projeto, promoveu maior facilidade no planejamento do projeto, além de ter permitido um melhor

acompanhamento e agilidade;

- As práticas de desenvolvimento ágil introduzidas por este estudo de caso foram consideradas bem vindas pela equipe de desenvolvimento, além de não terem afetado negativamente o desenvolvimento do projeto;

- A atividade “Criar história” não foi executada pelo cliente por definição da equipe de desenvolvimento, por esse motivo não foi adequadamente validada. O mesmo ocorreu com a atividade “Codificar” já que os testes unitários não foram codificados no estudo de caso aplicado;

- O artefato Lista de Funcionalidades deve ser adaptado para atender as funcionalidades sob visão da equipe de desenvolvimento também, visto que o artefato proposto neste trabalho atende somente as funcionalidades definidas pela equipe cliente;

Contudo, os resultados alcançados não podem ser generalizados. Acredita-se que estudos mais aprofundados em outras equipes com outros projetos precisam ser observadas antes que generalizações possam ser feitas sobre o uso e eficácia da metodologia de desenvolvimento proposta neste trabalho.

Sobre o uso da ferramenta Agilefant, alguns pontos fracos foram levantados pela equipe de desenvolvimento do Ava:

- Ausência da opção de anexar aquivos no projeto, iterações, histórias e tarefas;

- Ausência de um perfil específico para o “Cliente”. No cadastro de permissões não existe as opções de “Visualização” e permissões por projeto. A ferramenta foi utilizada somente pela equipe de desenvolvimento;

- Ausência da opção de visualizar os gráficos com datas retroativas e salvá-los em arquivo.

8 . CONCLUSÃO

O desenvolvimento deste trabalho propiciou utilizar e colocar em prática os conceitos de customização de processo definido por Sommerville (2007), algumas práticas de desenvolvimento ágil e os conceitos definidos nas normas NBR ISO/IEC 9126-1 e NBR ISO/IEC 14598-1.

Primeiramente foi apresentado o estudo bibliográfico sobre manutenção de *software* e customização de processos de desenvolvimento de *software*. Esse estudo contribuiu para a aquisição de conhecimento sobre diferentes abordagens sobre os temas e permitiram a definição da metodologia de customização de processos de manutenção de *software*.

A metodologia de customização de processos de *software* adotada na execução deste trabalho foi a de Sommerville (2007). Essa metodologia define a melhoria ou customização de processos como "aprimoramento de processo" e o define como uma atividade cíclica que passa por três estágios: medição de processo, análise do processo e mudança de processo.

Os estudos bibliográficos realizados sobre os métodos ágeis de desenvolvimento apresentaram as origens e características dos métodos ágeis *Adaptive Software Development* (ASD), *Dynamic Systems Development Method* (DSDM), *Crystal Methods* (CM), *Feature Driven Development* (FDD), *Lean Development* (LD), *Agile Modeling* (AM), *Extreme Programming* (XP) e *Scrum*. Foi verificado que todos os métodos compartilham em sua essência os valores do Manifesto Ágil, mas possuem algumas diferenças em suas abordagens. De forma geral, suas abordagens baseiam-se fortemente na decomposição de problemas em problemas menores, onde o processo de desenvolvimento ocorre em ciclos curtos e ao final de cada iteração ou conjunto de iterações acontece a entrega de uma parte do sistema.

Nos capítulos 4 e 5 foram apresentados os resultados da aplicação dos estágios de medição e análise do processo atual. A abordagem de medição de processo GQM (*Goal-Question- Metric*) adotada para a definição dos objetivos da customização foi importante por manter o alinhamento com o negócio (objetivos) e processos específicos de nível operacional (questões) . A análise de processo permitiu chegar à um modelo do processo atual e identificar os aprimoramentos a serem implantados. Esses aprimoramentos foram definidos baseados nos objetivos da customização.

A proposta de customização apresentada no capítulo 5 foi norteada pelos aprimoramentos identificados e pelos objetivos da customização, com a premissa de aplicar

práticas sugeridas nos métodos ágeis de desenvolvimento, ou adaptá-las às necessidades da equipe. O novo processo customizado proposto foi baseado na premissa de um processo iterativo e incremental e é composto por três fases, Planejamento, Desenvolvimento e Encerramento. Para apoiar o modelo proposto foi selecionada uma ferramenta através de um processo de avaliação baseado na série de normas NBR ISO/IEC 9126 e 14598. Através dessa avaliação de ferramentas foi possível aplicar os conceitos definidos nestas normas.

E por fim, com o objetivo de validar a metodologia de desenvolvimento proposta, foi aplicado um estudo de caso em uma equipe de desenvolvimento com as características definidas no objetivo deste trabalho. Através do estudo de caso foi possível verificar que os objetivos do trabalho foram alcançados. A metodologia proposta aplica-se para equipes de desenvolvimento compostas por analistas de sistemas e programadores, responsáveis pela manutenção evolutiva de *softwares* específicos *web*.

Com a aplicação do estudo de caso foi possível avaliar a aderência do processo proposto, onde algumas atividades não foram executadas como definidas no modelo proposto. Os testes unitários não foram codificados e as histórias não foram escritas pelo cliente, o que de uma certa forma fere os princípios das metodologias ágeis de desenvolvimento.

O estudo de caso também permitiu identificar os pontos fortes e fracos do processo proposto. Como pontos fortes destacam-se a maior colaboração e comunicação entre os envolvidos no processo. Também promoveu maior agilidade e melhor acompanhamento. Como pontos negativos identificou-se que o artefato Lista de Funcionalidades não atendeu as necessidades da equipe de desenvolvimento, uma vez que o artefato foi criado para listar as funcionalidades definidas pelo cliente. Existem atividades de grande impacto e representatividade que não são percebidas pelo cliente e sim pela equipe técnica. Outro aspecto que influenciou negativamente foi a ferramenta selecionada, Agilefant. A equipe identificou vários pontos que foram falhos e não atenderam totalmente suas necessidades.

Consolidando as experiências durante o estudo de caso, o parecer final da equipe de desenvolvimento do Ava foi positivo em relação à metodologia de desenvolvimento proposta. Utilizando o modelo proposto foi possível aperfeiçoar o processo de desenvolvimento e o relacionamento com as pessoas envolvidas, principalmente o cliente.

Como principais contribuições deste trabalho podem-se citar:

- Definição de uma metodologia para customizar processos de manutenção de *software* baseada no ciclo de aprimoramento sugerido por Sommerville (2007);

- Aplicação da abordagem GQM (*Goal-Question-Metric*);
- Aplicação do processo de avaliação definido na norma NBR ISO/IEC 14598-1;
- Validação através de um pequeno estudo de caso a aplicabilidade da metodologia de desenvolvimento proposta baseada nas características ágeis de desenvolvimento.

Em busca de maior aprimoramento sobre estudos de customização de processos de manutenção de *software* baseados na modelagem ágil, alguns trabalhos futuros são destacados:

- Aplicação da metodologia proposta em outras equipes com as mesmas características e a customização dessa metodologia para as necessidades da equipe e organização específica. Dessa forma aplicando um nova customização de processo e aplicando novas fases de medição do processo.
- Desenvolvimento, aplicação e análise estatística de um questionário mais elaborado sobre a avaliação dos usuários (equipe técnica e cliente) a respeito da metodologia proposta.
- Definição e aplicação de mais medidas seguindo a abordagem GQM (*Goal-Question-Metric*).
- Desenvolvimento (implementação) de uma ferramenta que atende as características específicas do processo proposto, incluindo a utilização de *workflow* e iteração com a equipe cliente seguindo a abordagem de metodologias ágeis.

9 . REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, Pekka *et al*. **New directions on agile methods**: a comparative analysis. Proceedings of the 25th International Conference on Software Engineering.[S.l.]. IEEE Software Society, 2003, p. 244-254.

