UNIVERSIDADE FEDERAL DO PARANÁ

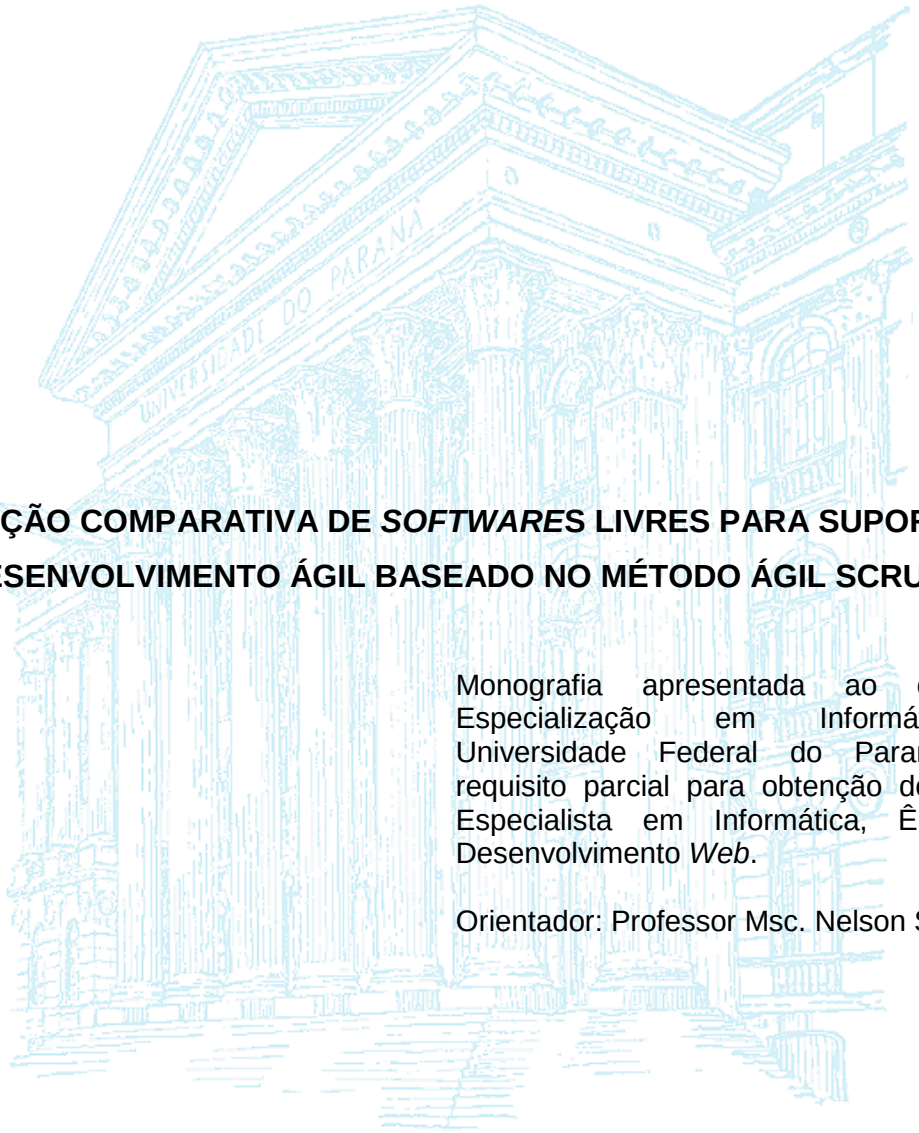
OTAVIO AUGUSTO FELTRIN

AVALIAÇÃO COMPARATIVA DE *SOFTWARE*S LIVRES PARA SUPORTE AO DESENVOLVIMENTO ÁGIL BASEADO NO MÉTODO ÁGIL SCRUM

CURITIBA

2012

OTAVIO AUGUSTO FELTRIN



AVALIAÇÃO COMPARATIVA DE *SOFTWARES* LIVRES PARA SUPORTE AO DESENVOLVIMENTO ÁGIL BASEADO NO MÉTODO ÁGIL SCRUM

Monografia apresentada ao curso de Especialização em Informática da Universidade Federal do Paraná, como requisito parcial para obtenção do título de Especialista em Informática, Ênfase em Desenvolvimento *Web*.

Orientador: Professor Msc. Nelson Suga.

CURITIBA

2012

PARECER DE APROVAÇÃO

MONOGRAFIA DE ESPECIALIZAÇÃO EM INFORMÁTICA

ÊNFASE EM DESENVOLVIMENTO WEB

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA/UFPR

Declaramos que o aluno **Otavio Augusto Feltrin** entregou a versão final da sua Monografia de Especialização em Informática da Universidade Federal do Paraná, com Ênfase em Desenvolvimento Web, intitulada **Avaliação Comparativa de *Softwares* Livres para Suporte ao Desenvolvimento Ágil Baseado no Método Ágil Scrum**.

Curitiba, 25 de abril de 2012.

Prof. Msc. NELSON SUGA
Professor Adjunto
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 - Curitiba-PR

Prof. Msc. SETEMBRINO SOARES FERREIRA JUNIOR
Professor Assistente
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 - Curitiba-PR

RESUMO

Como a gestão eficiente de projetos em uma empresa passou a representar vantagem competitiva, o *Scrum* é a metodologia ágil mais usada nas equipes de desenvolvimento de sistemas e o percentual de *softwares* livres nas organizações, em geral, ainda é muito baixo, este trabalho tem como objetivo realizar uma avaliação comparativa entre três *softwares* livres para suporte ao desenvolvimento ágil, que possuem seu funcionamento baseado nos princípios da metodologia ágil *Scrum.* Para tanto foi realizada uma busca na *internet* sobre *softwares* que se enquadrassem na licença de *software* livre e de acordo com outros critérios de seleção deste trabalho. Foram selecionadas três ferramentas para a avaliação, sendo elas: o *Agilefant*, *Firescrum* e o *Kunagi.* O processo de avaliação seguiu as recomendações da norma NBR ISO/IEC 14598-1 e as características de qualidade foram definidas em função dos princípios do *Scrum* e das recomendações da norma NBR ISO/IEC 9126-1. Ao final, com a pontuação determinada para cada sistema, uma comparação foi feita para determinar qual a melhor ferramenta. Nos resultados tem-se a ferramenta *Kunagi* em primeiro lugar, seguida por *Agilefant* e então pelo *Firescrum*. Espera-se, com o resultado deste trabalho, incentivar o aumento do uso de *software* livre nas organizações, servindo de referência na tomada de decisões e poupando o tempo dos gerentes de projetos de *software* que buscam por ferramentas livres de projeto voltadas à metodologia ágil *Scrum.*

Palavras-chave: Avaliação Comparativa. Gestão de Projetos. Metodologia Ágil. *Scrum*. *Software* livre.

ABSTRACT

As the efficient project management became a competitive advantage, the Scrum is the most used agile methodology for system development teams and the amount of free software used by the companies is still very low, this paper is aimed to making a benchmarking of three free softwares based on agile methodology called Scrum. For this purpose, an internet search was made for softwares that accord with the free software license and with other requirements from this paper. Three tools were selected for the evaluation: Agilefant, Firescrum and Kunagi. The comparison process followed the specifications of NBR ISO/IEC 14598-1 and the quality characteristics were defined according to the principles of Scrum and the recommendations of NBR ISO/IEC 9126-1. At the end, with the final score for each system, a comparison was done to determine the best tool. The results shows Kunagi as the best followed by Agilefant and Firescrum. With this work it is expected to encourage the use of free software in the companies, being used as reference for future decisions and providing information for saving the time of managers who search for free software based on Scrum.

Keywords: Agile Methodology. Free Software. Project Management. Scrum. Software evaluation.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – MODELO EM CASCATA 14
FIGURA 2 – DESENVOLVIMENTO EVOLUCIONÁRIO 15
FIGURA 3 – DESENVOLVIMENTO INCREMENTAL 16
FIGURA 4 – O PROCESSO *SCRUM* 25
FIGURA 5 – CARTAS DO *PLANNING POKER* 28
FIGURA 6 – EXEMPLO DE UM *SPRINT BURNDOWN CHART* 30
FIGURA 7 – QUADRO *KANBAN* 31
FIGURA 8 – ARQUITETURA MACRO DO *FIRESCRUM* 52
FIGURA 9 – GRÁFICO COMPARATIVO DA PONTUAÇÃO FINAL 76

LISTA DE QUADROS

QUADRO 1 – QUANTIDADE DE *DOWNLOADS*.....50
QUADRO 2 – CARACTERÍSTICAS *VERSUS SOFTWARES*.....54
QUADRO 3 – QUADRO COMPARATIVO FINAL 75

LISTA DE TABELAS

TABELA 1 – SUBCARACTERÍSTICAS DA NBR ISO/IEC 9126-1 41
TABELA 2 – CRITÉRIOS DE QUALIDADE - FUNCIONALIDADE 56
TABELA 3 – CRITÉRIO DE QUALIDADE - CONFIABILIDADE 57
TABELA 4 – CRITÉRIO DE QUALIDADE - USABILIDADE 57
TABELA 5 – CRITÉRIO DE QUALIDADE - EFICIÊNCIA 58
TABELA 6 – CRITÉRIO DE QUALIDADE - MANUTENIBILIDADE 58
TABELA 7 – CRITÉRIO DE QUALIDADE - PORTABILIDADE 58
TABELA 8 – VALORES DAS MÉTRICAS 59
TABELA 9 – PESO DA MÉTRICA 60
TABELA 10 – PESOS DO CRITÉRIO DE QUALIDADE - FUNCIONALIDADE 61
TABELA 11 – PESOS DO CRITÉRIO DE QUALIDADE - CONFIABILIDADE 62
TABELA 12 – PESOS DO CRITÉRIO DE QUALIDADE - USABILIDADE 62
TABELA 13 – PESOS DO CRITÉRIO DE QUALIDADE - EFICIÊNCIA 63
TABELA 14 – PESOS DO CRITÉRIO DE QUALIDADE - MANUTENIBILIDADE 63
TABELA 15 – PESOS DO CRITÉRIO DE QUALIDADE - PORTABILIDADE 64
TABELA 16 – PROCEDIMENTOS 66
TABELA 17 – AVALIAÇÃO DA FERRAMENTA *FIRESCRUM* 69
TABELA 18 – AVALIAÇÃO DA FERRAMENTA *AGILEFANT* 71
TABELA 19 – AVALIAÇÃO DA FERRAMENTA KUNAGI 73