AGILE MANIFESTO. **Manifesto for Agile Software Development.** 2001. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: 22 mar. 2010.

AMBLER, Scott W., **Modelagem ágil: práticas eficazes para a programação extrema e o processo unificado**. Porto Alegre: Bookman, 2004. 351 p.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO (SOFTEX). **MPS.BR - Guia Geral:2009.** Disponível em <http://www.softex.br/mpsbr/_guias/default.asp>. Acesso em: 4 mai. 2010.

BANKI, Andre L.; TANAKA, Sérgio Akio. **Metodologias Ágeis: Uma Visão Prática.** Engenharia de Software Magazine. Ano I, ed. 04, p.22-29, 2008. Disponível em: <http://www.devmedia.com.br/post-9868-Revista-Engenharia-de-Software-4.html>. Acesso em: 23 mar. 2010.

BASILI, V. R; ROMBACH, H. D. **The TAME project:** towards improvement-oriented software environments. IEEE Transactions on Software Engineering. vol. 14, n. 6, p. 758-773, Junho1988.

BECK, Kent *et al*. **Chysler goes to "extremes"**. Outubro, 1998. Disponível em: <www.xprogramming.com/publications/dc9810cs.pdf>. Acesso em: 1 jun. 2010.

BECK, Kent. **Programação extrema (XP) explicada: acolha as mudanças.** São Paulo: Bookman, 2004. 182 p.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML.** 2.ed. rev. e atual. Rio de Janeiro, RJ: Elsevier, 2007. xvii, 369 p.

BOEHM, Barry. **Get ready for agile methods.** IEEE Computer. Janeiro 2002, p.64–69.

COHEN, David; LINDVALL, Mikael; COSTA, Patrícia. **Agile software development**: a DACS state of art report. NY. Data Analysis Center Software - Fraunhoufer Center for Experimental Software Engineering Maryland and The University of Maryland, 2003. Disponível em: <http://www.thedacs.com/techs/>. Acesso em: 1 jun. 2010.

COLOMBO, Regina Maria Thienne. **Processo de Avaliação da Qualidade de Pacotes de Software**. Campinas: UNICAMP, 2004. Dissertação (Mestrado) - Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Campinas. 2004. 169 p. Disponível em: <http://cutter.unicamp.br/document/?code=vtls000321918>. Acesso em: 3 set. 2010.

FOWLER, Martin. **The New Methodology**, dez, 2005. Disponível em: <http://martinfowler.com/articles/newMethodology.html>. Acesso em: 18 mar. 2010.

GIL, Antonio Carlos. **Métodos e técnicas de pesquisa social.** 5.ed. São Paulo: Atlas, 1999. 206 p.

HIGHSMITH, Jim; COCKURN, Alistair. **Agile software development:** The business of innovation. IEEE Computer, September, 2001. Disponível em: <http://csdl.computer.org/dl/mags/co/2001/09/r9120.pdf>. Acesso em: 3 jun. 2010.

HIGHSMITH, Jim. **Agile Software Development Ecosystems**. Addison Wesley, 2002. 404p.

HIGHSMITH, Jim. **What Is Agile Software Development?** CrossTalk Magazine, Outubro, 2002. Disponível em: <http://www.stsc.hill.af.mil/crosstalk/2002/10/highsmith.html>. Acesso em: 5 jun. 2010.

[IBM, 2010] IBM. **Rational Team Concert Express:** Controle de versões, build, qualidade e entrega de sistemas através de ferramenta integrada**.** Disponível em <http://www.ibm.com/expressadvantage/br/catalogo/software_rational_team_concert_express.phtml>. Acesso em: 2 set. 2010.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO); INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC). **Information technology:** software process assessment : part 1 : concepts and introductory guide. Geneva, Suiça: ISO, 1998. 11 p. (Technical report)

JACOBSON, Ivar. **A Resouding Yes to Agile Processes - But Also Mor.** Janeiro. 2002. Disponível em: <http://cutterconsortium.com/content/itjournal/fulltext/2002/01/itj0201d.html>. Acesso em: 18 mar 2010.

KANE, David; ORNBURN, Steve. **Agile Development:** Weed or Wildflower? CrossTalk Magazine, Outubro, 2002. Disponível em: <http://www.stsc.hill.af.mil/crosstalk/2002/10/kane.html>. Acesso em: 7 jun. 2010.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de metodologia científica.** 6.ed. São Paulo: Atlas, 2005. 315 p.

MARTINS, José Carlos Cordeiro. **Técnicas para Gerenciamento de Projetos de Software**. Rio de Janeiro: Brasport, 2007. 456p.

MINISTÉRIO DO TRABALHO E EMPREGO. **Classificação Brasileira de Ocupações**: CBO 2002. Disponível em: <http://www.mtecbo.gov.br/cbosite/pages/downloads.jsf>. Acesso em: 21 jun. 2010.

MONTAIN GOAT SOFTWARE. **Introduction to Scrum - An Agile Process.** Disponível em: <http://www.mountaingoatsoftware.com/topics/scrum>. Acesso em: 19 jun. 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT); INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). **Tecnologia de informação:** processos de ciclo de vida de software. Rio de Janeiro: ABNT, 1998. 35 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT); INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO); INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC). **Tecnologia de informação:** avaliação de produto de software: parte 1: visão geral. Rio de Janeiro: ABNT, 2001. 14 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT); INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO); INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC). **Engenharia de software:** qualidade do produto: parte 1: modelo de qualidade. Rio de Janeiro: ABNT, 2003. 21 p.

NEURR, Sridhar. MAHAPATRA, Radhakanta; MANGALARAJ, George. **Challenges of migrating to agile methodologies**: organizations must carefully assess their readiness before treading the path of agility. Comunications of the ACM, [S.l], v.48, n.5, p.72-78 , 2005.

OBJECT MANAGEMENT GROUP (OMG). **Business Process Modeling Notation**. Versão1.2. Disponível em < http://www.omg.org/spec/BPMN/1.2>. Acesso em: 15 abr. 2010.

PEREIRA, Júlio Cesar Rodrigues. **Análise de dados qualitativos: estratégias metodológicas para as ciências da saúde, humanas e sociais**. São Paulo, SP: EDUSP, 2001. 156 p.

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

ROCHA, Ana Regina Cavalcanti da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software: teoria e prática.** São Paulo: Prentice Hall, 2001. 303 p.

ROCHA, Ana Regina Cavalcanti da. **Análise e projeto estruturado de sistemas.** Rio de Janeiro: Campus, 1987. 141 p.

SCHWABER, Ken; SUTHERLAND, Jeff. **Scrum Guide**. Novembro, 2009. 22 p. Disponível em: <http://www.scrum.org/scrumguides/>. Acesso em: 3 jun. 2010.

SERGIO, Marbilia Passagnolo, **Processo de Avaliação de Produto Final de Software para Aquisição**. Campinas: UNICAMP, 2004. Dissertação (Mestrado) - Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Campinas. 2004. 170 p. Disponível em: <http://cutter.unicamp.br/document/?code=vtls000332949>. Acesso em: 27 set. 2010.

SOFTWARE ENGINEERING INSTITUTE (SEI). **CMMI for Development (CMMI-DEV)**, Version 1.2, Technical Report CMU/SEI-2006-TR-008. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon Univerity, 2006. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/06tr008.cfm>. Acesso em: 17 mai. 2010.

SOFTWARE ENGINEERING INSTITUTE (SEI). **CMMI - Capability Maturity Model Integration.** Disponível em: <http://www.sei.cmu.edu/cmmi>. Acesso em:17 mai. 2010.

SOMMERVILLE, Ian. **Engenharia de software.** 8.ed. São Paulo: Pearson Addison Wesley, c2007. xiv, 552 p.

WAZLAWICK, Raul Sidnei. **Metodologia de pesquisa para ciência da computação.** Rio de Janeiro: Elsevier, c2009. 159 p.

WOHLIN, Claes; RUNESON, Per; HÖST, Martin; OHLSSON, Magnus C; REGNELL Bjoorn; WESSLÉN, Anders. **Experimentation in Software Engineering:** An Introduction. Boston MA: Kluwer Academic Publishers. USA, 2000. 204p.

ZANATTA, Alexandre Lazaretti. **xScrum:** uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI Florianópolis, 2004. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina. Disponível em <www.tede.ufsc.br/teses/PGCC0651.pdf>. Acesso em: 12 mai. 2010.

ANEXO A – Questionário aplicado na análise do processo atual

Universidade de Caxias do Sul
Centro de Computação e Tecnologia da Informação
Curso de Bacharelado em Sistemas de Informação
Trabalho de Conclusão de Curso
Acadêmica Daiane Morandi da Costa

QUESTIONÁRIO

LEVANTAMENTO DO PROCESSO ATUAL DE DESENVOLVIMENTO DE SOFTWARE

Seção 1 - Introdução

Este questionário possui os seguintes objetivos: a identificação do perfil e conhecimento da equipe, captura de informações do processo atual de desenvolvimento utilizado na manutenção evolutiva, identificação de informações sobre a percepção da equipe em relação ao processo atual, levantar sugestões de melhoria sob o ponto de vista da equipe de desenvolvimento e verificar a existência de interesse por adoção de práticas ágeis de desenvolvimento por parte da equipe.

O estudo está sendo realizado em uma equipe específica responsável pela manutenção evolutiva de um *software web* com orientação a objetos. Portanto partimos do pressuposto que os respondentes deste questionário são membros da mesma equipe de trabalho. O foco é para as atividades de desenvolvimento de software e não de gerenciamento.

Os dados aqui coletados serão utilizados única e exclusivamente em pesquisa acadêmica, sem qualquer finalidade comercial.

Muito obrigada!
Daiane Morandi da Costa
dmcosta@ucs.br

Seção 2 – Qualificação e Caracterização do Respondente

1.) Qual é o seu papel na equipe de desenvolvimento?
( ) Analista de Sistemas
( ) Programador
( ) Outro: _____

2.) A quanto tempo você trabalha na empresa? (informar em anos e meses)

3.) A quanto tempo faz parte desta equipe de desenvolvimento que está sendo estudada? (informar em anos e meses)

4.) Você participou do desenvolvimento do software que hoje realiza manutenção evolutiva?
( ) Sim ( ) Não ( ) Parcialmente

5.) Possui dificuldades em executar a manutenção evolutiva por falta de conhecimento nas funcionalidades já existentes?
( ) Sim ( ) Não ( ) Parcialmente

6.) Você possui conhecimento nos métodos ágeis de desenvolvimento de software?
( ) Sim ( ) Não ( ) Parcialmente

Se SIM ou PARCIALMENTE, em quais? _____

Seção 3 – Informações do processo atual de desenvolvimento de software

7.) Considere que uma metodologia de desenvolvimento de software engloba o processo, os métodos e as ferramentas. Você tem clareza dos métodos, técnicas, processo e ferramentas que compõem a 'metodologia' utilizada hoje pela sua equipe?

( ) Sim ( ) Não ( ) Parcialmente

Comentários:

8.) Definindo processo como o conjunto das atividades executadas, sua sequência e seus executores, você considera que sua equipe possui um processo de desenvolvimento de software bem definido (padronizado)?

( ) Sim ( ) Não ( ) Parcialmente

Comentários:

9.) Descreva brevemente, em termos simples, a sequência de atividades que são executadas no processo atual utilizado pela equipe a partir da solicitação do usuário até a disponibilização em produção. (Lembre-se que estamos considerando somente manutenção evolutiva). Para demonstrar o sequenciamento pode enumerar, exemplo: 1, 2, 3.

10.) Considerando características de seu processo atual, responda:

10.1) Gera muita documentação?
( ) Sim ( ) Não ( ) Parcialmente

10.2) A equipe de desenvolvimento trabalha no mesmo ambiente (sala) de trabalho?
( ) Sim ( ) Não ( ) Parcialmente

10.3) A equipe tem autonomia e poder de decisão sobre suas atividades?
( ) Sim ( ) Não ( ) Parcialmente

10.4) A equipe tem bastante contato diário ou semanal com o cliente (solicitante da manutenção evolutiva)?
( ) Sim ( ) Não ( ) Parcialmente

10.5) Quando ocorrem mudanças que afetam o processo de desenvolvimento, a equipe de adapta facilmente?
( ) Sim ( ) Não ( ) Parcialmente

10.6) A equipe possui alguma forma de comunicação diária?
( ) Sim ( ) Não ( ) Parcialmente

Seção 4 – Informações sobre oportunidades de melhorias

11.) Considerando uma oportunidade de melhoria no seu processo atual de desenvolvimento, enumere os itens abaixo dando sua prioridade de melhorias, ou seja, enumere 1 para o item que você considera mais importante/prioritário, e 17 para o menos importante:

( ) Especificação dos requisitos, entendimento das reais necessidades do usuário.
( ) Comunicação entre equipe técnica durante o desenvolvimento.

( ) Comunicação com clientes (usuários solicitantes) durante todo o processo.
( ) Agilidade (considere desde a entrada da demanda até a liberação em produção).
( ) Documentação na fase de concepção e elaboração.
( ) Documentação na fase de construção (projeto e codificação).
( ) Documentação na fase de transição (implantação).
( ) Rastreabilidade (considere a manutenibilidade).
( ) Padronização do processo.
( ) Ferramentas utilizadas.
( ) Testes unitários.
( ) Testes de sistema (negócio e integrações).
( ) Forma de comunicação e divulgação da liberação em produção.
( ) Refatoração (revisão do código).
( ) Reunião diárias e rápidas de acompanhamento.
( ) Autoavaliação pela equipe de desenvolvimento da liberação efetuada com objetivo de identificar e discutir problemas enfrentados.
( ) Avaliação do usuário em relação ao que foi solicitado *versus* disponibilizado em produção.
Comentários:

12.) Escreva em forma de itens suas sugestões de melhorias para o seu processo atual. Considere os métodos, ferramentas, técnicas, enfim, tudo aquilo que entenda que possa proporcionar melhorias no processo e consequentemente nas atividades desempenhadas em seu dia-a-dia.

Seção 5 – Interesse nos Métodos Ágeis

13.) Você tem interesse em aplicar métodos ágeis em seu processo?
( ) Sim ( ) Não ( ) Parcialmente
Se SIM ou PARCIALMENTE, em quais?_____
Obs.: Responda a questão 14 somente se a repostas da questão 13 for 'Sim' ou 'Parcialmente'.

14.) Cite algumas práticas (atividades, técnicas) dos métodos ágeis que gostaria de aplicar no seu processo de desenvolvimento. Pode citar em forma de itens enumerando pela prioridade (utilizando mesmo critério da questão 11).

Seção 6 – Comentários

Sinta-se a vontade para registrar seus comentários ou fornecer informações que julgar importante.

ANEXO B – Roteiro de entrevista aplicada na análise do processo atual

---

ROTEIRO DE ENTREVISTA ESTRUTURADA

---

1) Levantamento do Processo atual:

1.1) Entradas (demandas de manutenção evolutiva)
1.1.1) De que forma as entradas chegam à equipe?
1.1.2) Existe um padrão? (ferramentas, e-mail, reuniões)
1.1.3) Como é definido se o que foi solicitado será executado? Existem aprovações? Qual o fluxo e as pessoas(papéis) envolvidas ?