LISTA DE SIGLAS

ABNT	- Associação Brasileira de Normas Técnicas
AGPL	- GNU *Affero General Public License*
BSD	- *Berkeley Software Distribution*
DSDM	- *Dynamic System Development*
GPL	- GNU *Public License*
IEC	- *International Electrotechnical Commission*
ISF	- Instituto Sem Fronteiras
ISO	- *International Organization for Standardization*
JDK	- *Java Development Kit*
MIT	- *Massachusetts Institute of Technology*
PMBOK	- *Project Management Body of Knowledge*
RIA	- *Rich Internet Applications*
TDD	- *Test Driven Development*
WIP	- *Work In Progress*
XP	- *Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO 7
1.1 TEMA 9
1.2 PROBLEMA 10
1.3 HIPÓTESE 10
1.4 OBJETIVOS 10
1.4.1 Objetivo Geral 10
1.4.2 Objetivos Específicos 11
1.5 JUSTIFICATIVA 11
1.6 ESTRUTURA DA MONOGRAFIA 12
2 REVISÃO DA LITERATURA 13
2.1 MODELOS DE PROCESSO DE *SOFTWARE* 13
2.1.1 Modelo em Cascata 14
2.1.2 Modelo Evolucionário 15
2.1.3 Modelo Incremental 16
2.2 MODELOS ÁGEIS DE DESENVOLVIMENTO 17
2.2.1 Manifesto Ágil 18
2.2.2 Desenvolvimento Orientado a Testes 20
2.2.3 *Extreme Programming* 21
2.2.4 *Scrum* 23
2.2.4.1 Principais Papéis 25
2.2.4.2 *Product Backlog* 26
2.2.4.3 *Sprint Backlog* 27
2.2.4.4 *Planning Poker* 27
2.2.4.5 *Sprint Burndown Chart* 29
2.2.5 *Kanban* 30
2.3 GERÊNCIA DE PROJETOS 32
2.3.1 PMBOK *GUIDE* 33
2.4 *SOFTWARE* NO APOIO AO GERENCIAMENTO DE PROJETOS 34
2.4.1 *Software* Livre 34
2.4.2 Licenças de *Software* Livre 35
2.4.2.1 GPL 35
2.4.2.2 *Copyleft* 36
2.4.2.3 *Open Source* 36
2.4.2.4 MIT 37
2.4.2.5 AGPL 37
2.4.2.6 BSD 38
2.5 QUALIDADE DE *SOFTWARE* 39
2.5.1 NBR ISO/IEC 9126-1 40
2.5.2 NBR ISO/IEC 14598-1 42
2.5.2.1 Estabelecer os requisitos da avaliação 43
2.5.2.2 Especificar a avaliação 44
2.5.2.3 Projetar a avaliação 45
2.5.2.4 Execução da avaliação 46
3 MATERIAL E MÉTODOS 47
3.1 CLASSIFICAÇÃO DA PESQUISA 47
3.2 REVISÃO DA LITERATURA 47

3.3 O PROCESSO DE AVALIAÇÃO 48
3.4 SELEÇÃO DOS *SOFTWARES* LIVRES 49
3.5 FERRAMENTAS *SCRUM* 50
3.5.1 *Firescrum* 50
3.5.2 *Agilefant* 52
3.5.3 *Kunagi* 53
3.6 QUADRO COMPARATIVO 53
3.7 ANÁLISE DOS REQUISITOS 54
3.8 ESPECIFICAÇÃO DA AVALIAÇÃO 59
3.9 PLANEJAMENTO DA AVALIAÇÃO 65
3.10 EXECUÇÃO DA AVALIAÇÃO 67
4 RESULTADOS 75
5 CONCLUSÃO 78
6 RECOMENDAÇÕES PARA TRABALHOS FUTUROS 80
REFERÊNCIAS 81

1 INTRODUÇÃO

É fato que para uma empresa sobreviver no mercado ela precisa ser flexível, focada no cliente e implementar processos inovadores. Ainda para Santos e Carvalho (2006, apud PEDROSO, 2008), a gestão eficiente de projetos em uma empresa passou a ser utilizada como uma estratégia para ter mais vantagem competitiva perante as demais.

O nível de competitividade de uma empresa é determinado pelo sucesso ou pelo fracasso dos projetos. É na área de projetos, e não nos processos, que as estratégias são transformadas em resultados efetivos, e esta área está cada vez mais valorizada (MUNDOPM, 2008).

O capítulo de Santa Catarina do *Project Management Institute, Project Management Institute Santa Catarina* (2012), cita a definição de projeto dada pelo livro *Project Management Body of Knowledge* (PMBOK). Para ele, projeto é a utilização de um esforço temporário, constituído de uma série de tarefas e atividades e geralmente realizado por uma equipe, para alcançar um objetivo específico. Projetos representam componentes críticos da estratégia de negócios da empresa.

Basicamente, em um projeto de *software*, está envolvida uma equipe que realizará os objetivos do projeto, chamada de equipe de desenvolvimento, uma equipe de gerenciamento, geralmente representada por um gerente de projeto e os *stakeholders*, ou partes interessadas do projeto (KERZNER, 2006).

Para entregar um projeto com um grau maior de acertos, em termos de prazo e custos, é imprescindível escolher corretamente uma metodologia adequada. Uma metodologia adequada possibilita maior controle sobre os recursos que serão utilizados no projeto e faz com que a equipe seja mais eficiente (BARBI, 2012).

Em se tratando de projetos de *software*, mesmo com as constantes evoluções dos métodos, técnicas e ferramentas de desenvolvimento, os prazos e custos previamente estabelecidos nem sempre são cumpridos. As metodologias ágeis para desenvolvimento de *software* surgiram como uma resposta aos problemas encontrados nos métodos tradicionais e pesados (SOARES, 2004b). O modelo em cascata dominou até o início dos anos 90 o desenvolvimento de *software* (SOARES, 2004a). Segundo Boehm (2006, apud MARÇAL *et al.*, 2007) desde o ano 2000 temos uma tendência para o desenvolvimento ágil de aplicações e que no final dos

anos 90 vários métodos ágeis começaram a surgir dentre eles: *Adaptive Software Development, Crystal, Dynamic System Development* (DSDM)*, Extreme Programming (XP), Feature Driven Development e Scrum.*

Em 2001, um grupo de dezessete especialistas em processos de desenvolvimento de *software* criou o Manifesto para o Desenvolvimento Ágil, onde os princípios comuns entre os métodos *Extreme Programming (XP), Scrum,* DSDM*, Crystal* e outros foram unidos (AGILE MANIFESTO, 2012).

Os conceitos deste manifesto são:

- **Indivíduos e iterações** ao invés de processos e ferramentas;
- ***Software* executável** ao invés de documentação;
- **Colaboração do cliente** ao invés de negociação de contratos;
- **Respostas rápidas a mudanças** ao invés de seguir planos.

Uma metodologia ágil deve saber aceitar, avaliar e responder às mudanças ao invés de tentar prever o futuro. As diversas metodologias ágeis existentes compartilham características como: desenvolvimento iterativo e incremental, entrega rápida de *software*, comunicação e redução de documentação extensiva (SOARES, 2004b).

Segundo Soares (2004b), o custo de se alterar requisitos de *software* em várias fases do desenvolvimento, para quem usa uma metodologia em cascata, por exemplo, cresce de forma exponencial de acordo com as fases do projeto, o que faz com que essas alterações não sejam desejáveis neste tipo de modelo. Já no desenvolvimento utilizando uma metodologia ágil o crescimento do custo já se mostra consideravelmente menor ao longo do tempo. Soares (2004b) cita que “as metodologias ágeis incentivam a mudança nos requisitos, pois desta forma é possível realmente entregar ao cliente o produto que ele precisa”.

O *Scrum* é um processo ágil de desenvolvimento que mostra como uma equipe deve funcionar para produzir um *software* flexível em um ambiente em constante mudança. O termo “S*crum”* deriva de uma estratégia do jogo *Rugby*, onde o objetivo é recuperar uma bola perdida através do trabalho em equipe (SCHWABER; BEEDLE, 2002 apud ABRAHAMSSON *et al.*, 2002).

Pesquisa realizada pela *Forrester Research* em 2008, constatou-se que dentre as diversas metodologias ágeis existentes, o *Scrum* representava 84% de utilização nas empresas (SUTHERLAND, 2009).

Rosito, Callegari e Bastos (2007) comentam que, para um gerente de projetos conseguir cumprir objetivos, ele precisa de um suporte de um *software* de gestão, geralmente baseado em uma metodologia de gerência de projetos que permita lidar com as diferentes variáveis de um projeto.

A *Free Software Foundation* (2012), a grosso modo, define *software* livre como sendo aquele onde “os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o software”.

Pesquisa realizada pelo Instituto Sem Fronteiras (ISF) mostrou que o *software* livre está presente em 73% das empresas brasileiras entrevistadas que possuíam mais de mil funcionários e, em média, presente em 56% das empresas sem distinção do porte (PORTAL DA ADMINISTRAÇÃO, 2008).

Diante de todas estas colocações e da existência de diversos *softwares* livres e gratuitos para a gerência de projetos ágeis disponíveis na *internet*, esta pesquisa tem a intenção de elaborar uma avaliação comparativa entre três sistemas livres gratuitos de gestão de projetos de *software* baseado nos princípios da metodologia ágil de desenvolvimento denominada *Scrum*.

Os resultados obtidos servirão de apoio às pessoas ou empresas que futuramente desejem adotar qualquer *software* livre gratuito ou mesmo um dos três avaliados.