1.2) Processamento (desenvolvimento do software)
1.2.1) Como é o processo de desenvolvimento de software executado para realizar a manutenção solicitada considerando o levantamento dos requisitos, análise e modelagem, implementação e testes? Descreva as atividades que são executadas e por quem são realizadas.
1.2.2) Esse processo é padrão? Sempre é assim?
1.2.3) Como são administradas as dúvidas ou problemas que a equipe enfrenta na fase desenvolvimento?
1.2.4) A equipe possui o conhecimento necessário para realizar as manutenções solicitadas?
1.2.5) O usuário acompanha o andamento das atividades? De que forma?

1.3) Saídas (liberação do software para uso em produção)
1.3.1) Quais são as atividades realizadas para a liberação do sistema (nova versão) em ambiente de produção?
1.3.2) Existe testes realizados pelo usuário?
1.3.3) São realizados treinamentos com os usuários?
1.3.4) Como o usuário fica sabendo que as alterações solicitadas já estão disponíveis? Existe uma divulgação?
1.3.5) é gerado alguma documentação para o usuário? (*help*, manual, etc)
1.3.6) Como é o retorno (*feedback*) do usuário em relação ao que foi entregue?

1.4) Informações gerais do processo atual
1.4.1) O processo atual está documentado (formalizado)?
1.4.2) Se acontecer de entrar um novo membro na equipe, como seria o aprendizado desse novo membro?

2) Identificação das Ferramentas e artefatos

2.1) Quais ferramentas utilizadas no processo de desenvolvimento?
2.1.1) Para cada ferramenta, o que (quais) atividades ou artefatos são realizados?

2.2) Quais os artefatos gerados durante as fases de levantamento de requisitos, análise e modelagem, programação, testes e implantação?
2.2.1) Este artefatos são armazenados aonde? É utilizado uma ferramenta CASE? Ou são descartados?

2.3) Alguma documentação é gerada?
2.3.1) Qual ou que tipo?
2.3.2) Onde são armazenadas? Em papel ou ferramentas?
2.3.3) Quem tem acesso?
2.3.4) A qual documentação o usuário tem acesso?

3) Percepção da equipe

3.1) Fale sobre os pontos fortes (do processo e equipe)
3.2) Fale sobre os pontos fracos (do processo e equipe)
4.2.1) O que você sugere para minimizar ou acabar com essas fraquezas?

3.3) Quais são os problemas que você enfrenta no seu dia-a-dia?
4.3.1) O que já foi feito para resolver estes problemas?

3.4) Você sente necessidade de melhorias no seu processo atual, considerando as ferramentas, atividades, documentação, artefatos, pessoas, comunicação, etc...
3.5) Em relação à adoção de práticas sugeridas pelos métodos ágeis, qual a percepção da equipe?
3.5.1) A equipe utiliza alguma prática sugerida pelos métodos ágeis?
3.5.2) A equipe gostaria de adaptar seu processo atual introduzindo práticas ágeis de desenvolvimento? Quais?

4) Indicadores e medição do processo atual

4.1) Como é estabelecido o cronograma (prazo)? Onde fica registrado? E como são atualizados?
4.2) Quais são os controles realizados no processo atual em termos de acompanhamento?
4.3) Quais os indicadores ou metas? Como são atualizados e acompanhados?
4.4) Existe controle de erros, produtividade, retrabalho? Quais e como são avaliados?

ANEXO C – Documentação gerada no processo atual

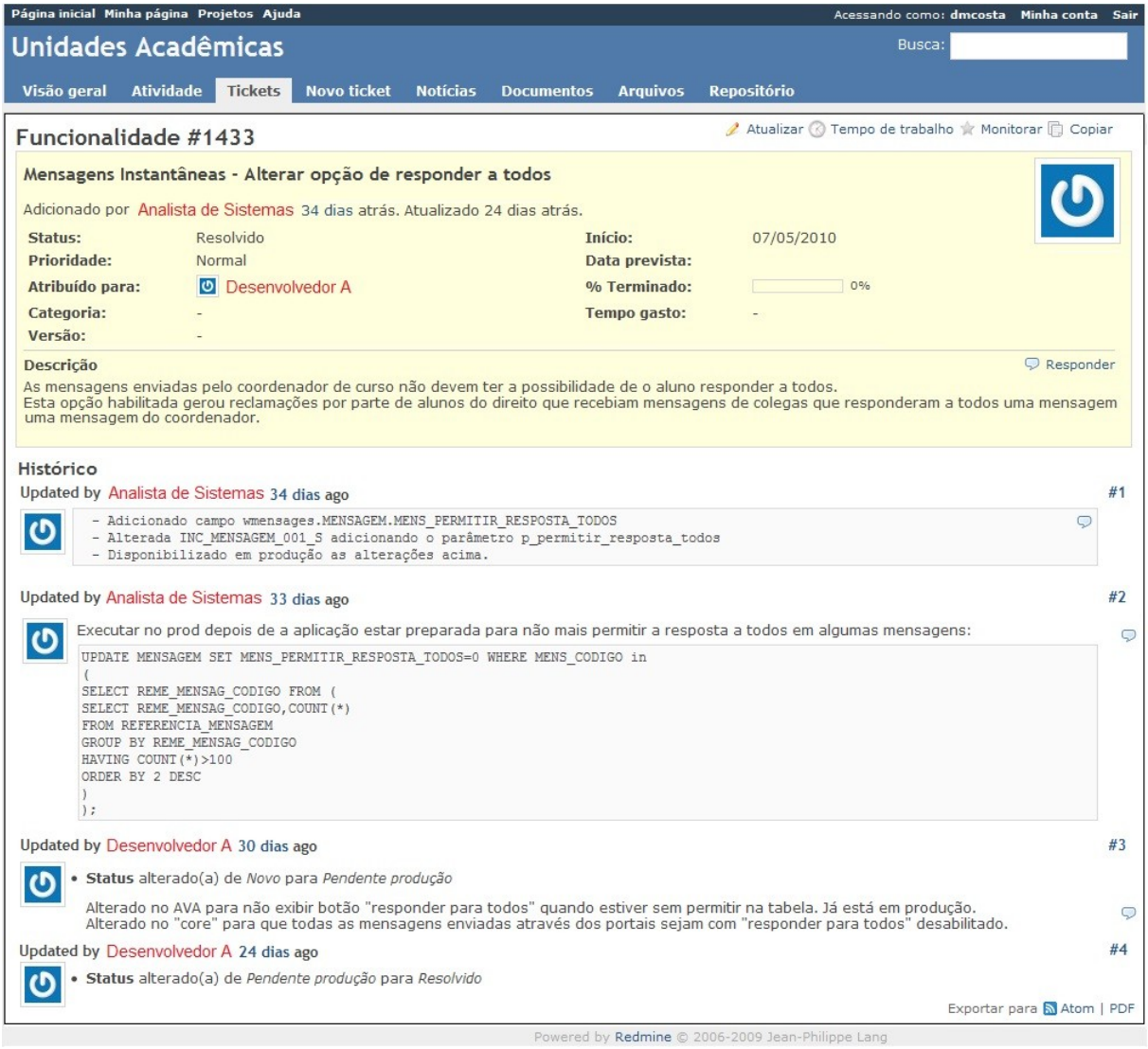


Figura 50: Documentação gerada no processo atual

ANEXO D – Resumo do estudo das Normas NBR ISO/IEC 9126 e 14598

1. Normas de qualidade do produto de *software*

Qualidade de *software* pode ser compreendida como um conjunto de características que devem ser alcançadas em um determinado nível para que o produto atenda as necessidades de seus usuários. Dessa maneira a qualidade de um produto de *software* será descrita e avaliada em virtude do conjunto de características que foram determinadas como necessárias (ROCHA, MALDONADO e WEBER, 2001).