1.1 TEMA

Avaliação comparativa de *softwares* livres para suporte ao desenvolvimento ágil baseado no método ágil *Scrum.*

1.2 PROBLEMA

Dentre os *softwares* livres para suporte ao desenvolvimento baseados no método ágil *Scrum* e disponíveis para *download* na *internet*, qual a melhor ferramenta?

Quanto um *software* livre atende aos princípios básicos do *Scrum e aos* requisitos de qualidade de *software*?

1.3 HIPÓTESE

O baixo percentual de uso, em geral, de *software* livre nas empresas, se dá pelo fato de que não existem muitas ferramentas disponíveis para *download* na *internet*, principalmente as voltadas à metodologia ágil *Scrum.*

Algumas dessas ferramentas podem deixar a desejar quanto ao nível de qualidade, utilização de recursos, atualização e atendimento aos princípios básicos a que se destinam.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Elaborar uma avaliação comparativa entre três *softwares* livres de suporte ao gerenciamento ágil baseados no método ágil *Scrum,* disponíveis na *internet*, seguindo as recomendações da norma NBR ISO/IEC 14598-1 e considerando as características de qualidade definidas pela norma NBR ISO/IEC 9126-1.

1.4.2 Objetivos Específicos

- Selecionar os *softwares* livres;
- Definir os requisitos para a avaliação;
- Especificar a avaliação;
- Planejar a avaliação;
- Executar a avaliação;
- Registrar os resultados obtidos;
- Comparar os resultados.

1.5 JUSTIFICATIVA

Atualmente os projetos de desenvolvimento de *softwares* têm seus requisitos alterados constantemente pelos clientes e a utilização de métodos ágeis de desenvolvimento tem sido fundamental para conseguir atender a essa situação.

É fato que para um gerente conseguir cumprir com seus objetivos é preciso do apoio de um *software* adequado de gestão e, quanto menor for o custo para aquisição e/ou manutenção deste *software*, melhor para a organização. Neste ponto encaixam-se os *softwares* livres, pois sua definição já diz que qualquer um pode usar, copiar, modificar e distribuir sem nenhum ônus para a empresa ou para quem for utilizá-los.

Como, em geral, os *softwares* livres ainda estão pouco presentes nas organizações, sem distinção de porte, pretende-se, com este trabalho, contribuir com os gerentes de projetos que buscam ferramentas livres que auxiliem no gerenciamento de suas atividades. Espera-se que, com este trabalho, estes gerentes poupem tempo pesquisando por ferramentas de gestão de desenvolvimento de projetos baseados no método *Scrum* e que este auxilie os mesmos na escolha de uma ferramenta. Também se espera obter um ótimo nível de atendimento nos resultados das avaliações, encorajando assim aqueles que ainda temem em utilizar um *software* livre em sua organização.

1.6 ESTRUTURA DA MONOGRAFIA

A estrutura do trabalho foi feita de forma que, no capítulo 2 encontra-se a revisão da literatura que serviu de apoio ao desenvolvimento de todo o trabalho, contendo alguns modelos tradicionais e ágeis de desenvolvimento de *software*, gerência de projetos, PMBOK, *software* livre e seus principais tipos de licença, qualidade de *software* e as normas NBR ISO/IEC 9126-1 e NBR ISO/IEC 14598-1.

No capítulo 3, denominado material e métodos, encontra-se a classificação deste trabalho e todo o procedimento de avaliação realizado, que vai desde a escolha das ferramentas avaliadas, a lista de procedimentos utilizados e os valores finais obtidos.

No capítulo 4, estão os resultados obtidos da avaliação e, no capítulo 5, as conclusões a respeito.

Por fim, o capítulo 6 possui algumas sugestões para trabalhos futuros referentes ao tema estudado pelo autor.

2 REVISÃO DA LITERATURA

2.1 MODELOS DE PROCESSO DE *SOFTWARE*

Quando vamos realizar uma atividade, seja ela qual for, seguimos uma sequência de tarefas para completar a atividade desejada. Geralmente essas atividades são realizadas na mesma ordem, sempre que executada. Esse conjunto de tarefas ordenadas, envolvendo um conjunto de ferramentas e técnicas, é considerado um processo (PFLEEGER, 2004, p. 36).

Ainda para Pfleeger (2004, p. 37), se o desenvolvimento de um produto é envolvido no processo, ele geralmente é referido como ciclo de vida, ou ciclo de vida do *software*, no caso de desenvolvimento de sistemas.

Um processo de desenvolvimento de *software* é um conjunto de atividades e resultados associados. O resultado deste processo é um produto e este reflete a forma como o processo foi conduzido (SOARES, 2004a).

Em geral, um processo de desenvolvimento de *software* costuma envolver os seguintes estágios:

- Análise e definição de requisitos;
- Projeto do sistema;
- Projeto do programa;
- Programação;
- Testes;
- Entrega do sistema;
- Manutenção.

Todos os processos de desenvolvimento de *software* possuem como início a definição dos requisitos do sistema e como final a entrega do produto prometido e, por consequência a manutenção do mesmo. Um modelo de processo ajuda a equipe envolvida a entender a diferença que existe entre o que deveria ocorrer e o que realmente está ocorrendo. Diversos modelos de processo de *software* foram

sugeridos ao longo dos anos. Neste capítulo serão abordados os modelos em cascata, o evolucionário e o incremental (SOARES, 2004a).

2.1.1 Modelo em Cascata

Para Soares (2004a), o modelo em cascata, ou modelo clássico, foi um dos primeiros modelos propostos. Este modelo dominou a forma de desenvolvimento de sistemas até a década de 90. Como o próprio nome diz, este modelo segue uma sequência de estágios, onde um estágio deve terminar para que o outro possa começar. Neste modelo, somente quando todos os requisitos já tiverem sido levantados, analisados e documentados é que a codificação, propriamente dita, é iniciada pelos programadores, conforme ilustrado na FIGURA 1.

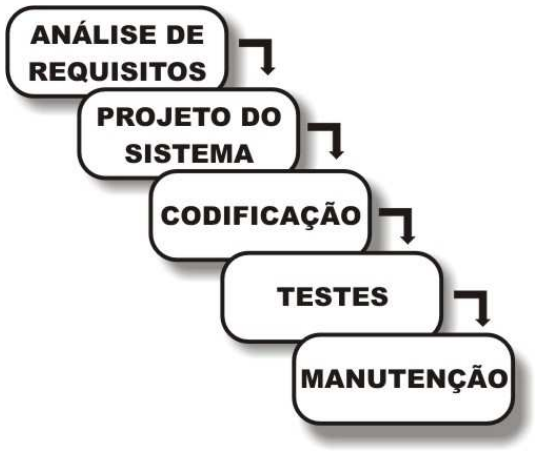


FIGURA 1 – MODELO EM CASCATA
FONTE: Pfleeger (2004)

O problema deste modelo é o fato de ser muito inflexível, o que torna as alterações, comuns em projetos, difíceis de serem aplicadas. Variantes mais complexas do modelo em cascata deram origem a outros modelos mais complexos, onde foram incorporados *loops* de *feedback* e outras atividades extras (SOARES, 2004a).

2.1.2 Modelo Evolucionário

O modelo de desenvolvimento evolucionário tem como principal característica a elaboração de versões do *software* até que se tenha o produto final. Neste modelo, é feita a entrega imediata de uma pequena versão do sistema ao cliente. Após a entrega espera-se o retorno do cliente com as impressões e opiniões que teve (*feedback),* para que seja possível elaborar uma nova versão melhorada, conforme pode ser visto na FIGURA 2 (SOMMERVILLE, 2003).

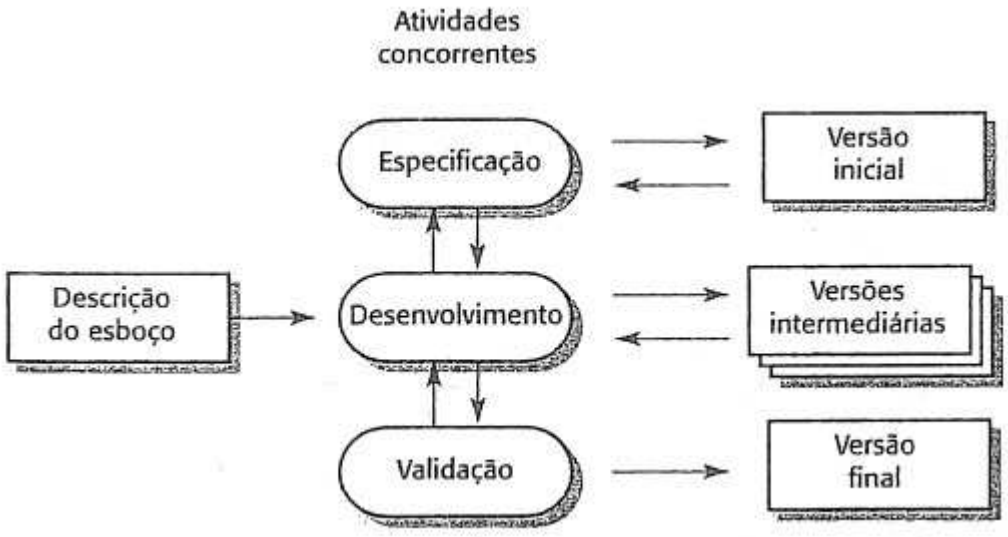


FIGURA 2 – DESENVOLVIMENTO EVOLUCIONÁRIO
FONTE: Sommerville (2003)

Este tipo de desenvolvimento é geralmente utilizado quando os requisitos do projeto são muito complexos para os desenvolvedores. Em projetos de pequeno ou médio porte, o modelo evolucionário tem se mostrado bastante eficiente. Dentre as principais vantagens deste tipo de desenvolvimento está a antecipação do produto final para a avaliação do cliente e a possibilidade de se antecipar um possível treinamento para os usuários. Dentre as desvantagens está a falta de linearidade no progresso do desenvolvimento do sistema e a necessidade de uma ferramenta para o desenvolvimento rápido (SOMMERVILLE, 2003).

2.1.3 Modelo Incremental

O modelo de desenvolvimento incremental é uma união de algumas vantagens dos modelos em cascata e evolucionário. Este tipo de modelo surgiu para tentar diminuir o retrabalho no processo de desenvolvimento de *software* e proporcionar ao cliente a possibilidade de testar primeiramente o sistema para então tomar uma decisão sobre os requisitos, conforme pode ser visto na FIGURA 3 (MILLS *et al*., 1980 apud SOMMERVILLE, 2003).

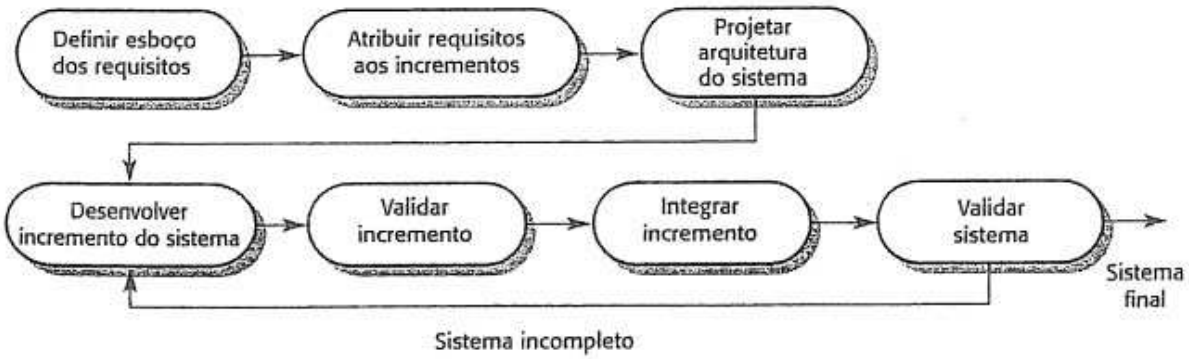


FIGURA 3 – DESENVOLVIMENTO INCREMENTAL
FONTE: Mills *et al*., 1980 apud Sommerville, 2003

Ainda segundo Sommerville (2003, p. 43):

> Em um processo de desenvolvimento incremental, os clientes identificam, em um esboço, as funções a serem fornecidas pelo sistema. Eles identificam quais funções são mais importantes e quais são menos importantes para eles. Em seguida é definida uma série de estágios de entrega, com cada estágio fornecendo um subconjunto das funcionalidades do sistema. A alocação de funções aos estágios depende da prioridade da função. As funções prioritárias são entregues primeiramente ao cliente.

Neste tipo de desenvolvimento têm-se algumas vantagens (SOMMERVILLE, 2003, p. 44):

- O *software* pode ser imediatamente utilizado;
- Os primeiros produtos podem ser considerados protótipos, que dão experiência aos clientes para os próximos requisitos;

- A possibilidade de fracasso é menor;
- Como as funções mais importantes são entregues primeiramente, permite que estas passem por mais fases de testes, dando maior confiabilidade ao produto final.

Embora este modelo tenha suas vantagens, ele também tem suas desvantagens (SOMMERVILLE, 2003, p. 44):

- Os incrementos devem ser relativamente pequenos;
- Cada incremento entregue deve produzir alguma funcionalidade para o sistema;
- Dificuldade em identificar facilidades comuns a todos os incrementos.

2.2 MODELOS ÁGEIS DE DESENVOLVIMENTO

Para que seja possível falar sobre os modelos de desenvolvimento de sistemas considerados ágeis é preciso, primeiramente, definir o que é ser ágil no que diz respeito a esse assunto.

Em uma empresa moderna, as estratégias e as regras de negócio mudam constante e rapidamente. Os interessados no produto final, ou os *stakeholders*, continuam mudando de idéia, até mesmo no último minuto do prazo de entrega, e a gerência exige da equipe respostas quase que imediatas (PRESSMAN; LOWE, 2009). Pressman e Lowe (2009) ainda dizem que “com tudo isso em mente, uma equipe precisa enfatizar a agilidade”.

Jacobson (2002, apud PRESSMAN; LOWE, 2009) enfatiza que “uma equipe ágil é uma equipe ligeira, capaz de responder apropriadamente às mudanças. A mudança é tudo o que o desenvolvimento de *software* é”. Para ele as mudanças podem ser no *software* que está sendo construído, nos membros da equipe, utilização de novas tecnologias ou qualquer outro tipo de mudança que tenha impacto sobre o produto que está sendo construído.

Segundo Soares (2004b), as metodologias ágeis são uma resposta aos modelos pesados ou tradicionais de desenvolvimento de *software* que, diante das mudanças constantes, colocavam em dúvida a qualidade do *software* entregue ou mesmo o cumprimento dos prazos estabelecidos. Segundo ele, o termo “metodologias ágeis” tornou-se popular em 2001 quando um grupo de dezessete especialistas em processos de desenvolvimento de *software* estabeleceu princípios comuns, que eram compartilhados por todos os métodos ágeis de desenvolvimento. Estes princípios deram origem a uma Aliança Ágil e a criação do Manifesto Ágil (AGILE MANIFESTO, 2012).

2.2.1 Manifesto Ágil

Conforme dito na seção anterior, em 2001, um grupo de dezessete especialistas em processos de desenvolvimento de *software* criou o Manifesto para o Desenvolvimento Ágil, onde os princípios comuns entre os métodos *Extreme Programming (XP), Scrum,* DSDM*, Crystal* e outros foram unidos (AGILE MANIFESTO, 2012).

Os conceitos deste manifesto são:

- **Indivíduos e iterações** ao invés de processos e ferramentas;
- ***Software* executável** ao invés de documentação;
- **Colaboração do cliente** ao invés de negociação de contratos;
- **Respostas rápidas a mudanças** ao invés de seguir planos.

A “Aliança Ágil” propõe 12 princípios para que a agilidade seja alcançada (AGILE MANIFESTO, 2012):

1. Priorizar a satisfação do cliente, realizando entregas de *software* em um curto período e de forma contínua;
2. Aceitação de mudanças em qualquer momento do desenvolvimento do *software*;

3. Entrega de *software* funcional de forma frequente em pequenos intervalos de tempo;
4. Interação entre equipe de desenvolvimento e cliente;
5. Os indivíduos da equipe devem estar motivados. É necessário confiança no potencial de cada um e um bom ambiente de trabalho;
6. A melhor forma de disseminar informação entre a equipe e manter uma boa comunicação é um diálogo face-a-face;
7. A primeira medida do progresso é o *software* entregue funcionando;
8. Trabalhar a favor do desenvolvimento sustentável. Os *stakeholders* devem ter a capacidade de manter envolvimento em ritmo de trabalho constante;
9. Maiores cuidados com as boas técnicas de programação e um projeto bem elaborado facilitam a agilidade;
10. Manter a simplicidade, o principal objetivo é o *software* funcionando;
11. As melhores arquiteturas, requisitos e designs são construídas a partir de equipes auto-gerenciáveis;
12. A cada intervalo entre as iterações, a equipe discute a melhor maneira de se tornar mais eficiente, e então adaptações são feitas em seu comportamento.

Diante de tudo isso pode parecer que o Manifesto Ágil rejeita os processos, ferramentas, documentações e negociação de contratos, porém isso não é verdade. Ele mostra que isso tudo tem uma importância secundária perto de indivíduos, iterações, *software* funcionando, colaboração contínua com o cliente e resposta rápida às mudanças (AGILE MANIFESTO, 2012).

2.2.2 Desenvolvimento Orientado a Testes

O Desenvolvimento Orientado a Testes ou o *Test Driven Development* (TDD) consiste em uma técnica, criada por Kent Beck, que se baseia em ciclos curtos de repetições no desenvolvimento do *software* (TEST DRIVEN DEVELOPMENT, 2012).

Nesta técnica, o ciclo de desenvolvimento é bastante simples. A sequência de passos descrita abaixo deve ser repetida até que não se consiga pensar em novos testes. Quando isso acontecer significa que a funcionalidade está pronta (SANCHEZ, 2006).

Segundo Sanchez (2006), o processo de criação de programação orientado a testes segue as seguintes etapas:

1. Escreva um teste que falhe: não há um limite de testes que podem ser feitos. Deve-se pensar nas verificações que precisam ser feitas e o teste deve falhar, pois a funcionalidade ainda não existe;
2. Faça o teste passar: deve-se escrever o mínimo de código que faça o teste passar, sem pensar que o código deve fazer mais coisas;
3. Refatore: se o teste passou, verificar no código o que pode ser melhorado, removendo duplicações, sem comprometer funcionalidades e garantindo que os testes vão continuar passando sem falha.

Para Ribeiro (2010), o TDD é uma prática que visa aumentar a velocidade de entrega de um *software*. Leewis (2004, apud RIBEIRO, 2010) diz que a qualidade não pode ser alcançada através da avaliação de um produto já feito, devem-se prevenir os defeitos ou deficiências em primeiro lugar. Beck (1999, apud RIBEIRO, 2010) complementa citando dois riscos relacionados à qualidade e que podem ser evitados utilizando o TDD: a taxa de defeitos e a deterioração do sistema.

É fato que esta técnica reduz a quantidade de defeitos e constrói e mantém apenas aquilo que é realmente necessário, mas deve-se tomar cuidado, pois somente ela não garante um nível de qualidade no que diz respeito à usabilidade e desempenho, por exemplo (RIBEIRO, 2010).

2.2.3 *Extreme Programming*

A *Extreme Programming* (XP) é uma metodologia ágil, voltada para pequenas ou médias equipes, que precisam desenvolver *softwares* que possuem requisitos poucos claros e constantes mudanças ao longo do seu desenvolvimento (BECK, 2004).

Soares (2004b) diz que a XP “enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas”.