Em busca da qualidade do produto de *software* as organizações que produzem ou adquirem, estão procurando estruturar métodos para avaliá-los. O conjunto de características que são apontadas como necessárias para a qualidade, devem ser organizadas para uma melhor avaliação do produto de *software* em questão. Portanto torna-se necessária a escolha de um modelo que organize e oriente da melhor maneira, a avaliação deste produto (ROCHA, MALDONADO e WEBER, 2001).

Quanto à abordagem da qualidade do produto de *software*, considerando produto de *software* como o foco da avaliação, tem-se a série de normas NBR ISO/IEC 9126 (ABNT, 2003) e 14598 (ABNT, 2001), que definem um modelo de qualidade para produtos de *software* e processos de avaliação do *software* (ROCHA, MALDONADO e WEBER, 2001) e estão definidas nos seguintes documentos:

a) Qualidade de Produto de software

- *ISO / IEC 9126-1 - Modelo de Qualidade:* Define um modelo de qualidade para produtos de *software*, apresentando um conjunto de características e suas subcaracterísticas;

- *ISO / IEC 9126-2 - Métricas Externas:* Apresenta métricas externas para medir os atributos das características de qualidade definidas na NBR ISO/IEC 9126-1. Estas métricas representam a perspectiva externa da qualidade do produto de *software*, quando o mesmo já está pronto para execução;

- *ISO / IEC 9126-3 - Métricas Internas:* Apresenta métricas internas para medir os atributos das características de qualidade definidas na NBR ISO/IEC 9126-

1. Estas métricas representam a perspectiva interna da qualidade do produto de *software* e estão associados a produtos intermediários, como projeto e código;

- *ISO / IEC 9126-4 - Métricas da Qualidade em Uso:* Apresenta métricas de qualidade em uso para medir os atributos de qualidade definidas na NBR ISO/IEC 9126-1. Estas métricas representam a perspectiva do usuário para qualidade do produto de *software*;

b) Avaliação de produtos de software

- *ISO / IEC 14598-1: Visão Geral:* Fornece uma visão do processo de avaliação da qualidade dos produtos de *software* e define toda a estrutura de funcionamento da série de normas NBR ISO/IEC 14598;

- *ISO / IEC 14598-2 - Planejamento e Gestão:* Refere-se ao planejamento e gestão do processo de avaliação apresentando requisitos, recomendações e orientação para uma função de suporte ao processo;

- *ISO / IEC 14598-3 - Processo para Desenvolvedores:* Define o processo para desenvolvedores. Destina-se ao uso durante o processo de desenvolvimento e manutenção de *software*;

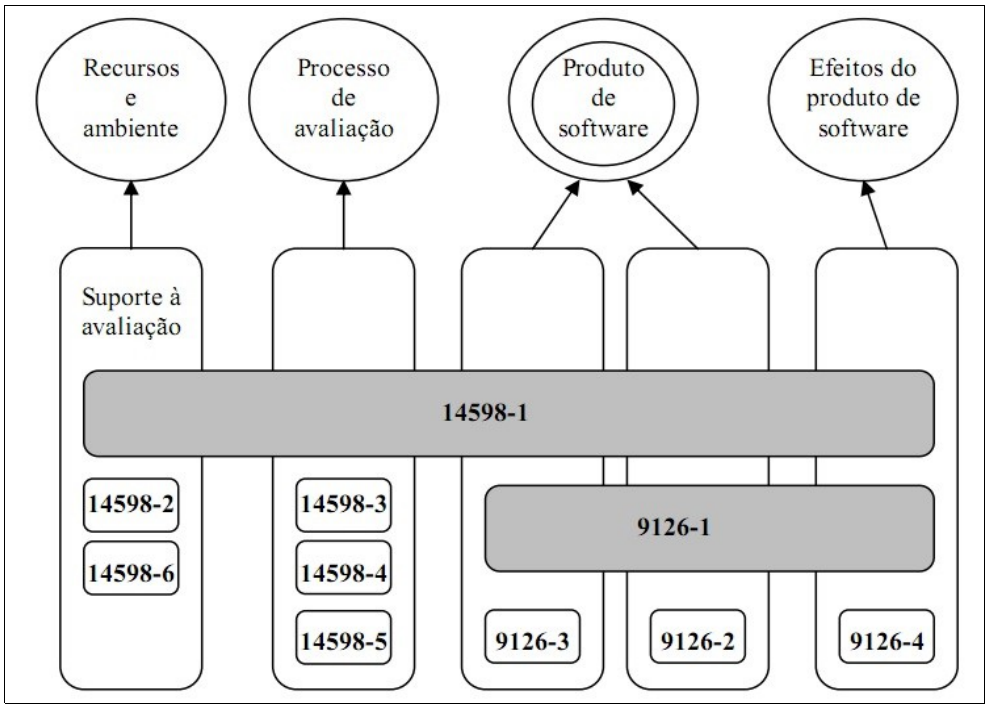
- *ISO / IEC 14598-4 - Processo para Adquirentes:* Define o processo para adquirentes, estabelece um processo sistemático para avaliação de: produtos de *software* tipo pacote, produtos de *software* sob encomenda, ou ainda modificações em produtos já existentes;

- *ISO / IEC 14598-5 - Processo para Avaliadores:* Define o processo para avaliadores, fornecendo orientação para a implementação prática de avaliação de produtos de *software*;

- *ISO / IEC 14598-6 - Documentação de Módulos de Avaliação:* Fornece orientação para documentação de módulos de avaliação. Este módulo contém a especificação do modelo de qualidade, as informações e dados relativos à aplicação prevista do modelo e informações sobre a real aplicação do modelo.

A figura 51 mostra o relacionamento entre as normas das séries NBR ISO/IEC 9126 e

14598, deixando clara a abrangência da norma NBR ISO/IEC 14598-1 sobre todo o processo de avaliação, bem como a necessidade do uso da norma NBR ISO/IEC 9126-1 como referência na aplicação das métricas.



Fonte: ABNT (2003)

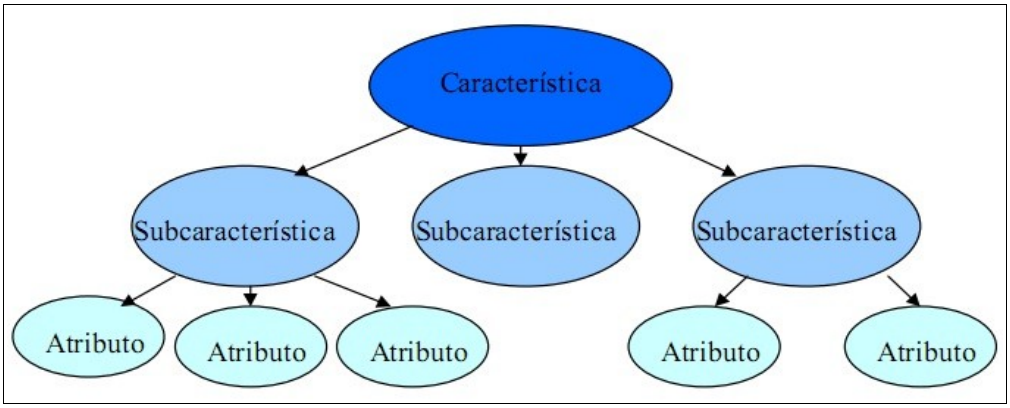
Figura 51: Relacionamento entre as normas NBR ISO/IEC 9126 e 14598

1.1 Qualidade dos produtos de software

Segundo Rocha, Maldonado e Weber (2001), o modelo de qualidade definido na norma NBR ISO/IEC 9126-1 é um modelo de referência para o processo de avaliação da qualidade de produto de *software*. Essa norma está subdividida em duas partes: modelo de qualidade para características externas e internas e modelo de qualidade para qualidade em uso.