Os seguidores desta metodologia de desenvolvimento são guiados por quatro valores que proporcionam um ambiente agradável de desenvolvimento: (1) comunicação, (2) simplicidade, (3) *feedback* e (4) coragem (BECK, 2004).

O princípio da comunicação visa manter um ótimo relacionamento entre os clientes e os desenvolvedores, e os desenvolvedores e o gerente de projeto. Este princípio estimula que as conversas entre eles devam ser feitas preferencialmente pessoalmente ao invés de utilizar outros meios (SOARES, 2004b).

Soares (2004b) comenta que a simplicidade visa desenvolver um código que contenha a menor quantidade possível de classes e métodos, evitando funções desnecessárias e focando na implementação de requisitos atuais, e não nos futuros. Ele complementa dizendo que “a proposta da XP é que é melhor fazer algo simples hoje e pagar um pouco mais amanhã para fazer modificações necessárias do que implementar algo complicado hoje que talvez não venha a ser usado”.

A idéia geral do *feedback* é que o programador tenha informações constantes, tanto do código quanto do cliente. A informação do código é dada pelos testes constantes, que mostrarão tanto erros individuais quanto do *software* como um todo. A informação do cliente vem da avaliação que eles fazem enquanto usam as partes funcionais do *software* que recebem constantemente. Ele irá sugerir mudanças e novas informações para o desenvolvedor (SOARES, 2004b).

O último valor fala sobre a coragem. Para Soares (2004b), é preciso coragem para que os três valores anteriores sejam aplicados na prática. Nem todas as pessoas possuem facilidade de comunicação. É preciso ter coragem para experimentar uma oportunidade de simplificar uma funcionalidade, quando ela é

percebida, mudando todo o código já feito. Por fim é preciso coragem para conseguir manter um *feedback* constante do cliente.

Os quatro valores são critérios para uma solução de sucesso, mas eles são muito vagos para ajudar a decidir que práticas usar. É preciso transformar esses valores em práticas que possam ser usadas (BECK, 2004). A XP baseia-se em 12 práticas a seguir (SOARES, 2004b):

1. Planejamento: consiste em planejar o que precisa ser feito (prioridades) e o que pode ser adiado no projeto. O cliente é fundamental neste processo e, assim ele fica sabendo o que está sendo feito e o que vai acontecer;
2. Entregas frequentes: entrega de pequenas versões aos clientes contendo os requisitos de maior valor para o negócio. O cliente vai avaliando o que está sendo desenvolvido e dá retorno aos desenvolvedores sobre possíveis modificações. Conforme os requisitos vão sendo alterados, o *software* vai sendo atualizado. Isso aumenta a possibilidade de o *software* final estar de acordo com o que o cliente espera;
3. Metáfora: evitar ou eliminar a utilização de muitos termos técnicos pelo cliente, visando a total compreensão do que ele quer dos desenvolvedores;
4. Projeto simples: o programa desenvolvido deve possuir um código simples (e não fácil) e apenas os requisitos atuais, e não os futuros;
5. Testes: o sistema deve ser validado em todo o processo de desenvolvimento, onde os programadores devem desenvolver o *software* criando primeiramente os testes;
6. Refatoração: o *software* deve ser refeito quando um ou os dois desenvolvedores identificar que é possível simplificar um módulo, mantendo sua funcionalidade, visando o aperfeiçoamento do projeto;
7. Programação em pares: a implementação do código é feita por dois programadores em um micro. Enquanto um escrever o código, o outro observa, procurando identificar erros, e pensa como é possível melhorar o código que está sendo implementado. Esses papéis podem, e devem, ser alterados continuamente;

8. Propriedade coletiva: o código escrito pertence a todos os envolvidos no projeto. Se um membro identifica que pode adicionar algum valor a um código, ele pode fazê-lo, mesmo que não tenha sido ele que o escreveu, desde que realize a bateria de testes que for necessária;
9. Integração: integrar um conjunto de modificações de cada vez, e no mesmo dia em que ela for concluída. A utilização de apenas uma máquina de integração facilita esta prática;
10. 40 horas de trabalho semanal: esta prática tem seu foco nas pessoas, e enfatiza que não se deve fazer horas extras constantemente. Se for necessário fazer horas extras por duas semanas consecutivas é sinal de que existe algum problema sério no projeto, e este deve ser replanejado;
11. Cliente presente: o cliente deve participar do projeto, estar sempre disponível para sanar eventuais dúvidas de requisitos, afim de evitar atrasos;
12. Código padrão: deve-se padronizar a arquitetura do código, para que este possa ser compartilhado entre todos os programadores da equipe.

2.2.4 *Scrum*

A primeira referência que se tem na literatura para o uso do termo "*Scrum"* data do ano de 1986, no artigo de Takeuchi e Nonaka onde um processo de desenvolvimento adaptativo, rápido e auto-organizável, originário do Japão, foi apresentado. O termo "S*crum"* deriva de uma estratégia do jogo *Rugby*, onde o objetivo é recuperar uma bola perdida através do trabalho em equipe. Em se tratando de desenvolvimento de *softwares*, o *Scrum* é um processo ágil de desenvolvimento que mostra como uma equipe deve funcionar para produzir um *software* flexível em um ambiente em constante mudança (SCHWABER; BEEDLE, 2002 apud ABRAHAMSSON *et al.*, 2002).

Ainda referente ao desenvolvimento de *softwares,* o *Scrum* é derivado do processo inventado por Takeuchi e Nonaka e representa um *framework* de desenvolvimento ágil, criado por Jeff Sutherland, que foi também um dos criadores

do Manifesto Ágil, quando trabalhava na empresa *Easel Corporation*, em 1993. Esta metodologia utiliza uma abordagem criativa para o desenvolvimento de *softwares* complexos e inovadores e que envolvem uma grande equipe de desenvolvedores. É utilizado por grandes empresas devido à sua alta produtividade, mesmo com grandes equipes distribuídas ou terceirizadas (SUTHERLAND, 2011).

Scrum é um processo iterativo e incremental para projetos de produtos ou de aplicativos baseado em ciclos chamados de *Sprints*. Estes ciclos não podem demorar mais do que um mês e são realizados em sequência, sem pausa entre eles. Os S*prints* possuem datas definidas para o término, e devem, necessariamente, terminar na data definida, independente de terem sido terminados ou não. A cada início de um S*print*, o membro responsável da equipe prioriza os itens juntamente com o cliente, e estes itens não podem ser alterados durante o S*print*. Todo dia é realizada uma reunião para verificar os progressos e ajustar os passos necessários para completar as tarefas do S*print*. No final do S*print*, a equipe se reúne com os *stakeholders* para mostrar o que foi feito e obter o *feedback*. No caso de desenvolvimento de *softwares*, no final do S*print* somente importa o que realmente foi feito, ou seja, códigos que foram implementados, testados e são potencialmente entregáveis (SUTHERLAND, 2011). O resumo deste processo pode ser visto na FIGURA 4.

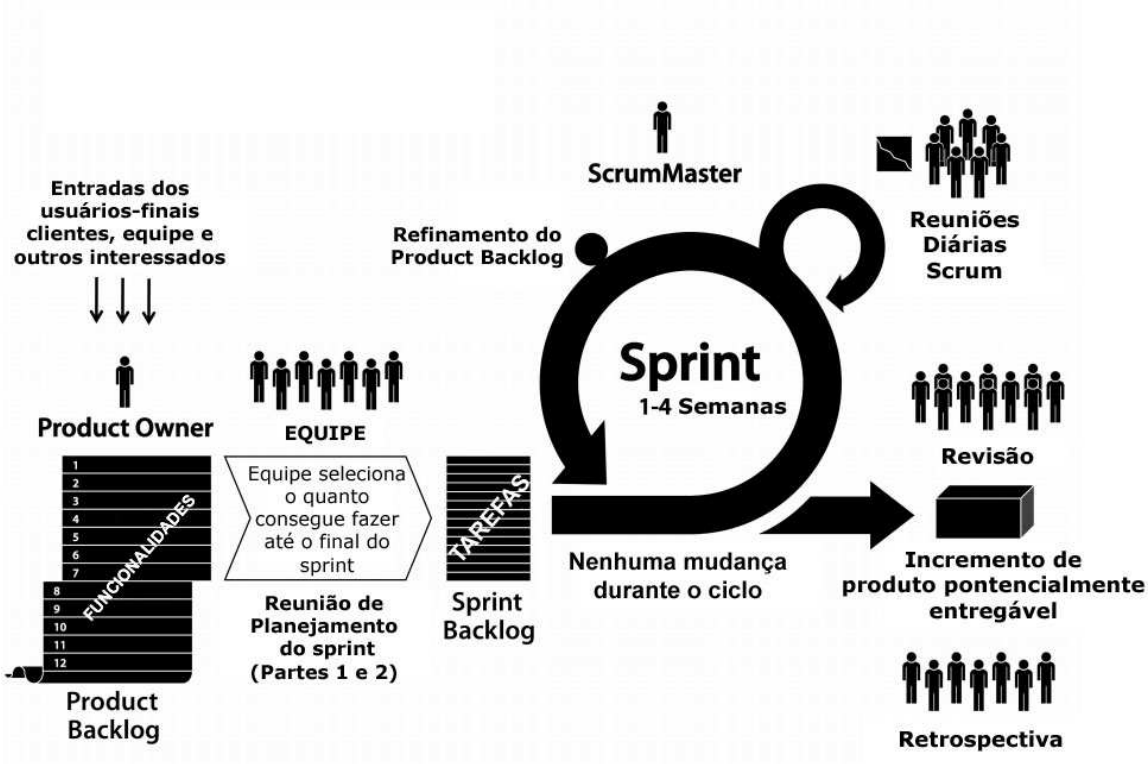


FIGURA 4 – O PROCESSO *SCRUM*

FONTE: Traduzido pelo autor, de Sutherland (2011)

2.2.4.1 Principais Papéis

No *Scrum* existem papéis com diferentes tarefas e responsabilidades, são eles: o *Scrum Master*, o *Product Owner*, equipe *Scrum,* cliente e gerente. As definições para cada um desses papéis são as seguintes, de acordo com Schwaber e Beedle (2002, apud ABRAHAMSSON *et al.*, 2002):

- *Scrum Master*: responsável por verificar se o projeto está sendo conduzido de acordo com as práticas, valores e regras do *Scrum* e se está tudo ocorrendo conforme planejado. Também é responsável por gerenciar os impedimentos ao longo do projeto a fim de manter a equipe produtiva;
- *Product Owner*: selecionado pelo S*crum Master,* cliente e pelo gerente, ele é o responsável pelo controle e gestão do projeto, tornando visível

a lista de atividades do *Product Backlog.* Ele é quem dá a palavra final sobre os itens do *Product Backlog*, principalmente quanto às estimativas de tempo no que diz respeito ao esforço necessário para a realização das atividades;

- Equipe *Scrum*: é a equipe que tem a autoridade para organizar as atividades de cada S*print*, seja na estimativa de tempo necessário no *Sprint Backlog*, ou apenas na priorização das atividades. Também podem sugerir uma lista de impedimentos que devem ser removidos do projeto;
- Cliente: participa na elaboração das tarefas do *Product Backlog* para o *software* que está sendo desenvolvido ou aperfeiçoado;
- Gerente: responsável pela tomada de decisão final, juntamente com os padrões determinados, normas e convenções que devem ser seguidos. Também participa na determinação dos requisitos, seleção do *Product Owner,* confere o progresso e a lista de itens do *Product Backlog* juntamente com o *Scrum Master.*

2.2.4.2 *Product Backlog*

O *Product Backlog* consiste em uma lista de funcionalidades e requisitos a serem desenvolvidos no sistema, contendo o seu tempo estimado de execução para completar aquele item e a prioridade em função da importância para o negócio. As decisões de quais itens irão compor esta lista é feita através de reuniões entre todos os envolvidos, e a ordenação destes itens se dá pela prioridade definida pelo *Product Owner*. No início de cada *Sprint*, os itens do *Product Backlog* podem ser repriorizados, se for necessário, antes de serem transformados em tarefas para o próximo *Sprint* (SUTHERLAND, 2011).

2.2.4.3 *Sprint Backlog*

Antes do início de um *Sprint*, é feita uma reunião chamada de "*Sprint Planning Meeting*", a qual é dividida em duas partes: *Sprint Planning Part One* e *Sprint Planning Part Two*. Na primeira parte da reunião, o *Product Owner* e a Equipe definem a priorização dos itens do *Product Backlog* e selecionam quais itens de maior prioridade entrarão no S*print*. Também repassam qual vai ser a definição de "realizado" para um item. A segunda parte da reunião é focada no detalhamento de como cada item do *Product Backlog*, selecionado para o *Sprint*, será implementado. A equipe decide quanto tempo cada atividade levará para ser executada, baseados em sua própria análise, e o *Product Owner* fará o registro dessas estimativas. Em seguida é feito o cálculo da capacidade de produção da equipe para aquele *Sprint*, somando a quantidade de horas que cada um poderá produzir. Com a capacidade da equipe levantada, eles determinam quantos itens do *Product Backlog* eles conseguem realmente produzir naquele tempo. Os itens do *Product Backlog*, com seus detalhamentos (tarefas) selecionados, de acordo com a capacidade de produção da equipe, é que compõem e definem o *Sprint Backlog* (SUTHERLAND, 2011).

Ainda para Sutherland (2001), uma reunião de planejamento de *Sprint* não pode durar mais do que oito horas para ser realizada, referente a um *Sprint* de quatro semanas.

2.2.4.4 *Planning Poker*

O *Planning Poker* é uma técnica para calcular uma estimativa de esforço ou tamanho relativo de tarefas, baseado no consenso. Muito utilizado no desenvolvimento de *software*s, este método foi descrito pela primeira vez por James Grenning em 2002 e foi popularizado por Mike Cohn mais tarde. O *Planning Poker* geralmente utiliza uma série de cartas numeradas. Uma sequência bastante usada é a sequência de *Fibonacci* incluindo o zero (0, 1, 2, 3, 5, 8, 13, 21, etc). Existem outros tipos comerciais de cartas que utilizam outros padrões (FIGURA 5), algumas,

inclusive, contendo uma carta com uma xícara desenhada, indicando que se esta carta sair está na hora de uma pausa para o café (PLANNING POKER, 2012).

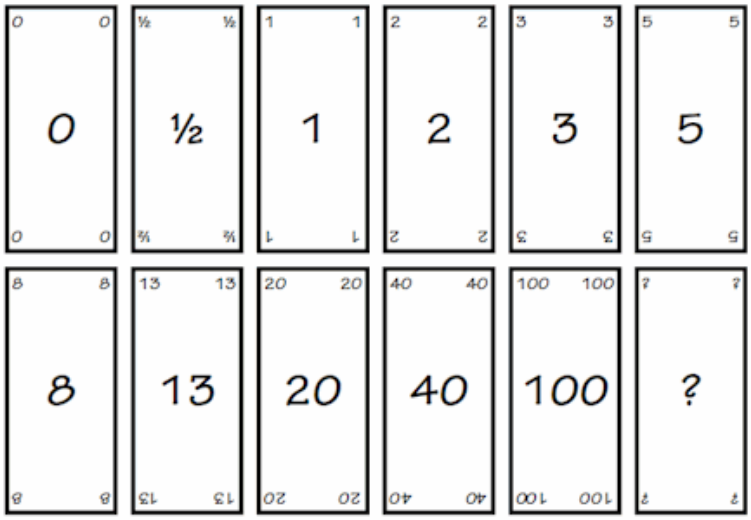


FIGURA 5 – CARTAS DO *PLANNING POKER*
FONTE: Agile Planning Poker | View from the Potting Shed (2012)

Seu funcionamento se dá da seguinte forma (PLANNING POKER, 2012):

1. Um moderador, que não participa do "jogo", atua como mediador;
2. Ele pega um item da lista, apresenta para a equipe e dá a oportunidade para que todos discutam a respeito e tirem suas dúvidas;
3. Cada membro pega uma carta contendo o valor que corresponde ao esforço (que pode ser em dias, horas, meses, etc.) que imagina ser necessário para a realização daquela atividade. Coloca a carta escolhida sobre a mesa, com a face contendo o número voltado para baixo, sem mostrar para os outros participantes;
4. Quando todos selecionarem a sua carta, o mediador pede para que virem e mostrem a carta com o número selecionado simultaneamente;
5. A pessoa com a carta contendo a menor numeração e a que está com a carta com maior numeração são solicitadas para que dêem sua justificativa para o que estimaram. O moderador toma nota das observações e, se achar necessário, faz nova rodada de cartas (repete o passo 3);

6. Os passos de 3 a 5 se repetem até que a grande maioria das estimativas esteja próxima ou que ele ache necessário finalizar. Se o resultado for, por exemplo, 5, 5, 5 e 3. Ele pergunta ao participante que colocou o valor 3 se está de acordo em colocar o valor 5 na estimativa e dá por encerrada aquela sessão de *Planning Poker* para aquela atividade.

A razão para se utilizar o *Planning Poker* é porque ele evita a influência dos outros participantes e força a independência de cada um na decisão. Se um participante fala qual é sua estimativa, isso pode intimidar quem tem uma opinião totalmente diferente e soa como uma sugestão para quem está na dúvida de quanto falar. Estudo realizado por K. Molokken-Ostvold e N.C. Haugen, mostrou que as estimativas obtidas através do *Planning Poker* eram menos otimistas e mais precisas do que as obtidas pela combinação mecânica de estimativas individuais (PLANNING POKER, 2012).

2.2.4.5 *Sprint Burndown Chart*

O *Sprint Burndown Chart* consiste em um gráfico que mostra, diariamente, o quanto do trabalho foi feito e quanto ainda falta ser feito até que o *Sprint Backlog* seja finalizado. Geralmente medido em horas/dia e com uma reta inclinada para baixo, da esquerda para a direita, ele representa o esforço que ainda é necessário para concluir as atividades e tende a ter o valor final zero, no último dia do *Sprint*. As informações deste gráfico são alimentadas diariamente pelos membros da equipe e, conforme pode ser visto na FIGURA 6, ele contém duas linhas, que demonstram o esforço estimado (proveniente das reuniões *Sprint Planning Meeting*) *versus* o esforço realmente gasto, e auxilia a equipe a saber como está indo em relação ao objetivo final e se serão necessários ajustes para que o objetivo seja cumprido (SUTHERLAND, 2011).

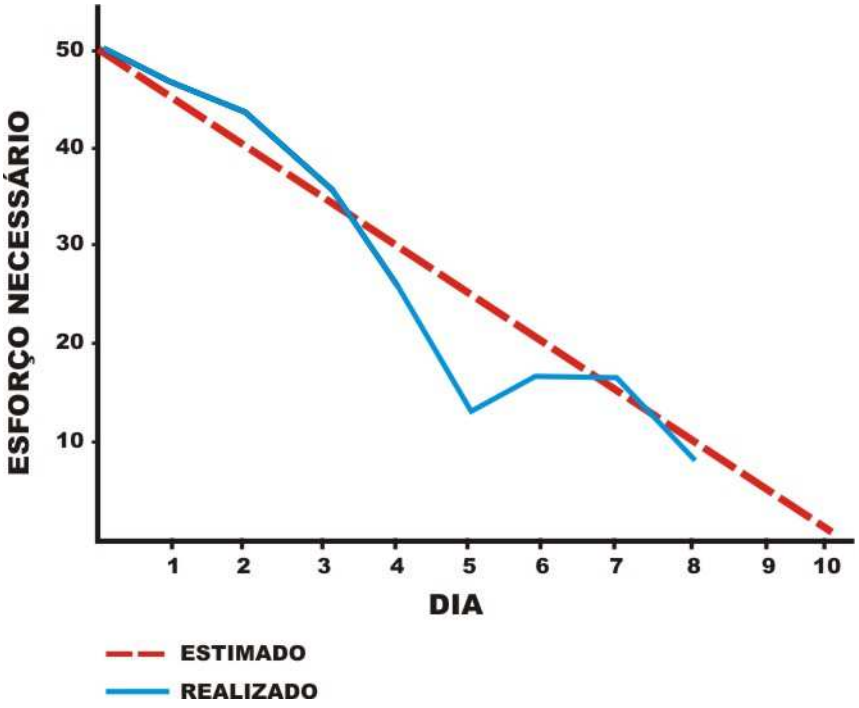


FIGURA 6 – EXEMPLO DE UM *SPRINT BURNDOWN CHART*
FONTE: O autor (2012)

2.2.5 *Kanban*

Literalmente, *Kanban* é uma palavra japonesa que significa “cartões visuais”. Originário do sistema de produção da Toyota, representava um dispositivo físico e visual que orientava a movimentação ou a criação das peças no sistema de produção. O uso de *Kanban* na Engenharia de Produção de *Softwares* tem crescido desde 2003, e tem David Anderson como principal promotor dessa técnica. Em se tratando de desenvolvimento de *softwares*, *Kanban* representa uma abordagem enxuta de desenvolvimento ágil (HEPTAGON, 2012).