O modelo de qualidade para características externas e internas classifica os atributos de qualidade de *software* em seis características que são, por sua vez, desdobradas em subcaracterísticas. As subcaracterísticas podem ser desdobradas em mais níveis que caracterizam os atributos de qualidade. As métricas internas e externas aplicam-se, em geral, ao nível dos atributos de qualidade. Esse modelo de qualidade apresenta uma estrutura hierárquica definindo características, subcaracterísticas de qualidade e atributos, como mostra

a figura 52.



Fonte: ABNT (2003)

Figura 52: Modelo de qualidade

A norma NBR ISO/IEC 9126-1 estabelece um conjunto de seis características da qualidade de um produto de *software* e suas respectivas subcaracterísticas conforme descritas abaixo:

a) Funcionalidade: Capacidade do produto de *software* de prover funções que atendam às necessidades explícitas e implícitas, quando o *software* estiver sendo utilizado sob condições especificadas. Tem como subcaracterísticas: adequação, acurácia, interoperabilidade, segurança de acesso e conformidade.

b) Confiabilidade: Capacidade do produto de *software* de manter um nível de desempenho especificado, quando usado em condições especificadas. Tem como subcaracterísticas: maturidade, tolerância a falhas, recuperabilidade e conformidade.

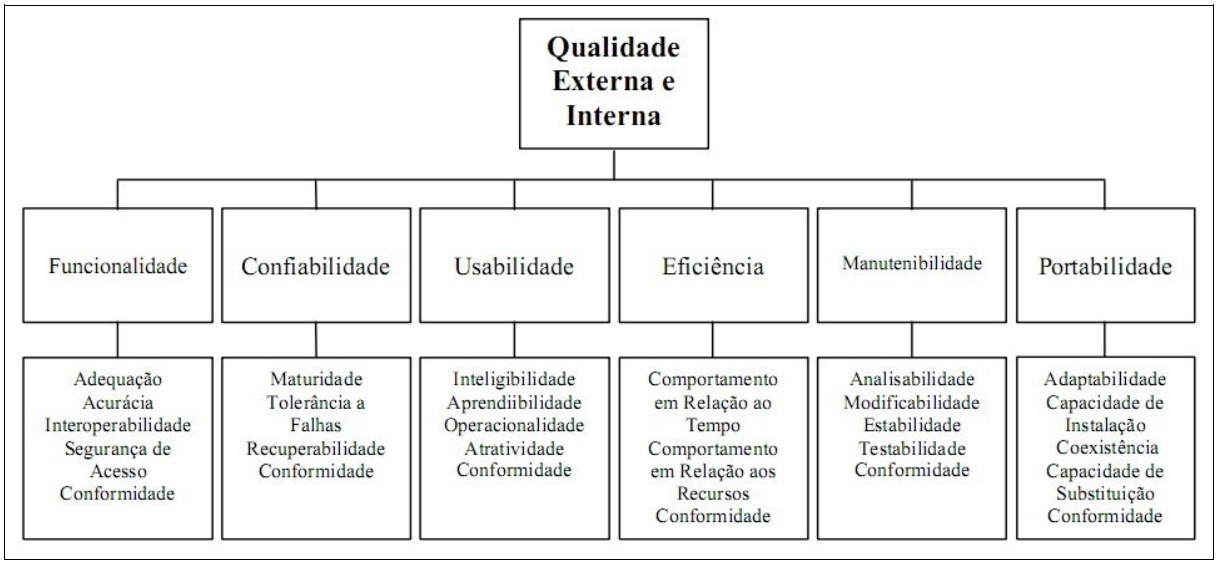
c) Usabilidade: Capacidade do produto de *software* de ser compreendido e utilizado, bem como o julgamento individual de tal uso por um conjunto explícito ou implícito de usuários. Tem como subcaracterísticas: inteligibilidade, apreensibilidade, operacionalidade, atratividade, e conformidade.

d) Eficiência: Capacidade do produto de *software* de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições especificadas. Tem como subcaracterísticas: comportamento em relação ao tempo, comportamento em relação aos recursos e à conformidade.

e) Manutenibilidade: Capacidade do produto de *software* de ser modificado. Tem como subcaracterísticas: analisabilidade, modificabilidade, estabilidade, testabilidade e conformidade.

f) Portabilidade: Capacidade do produto de *software* de ser transferido de um ambiente para outro. Tem como subcaracterísticas: adaptabilidade, capacidade para ser instalado, coexistência, capacidade para substituir e conformidade.

Na figura 53 são demonstradas as seis características e suas subcaracterísticas conforme o modelo de qualidade especificado na norma ISO / IEC 9126-1.



Fonte: ABNT, (2003)

Figura 53: Modelo de qualidade para características externas e internas

A qualidade em uso é a visão da qualidade do produto de *software*, do ponto de vista do usuário, quando este produto é usado em ambiente e contexto específicos. Podendo se dizer que é o efeito combinado das seis características de qualidade externa e interna do produto de *software* (SERGIO, 2004).

Já Rocha, Maldonado e Weber (2001) definem a qualidade em uso como "a capacidade do produto de permitir que usuários específicos atinjam metas especificadas com eficácia, produtividade, segurança e satisfação em seus respectivos contextos."

No modelo de qualidade em uso, os atributos são classificados em quatro características: *efetividade, produtividade, segurança e satisfação* (NBR ISO/IEC 9126-1), descritas abaixo:

a) Efetividade: é a capacidade de permitir que usuários atinjam metas especificadas com acurácia e completude, em um contexto de uso especificado.

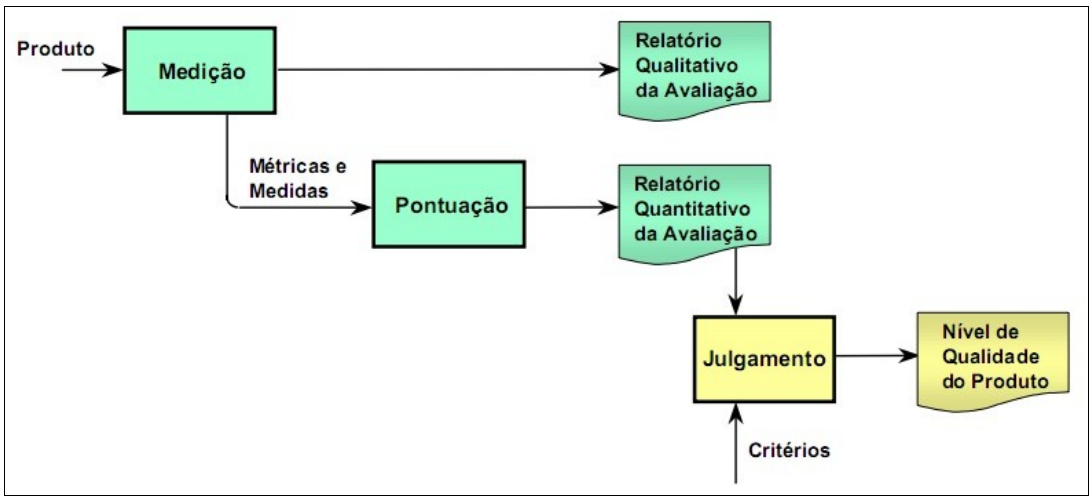
b) Produtividade: é a capacidade de permitir que seus usuários empreguem quantidade apropriada de recursos em relação à eficácia obtida, em um contexto de uso.

c) Segurança: é a capacidade de apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, *software*, propriedades ou ao ambiente, em um contexto de uso.

d) Satisfação: é a capacidade de satisfazer usuários, em um contexto de uso especificado.

O modelo de qualidade especificado na norma NBR ISO/IEC 9126-1 pode ser aplicável na fase de definição de requisitos de qualidade de um *software* a ser desenvolvido, e também na fase de aceitação de produtos, avaliando produtos intermediários e o produto final.