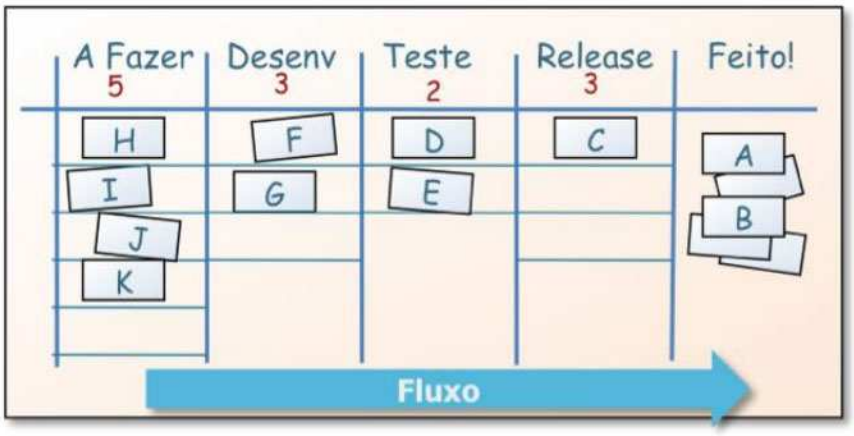


FIGURA 7 – QUADRO *KANBAN*
FONTE: Kniberg; Skarin (2009)

Como ferramenta visual, o *Kanban* é bem simples. Nesta técnica, todo o trabalho deve ser dividido em partes e cada parte escrita em um cartão. Deve-se nomear cada coluna de acordo com cada etapa do trabalho. À medida que uma etapa de uma parte do trabalho for sendo concluída, deve-se passar o cartão para uma etapa seguinte, conforme pode ser visto na FIGURA 7. Deve-se associar limites de quantidade de cartões (tarefas) para cada estado do fluxo, ou seja, limitar quantos itens podem estar naquela etapa. Este limite recebe o nome de *Work in Progress* (WIP). Com o *Kanban* pode-se acompanhar o tempo médio que um item leva para completar uma etapa ou todo o ciclo (tempo de ciclo), servindo de auxílio para quem desejar otimizar o processo de produção, seja fazendo previsões ou reduzindo o tempo de produção (KNIBERG; SKARIN, 2008).

Em desenvolvimento de *softwares,* esta técnica resume-se a limitar a quantidade de tarefas para os desenvolvedores (WIP), evitando a sobrecarga e o acúmulo de produtos inacabados. O ritmo de produção é ditado pelo andamento da fila de tarefas, onde à medida que uma tarefa é concluída, outra é adicionada, de acordo com os limites pré-estabelecidos para aquela etapa (HEPTAGON, 2012).

Ainda para Heptagon (2012), *Kanban* funciona muito bem com qualquer método tradicional de desenvolvimento de *software,* mas complementa muito bem as abordagens ágeis de desenvolvimento, como *Scrum* e XP. Heptagon (2012) enfatiza que com “o WIP também diminuímos a multitarefa nociva, principal responsável por atrasos, problemas de qualidade, estresse e insatisfação” e com uma sobrecarga

menor em todo o sistema, é melhor e mais ágil na adaptação das mudanças de demanda dos clientes.

2.3 GERÊNCIA DE PROJETOS

Para falar sobre gerência de projetos é preciso primeiramente definir o que é um projeto. Para Kerzner (2004), um projeto é “um empreendimento com um objetivo bem definido, que consome recursos e opera sob pressões de prazos, custos e qualidade”.

Bruzzi (2008, p. 15) define projeto como sendo:

> Um empreendimento não repetitivo, caracterizado por uma sequência clara e lógica de eventos, com início, meio e fim, que se destina a atingir um objetivo claro e definido, sendo conduzido por pessoas dentro de parâmetros pré-definidos de tempo, custo e recursos envolvidos com qualidade.

Ainda para Bruzzi (2008), o gerenciamento de projetos pode ser definido como o “planejamento, programação e controle das atividades do referido projeto para atingir os seus objetivos”.

Santos e Carvalho (2006, apud PEDROSO, 2008) comentam que a gestão eficiente de projetos em uma empresa passou a ser utilizada como uma estratégia para ter mais vantagem competitiva perante as demais.

O nível de competitividade de uma empresa é determinado pelo sucesso ou pelo fracasso dos projetos. É na área de projetos, e não nos processos, que as estratégias são transformadas em resultados efetivos, e esta área está cada vez mais valorizada (MUNDOPM, 2008).

O *Project Management Institute Santa Catarina* (2012) enfatiza a importância dos projetos dizendo que “os projetos representam componentes críticos da estratégia de negócios da empresa”.

O desenvolvimento de produtos de *software* exige o planejamento e execução de atividades definidas no escopo do projeto, onde é necessário unir a gestão às questões técnicas. Em desenvolvimento de *software*, gerência de projetos envolve

pessoas e outros recursos com o objetivo de planejar, analisar, projetar, construir, testar e manter um *software* (ROSITO; CALLEGARI; BASTOS, 2012).

Basicamente, em um projeto de *software* está envolvida uma equipe que realizará os objetivos do projeto, chamada de equipe de desenvolvimento, uma equipe de gerenciamento, geralmente representada por um gerente de projeto, e os *stakeholders*, ou partes interessadas do projeto (KERZNER, 2006).

Todo projeto tem como principais fatores a qualidade, o custo e o tempo. O escopo do projeto é que determina o que será ou não atendido e seus limites. É aconselhável que se atribua pesos e importâncias para cada um desses itens (BRUZZI, 2008).

Para Soares (2004b), em se tratando de projetos de *software*, mesmo com as constantes evoluções dos métodos, técnicas e ferramentas de desenvolvimento, os prazos e custos previamente estabelecidos nem sempre são cumpridos.

Em 1983, um guia destinado a dar suporte aos profissionais de gerência de projetos foi publicado, o *Project Management Body of Knowledge*. Desde então, este guia vem sendo atualizado periodicamente, sendo a última, na quarta edição, publicada em 2008 (PROJECT MANAGEMENT INSTITUTE, 2012), conforme pode ser visto na seção seguinte.

2.3.1 PMBOK *GUIDE*

Segundo o *Project Management Institute* (2012), publicado originalmente em 1983, o *Project Management Body of Knowledge* (PMBOK), ou PMBOK *GUIDE*, como é mundialmente conhecido, reúne as melhores práticas que podem ser aplicadas à maioria dos projetos, sejam eles de *softwares* ou não. Atualmente está em sua quarta edição e já foi traduzido para diversos idiomas, sendo eles: o árabe, chinês, francês, alemão, italiano, japonês, coreano, português, russo e espanhol. O PMBOK *GUIDE* não determina quais ações devem ser tomadas e nem como elas devem ser executadas, ele apenas identifica subconjuntos genéricos do conhecimento sobre gerência de projetos e constrói uma coleção de boas práticas (ROSITO; CALLEGARI; BASTOS, 2012).

Para que o cumprimento dessas boas práticas seja possível, é fortemente recomendável que o gerente de projetos utilize algum tipo de suporte baseado em uma metodologia de gerência de projetos como, por exemplo, um *software* de gestão de projetos (ROSITO; CALLEGARI; BASTOS, 2012).

2.4 *SOFTWARE* NO APOIO AO GERENCIAMENTO DE PROJETOS

Lacy (2006, p. 46) comenta que aplicativos voltados ao gerenciamento de projetos existem há cerca de 50 anos. Para ele, desde 2006 o foco desses aplicativos está na colaboração dos envolvidos e não mais nos cálculos sofisticados. Ele complementa dizendo que “para situações onde a quantidade de projetos é maior, a quantidade de pessoas envolvidas é grande ou a dinâmica de trabalho é elevada, é necessário o uso de ferramentas especiais para isso”, ou seja, a utilização de um aplicativo voltado para esta finalidade e que rode em um computador, seja ele um sistema na *web* ou não.

Ainda segundo Lacy (2006, p. 46), a tendência é a utilização de *softwares* na *web*, ou seja, que são acessados utilizando um navegador de *internet*.

Pesquisa realizada pelo Instituto Sem Fronteiras (ISF) mostrou que o *software* livre está presente em 73% das empresas entrevistadas que possuíam mais de mil funcionários e em média, presente em 56% das empresas sem distinção de porte (PORTAL DA ADMINISTRAÇÃO, 2008).

2.4.1 *Software* Livre

Desde a década de 60, o desenvolvimento de *softwares* livres tem sido contínuo, com o período de maior produtividade começando no ano de 1992 até o presente momento (HEXSEL, 2002).