A avaliação de um produto de *software* nesse contexto traz consigo três importantes conceitos: **Medição**, **Pontuação** e **Julgamento** (ABNT, 2003), como pode ser observado na figura 54 que mostra a avaliação segundo a norma ISO/IEC 9126-1.



Fonte: adaptado de ABNT (2003)

Figura 54: Avaliação da qualidade do software segundo NBR ISO/IEC 9126-1

Medição é a ação de aplicar as medições escolhidas ao produto de *software*, onde o resultado é dado em valores nas escalas das medições. **Pontuação** é o processo técnico de medição da qualidade, onde o nível de pontuação é determinado a partir do valor medido.

Julgamento é a emissão de um juízo sobre a qualidade do produto de *software*, no qual um conjunto de níveis pontuados é sintetizado. (ABNT, 2003).

Através da medição de um produto de *software*, é possível conhecer qualitativamente o nível de qualidade; após a pontuação, é possível conhecer quantitativamente o nível de qualidade. Assim, com a aplicação dos critérios de julgamento, é possível fazer o julgamento da qualidade (ABNT, 2003).

Os requisitos de qualidade não devem ser utilizados com a finalidade de atingir a qualidade perfeita, mas a qualidade necessária e suficiente para cada contexto específico de uso (ABNT, 2003).

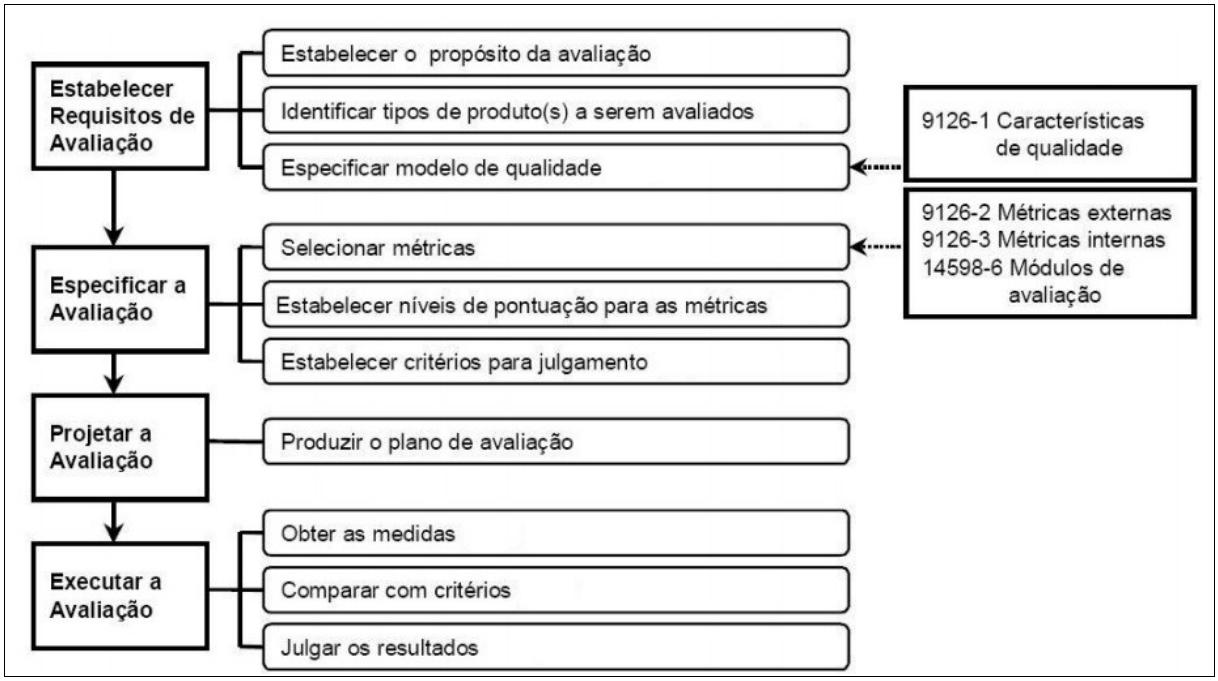
A maior contribuição da norma ISO/IEC 9126-1 é ter definido e estabelecido um modelo teórico de qualidade. Contudo, para a aplicação prática desse modelo dentro de um processo de avaliação de produto de *software*, ela deve ser usada em conjunto com a série ISO/IEC 14598 (SERGIO, 2004).

1.2 Avaliação dos produtos de software

O processo de avaliação dos produtos de *software* está definido na série de normas ISO/IEC 14598, que deve ser utilizada em conjunto com a série ISO/IEC 9126 (ROCHA, MALDONADO, WEBER, 2001).

A norma NBR ISO/IEC 14598-1 fornece uma estrutura para avaliar a qualidade de *software*, define um modelo de processo de avaliação genérico e estabelece os requisitos para a medição e avaliação. Junto com as normas NBR ISO/IEC 14598-2 e 14598-6 estabelece os itens necessários para o suporte à avaliação. Já as normas NBR ISO/IEC 14598-3, 14598-4 e 14598-5 estabelecem o processo de avaliação específico para desenvolvedores, adquirentes e avaliadores de *software* (ABNT, 2001).

O processo de avaliação proposto pela norma 14598-1 (ABNT, 2001) é apresentado na figura 55. Contém quatro etapas, distribuídas em dez atividades, e fornece requisitos gerais de avaliação. Descreve aspectos comuns entre as outras partes da norma e faz recomendações práticas para a implementação da avaliação de um produto de *software*.



Fonte: ABNT (2001)

Figura 55: Processo de avaliação segundo norma NBR ISO/IEC 14598-1

As quatro etapas do processo de avaliação segundo a NBR ISO/IEC 14598-1, demonstradas na figura 24, são sintetizadas a seguir:

a) Estabelecer requisitos de avaliação: Visa levantar os requisitos gerais da avaliação.

- **Estabelecer o propósito da avaliação:** Define quais os objetivos da avaliação. Tais objetivos estão relacionados ao uso pretendido do produto e tenta assegurar a qualidade requerida.

- **Identificar tipos de produto(s) a serem avaliados:** Define o tipo de produto a ser avaliado, se são um dos produtos intermediários ou o produto final.

- **Especificar modelo de qualidade:** A primeira etapa na avaliação de *software* consiste em selecionar as características de qualidade relevantes, utilizando um modelo de qualidade que desdobre a qualidade de *software* em diferentes características. Nesta fase da avaliação é escolhido o modelo de qualidade a ser utilizado visando definir os requisitos de qualidade para o produto de *software*.

b) Especificar a avaliação: Define a abrangência da avaliação e das medições a serem realizadas sobre o produto submetido para avaliação.

- **Selecionar métricas:** A forma pela qual as características de qualidade têm sido definidas não permite sua medição direta. É necessário estabelecer métricas que se correlacionem às características do produto de *software*. Nesta fase da avaliação são selecionadas as métricas a serem utilizadas durante a avaliação.

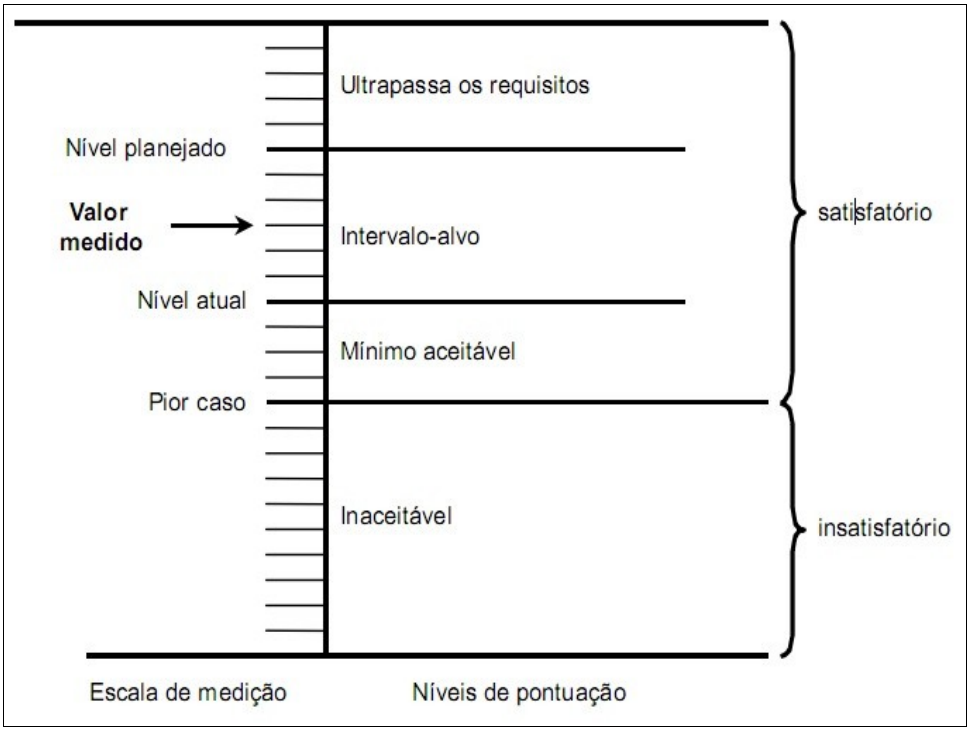
- **Estabelecer níveis de pontuação para as métricas:** Para cada métrica selecionada devem ser definidos os níveis de pontuação e uma escala relacionada, onde poderão ser representados os vários níveis que os atributos poderão assumir.

  A escala atribui rótulos numéricos aos atributos de uma entidade, resultando a representação da fidedignidade do atributo (PEREIRA, 2001). Ela é um conjunto de valores com propriedades definidas.

  Exemplos de tipos de escalas são: escala **nominal,** que corresponde a um conjunto de categorias; escala ordinal, que corresponde a um conjunto ordenado de pontos de escala; escala de intervalos, que corresponde a uma escala ordenada com pontos de escala equidistantes; escala de proporção, que tem pontos de escala equidistantes e também um zero absoluto (COLOMBO, 2004).

  Métricas usando escalas ordinais ou nominais produzem dados qualitativos; métricas usando escalas de intervalos ou de proporção produzem dados quantitativos (NBR ISO/IEC 14598-1, 2001).

  De acordo com a NBR ISO/IEC 14598-1 (NBR ISO/IEC 14598-1, 2001), uma escala pode assumir diferentes níveis de pontuação, como mostra a figura 56. O nível de pontuação deve ter um valor limite entre o que seja satisfatório e insatisfatório.



Fonte: ABNT (2001)

Figura 56: Níveis de pontuação para as métricas

- **Estabelecer critérios para julgamento:** Facilitar a interpretação dos resultados da avaliação de cada característica sintetizando-os. É desejável que se utilize de um método para isto, onde cada característica poderá ser representada em termos de suas subcaracterísticas ou de uma combinação ponderada de suas subcaracterísticas.

c) Projetar a avaliação: Deve-se documentar os procedimentos a serem utilizados pelo avaliador para realizar as medições contidas na especificação de avaliação.

- **Produzir plano de avaliação:** O avaliador deve produzir um plano de avaliação que descreva os recursos necessários para realizar a avaliação especificada, bem como a distribuição destes recursos entre as diversas ações a serem realizadas.

d) Executar a avaliação: Obter os resultados da execução das ações de medição e verificação do produto de *software* de acordo com os requisitos de avaliação, como especificado na especificação de avaliação e planejado no plano de

avaliação.

- **Tomar as medidas:** As métricas selecionadas são aplicadas ao produto de *software* obtendo como resultado valores nas escalas das métricas.

- **Comparar com critérios:** O valor medido para cada métrica é comparado com os critérios determinados na especificação da avaliação.

- **Julgar os resultados:** O julgamento é a etapa final da avaliação, onde um conjunto de níveis pontuados é resumido. O resultado é uma declaração de quanto o produto de *software* atende aos requisitos de qualidade.

ANEXO E – Tabulação das notas da avaliação de ferramentas

Requisitos		Peso	AgileFant		Ágilo		Explanpmt		IceScrum		Rational TC		ScrumWorks		VersionOne		Xplanner	
			N	P	N	P	N	P	N	P	N	P	N	P	N	P	N	P
Requisitos funcionais																		
RF_01	Permitir controle por projetos	3	2	6	0	0	2	6	2	6	2	6	2	6	2	6	2	6
RF_02	Possuir iterações ou sprints	3	2	6	2	6	2	6	2	6	2	6	2	6	2	6	2	6
RF_03	Possuir cartão de histórias	3	2	6	1	3	2	6	2	6	2	6	1	3	2	6	2	6
RF_04	Possuir cartão de tarefas	3	2	6	1	3	1	3	1	3	1	3	1	3	2	6	2	6
RF_05	Possuir controle de horas trabalhadas por recurso	3	2	6	1	3	0	0	0	0	2	6	0	0	2	6	2	6
RF_06	Possuir gráficos de acompanhamento	2	2	4	1	2	1	2	2	4	2	4	2	4	2	4	2	4
RF_07	Possuir consultas de acompanhamento	3	2	6	2	6	1	3	2	6	2	6	2	6	2	6	2	6
RF_08	Possuir relatórios de acompanhamento	2	1	2	2	4	0	0	2	4	2	4	2	4	2	4	1	2
RF_09	Possuir cadastro de equipes (recursos)	1	2	2	2	2	2	2	0	0	1	1	1	1	2	2	0	0
RF_10	Possuir mural virtual de lembretes (planejamento de atividades)	2	2	4	2	4	0	0	1	2	2	4	0	0	2	4	0	0
Requisitos Não funcionais																		
RNF_01	Ser um software livre (free)	3	2	6	2	6	2	6	2	6	0	0	0	0	0	0	2	6
RNF_02	Possuir versão web	3	2	6	2	6	2	6	2	6	2	6	2	6	2	6	2	6
RNF_03	Permitir que seja utilizado nos navegadores Firefox e I.E.	3	2	6	2	6	2	6	2	6	2	6	2	6	2	6	2	6
RNF_04	Utilizar banco de dados free	3	2	6	2	6	2	6	2	6	2	6	2	6	2	6	2	6
RNF_05	Possuir versão com idioma português	2	0	0	0	0	0	0	0	0	2	4	0	0	0	0	0	0
RNF_06	Possuir controle de acesso por usuários no sistema	3	2	6	2	6	2	6	2	6	2	6	2	6	2	6	2	6
RNF_07	Possuir cadastro de permissões por projeto	2	0	0	0	0	1	2	1	2	2	4	0	0	2	4	2	4
RNF_08	Possuir interface amigável	3	2	6	1	3	1	3	1	3	2	6	2	6	2	6	2	6
RNF_09	Possuir facilidade na operação	3	2	6	0	0	1	3	1	3	1	3	2	6	1	3	2	6
RNF_10	Deve se comportar de maneira estável (ausência de erros)	3	2	6	1	3	2	6	2	6	2	6	2	6	2	6	0	0
RNF_11	Possuir mensagens compreensíveis	2	1	2	1	2	2	4	2	4	2	4	2	4	2	4	1	2
	Pontuação Requisitos Funcionais			48		33		28		37		46		33		50		42
	Pontuação Requisitos Não Funcionais			50		38		48		48		51		46		47		48
	Pontuação Total			**98**		**71**		**76**		**85**		**97**		**79**		**97**		**90**
	Percentual sobre o total de pontos possíveis (110)			**89,1%**		**64,5%**		**69,1%**		**77,3%**		**88,2%**		**71,8%**		**88,2%**		**81,8%**

Figura 57: Notas e pontuações da avaliação de ferramentas