A *Free Software Foundation* (2012), a grosso modo, define *software* livre como sendo aquele onde “os usuários possuem a liberdade de executar, copiar,

distribuir, estudar, mudar e melhorar o software". Outra definição é a de Hexsel (2002, p. 2), que define *software* livre como sendo:

> Um *software* cujo autor o distribui e outorga a todos a liberdade de uso, cópia, alteração e redistribuição de sua obra. A liberdade de uso e alteração é somente viabilizada pela distribuição dos programas na forma de texto legível por humanos, isto é, com seu código-fonte, bem como no formato executável por um computador. Além do código-fonte, o autor do programa outorga a liberdade para que outros programadores possam modificar o código original e redistribuir versões modificadas.

Embora Lacy (2006) relate ser difícil encontrar um *software* livre que atenda "de verdade" um gerente de projetos, se for feita uma busca por "*software* livre para gerência de projetos", na *internet,* muitas páginas com opções serão mostradas. Prova disso é o resultado da pesquisa feita pelo Instituto Sem Fronteiras no ano de 2008, onde revelou que em 73% das grandes empresas (com mais de mil funcionários) utilizavam *softwares* livres.

Quando um autor disponibiliza seu programa, ele pode escolher o "grau de liberdade" que deseja dar ao seu programa, limitando assim a forma com que modificações e redistribuições podem ser realizadas. Estas definições de liberdade constituem as licenças de *software* (LACY, 2006).

2.4.2 Licenças de *Software* Livre

Existem diversos tipos de licença para a distribuição de *software* livre, que se distinguem pelo grau de liberdade que dá ao usuário. Neste trabalho apenas as licenças GPL, *copyleft*, *Open Source*, MIT, AGPL e BSD serão explanadas.

2.4.2.1 GPL

Segundo Hexsel (2002, p. 5), a Licença Pública Geral GNU (*GNU General Public License – GPL)* é:

> Uma licença que acompanha os pacotes distribuídos pelo Projeto GNU, e mais uma grande variedade de *software*, incluindo o núcleo do sistema operacional *Linux*. A formulação da GPL é tal que ao invés de limitar a distribuição do *software* por ela protegido, ela de fato impede que este *software* seja integrado em *software* proprietário.

Segundo GNU GENERAL PUBLIC LICENSE (2012), a licença GPL é a mais pura designação de *software* livre, onde é permitido:

- Executar o programa para qualquer propósito;
- Estudar o funcionamento do programa e adaptá-lo à sua necessidade, onde o acesso ao código fonte é um pré-requisito;
- Redistribuir cópias para ajudar ao próximo;
- Aperfeiçoar o programa e liberar seus aperfeiçoamentos, onde o acesso ao código-fonte também é um pré-requisito.

2.4.2.2 *Copyleft*

Segundo GNU Project (2012), a licença *copyleft* é basicamente uma variação do *copyright* (direitos autorais) do programa. Nesta modalidade de licença qualquer um pode distribuir o *software*, com ou sem modificações, mas seu termo de distribuição diz que para isso é necessário que a pessoa que o redistribui passe a diante a liberdade de copiar e modificar o programa novamente, impedindo assim que este programa seja transformado em um *software* proprietário, ou seja, aquele onde a cópia, modificação ou redistribuição são proibidas pelo autor.

2.4.2.3 *Open Source*

A licença *Debian,* ou *Open Source,* é basicamente o fruto de um contrato social entre a *Debian* e a comunidade de usuários de *software* livre. Os critérios para distribuição desta licença são (HEXSEL, 2002):

- A redistribuição deve ser livre;
- O código-fonte deve ser incluído e poder ser redistribuído;
- Versões derivadas podem ser redistribuídas sob a mesma licença original;
- Pode haver restrições quanto à redistribuição do código-fonte, se o original foi modificado;
- Não pode haver discriminação contra qualquer pessoa, grupo e nem forma de utilização do *software;*
- Os direitos outorgados não podem depender da distribuição onde o *software* se encontra;
- A licença não pode “contaminar” outro *software.*

2.4.2.4 MIT

Segundo MIT License (2012), a licença MIT originou-se do *Massachusetts Institute of Technology* (MIT). Neste tipo de licença é permitido, gratuitamente a qualquer pessoa que obtenha uma cópia do *software*, utilizá-lo sem restrição, podendo usar, copiar, modificar, redistribuir, sublicenciar ou vender cópias do *software*. No entanto, exige que todas as cópias ou partes substanciais distribuídas devem conter um aviso de *copyright* e de permissão especificado por esta licença (OPEN SOURCE INITIATIVE, 2012).

2.4.2.5 AGPL

A licença GNU *Affero General Public License* (AGPL) é uma licença GPL ligeiramente modificada. Esta licença é voltada especificamente para *softwares* que funcionam como serviços em uma rede, e exige que o responsável pelo servidor de rede, onde está funcionando o *software*, disponibilize o código fonte das versões modificadas aos usuários deste servidor. Esta licença é comumente usada em aplicações *web*, ou seja, que rodam atrás de um servidor (GNU PROJECT, 2012).

2.4.2.6 BSD

A licença *Berkeley Software Distribution* (BSD) impõe poucas restrições sobre a forma de uso, alteração e redistribuição do *software* licenciado. O *software* pode ser transformado em proprietário, vendido, e não exige inclusão do código fonte. Esta licença garante que o crédito de autoria será mantido, mas não garante que ele seja mantido como um *software* livre (HEXSEL, 2002).

2.4.3 Vantagens de um *software livre*

Hexsel (2002) cita algumas vantagens decorrentes da utilização de um *software* livre:

- Custo social é baixo;
- Não se fica refém de tecnologia proprietária;
- Independência de fornecedor único;
- Desembolso inicial próximo de zero;
- Não obsolescência do hardware;
- Robustez e segurança;
- Possibilidade de adequar aplicativos e redistribuir versão alterada;
- Suporte abundante e gratuito;
- Sistemas e aplicativos geralmente muito configuráveis.

2.4.4 Desvantagens de um *software* livre

Hexsel (2002) também cita algumas desvantagens em relação ao uso de *software* livre:

- *Interface* de usuário não uniforme nos aplicativos;
- Instalação e configuração difícil;
- Mão-de-obra escassa e/ou custosa para desenvolvimento e/ou suporte.

2.5 QUALIDADE DE *SOFTWARE*

Para poder falar sobre qualidade de *software* convém primeiramente falar sobre três organismos normalizadores, com importância internacional e nacional no setor de *software*: a *International Organization for Standardization* (ISO), a *International Electrotechnical Commission* (IEC) e a Associação Brasileira de Normas Técnicas (ABNT).

A *International Organization for Standardization* é uma organização não governamental, criada em 1947, que promove a padronização de normas técnicas no âmbito mundial. Ela coordena o trabalho de 163 países e possui escritório central em Genebra, na Suíça, que coordena todo o sistema. Devido ao fato de a *International Organization for Standardization* possuir diferentes acrônimos nos diversos países, foi adotada a sigla ISO como padrão internacional para esta organização (ISO, 2012).

A *International Electrotechnical Commission* é uma comissão, fundada em 1906, que publica normas internacionais relacionadas com eletricidade, eletrônica e áreas correlatas. O acrônimo padrão internacional adotado para este organismo foi o IEC. Quando necessário, a IEC trabalha em conjunto com a ISO para garantir que as normas internacionais se encaixem perfeitamente e se completem (IEC, 2012).

A Associação Brasileira de Normas Técnicas é uma entidade privada, sem fins lucrativos, fundada em 1940. É o órgão responsável pela normalização técnica

no Brasil. A ABNT é o representante oficial no Brasil das entidades ISO e IEC, dentre outras (ABNT, 2012).

Segundo a ABNT (2012), o trabalho conjunto desses três organismos deu origem a diversas normas, que são utilizadas no mundo inteiro atualmente. Duas delas, relacionadas à qualidade de *software,* são utilizadas neste trabalho: a NBR ISO/IEC 9126-1 e a NBR ISO/IEC 14598-1. Ambas serão detalhadas nas duas seções seguintes.

A norma NBR ISO/IEC 9126-1, publicada em 2003, define qualidade de *software* como sendo a “totalidade de características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003).

Necessidades explícitas são condições e objetivos propostos por aqueles que produzem o *software*, e são percebidos somente pelas pessoas que trabalham no seu desenvolvimento. Necessidades implícitas são necessidades subjetivas dos usuários, e podem ser percebidas tanto pelos desenvolvedores quanto pelos usuários (GOMES, 2000).

Ainda segundo Gomes (2000), medir a qualidade de um *software* é o principal problema com que se defronta a engenharia de *software.* A qualidade de um dispositivo mecânico geralmente é medida em função da capacidade em suportar desgaste. Um *software* não se desgasta e esse tipo de medição não pode ser aplicado nesse caso. É nesse ponto que entra a importância da parte 1 da NBR ISO/IEC 9126, conforme pode ser visto na próxima seção.

2.5.1 NBR ISO/IEC 9126-1

Desde 1976 diversas pessoas tentaram definir uma estrutura de qualidade de *software.* Por um longo período, a confiabilidade foi o único meio de certificar a qualidade de um *software*. Diversos outros modelos de qualidade foram propostos e utilizados. Embora muitos modelos tenham sido usados com sucesso, eles também causaram confusão por oferecer muitos aspectos de qualidade. Este cenário fez surgir a necessidade de um modelo padronizado de qualidade de *software* (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003).

Por essa razão o comitê técnico da NBR ISO/IEC 9126-1 começou a desenvolver este padrão que era requerido mundialmente e, em 1978, surgiu o primeiro modelo da NBR ISO/IEC 9126-1. A NBR ISO/IEC 9126-1 define um modelo de qualidade com seis características para produtos de *software*, que permite avaliar a qualidade de um produto de *software* utilizando algum método quantitativo (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003).

As seis características, definidas pela NBR ISO/IEC 9126-1, que compõem o modelo de qualidade externa e interna são (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003):

1. Funcionalidade;
2. Confiabilidade;
3. Usabilidade;
4. Eficiência;
5. Manutenibilidade;
6. Portabilidade.

Cada característica definida é subdividida em subcaracterísticas, que podem ser medidas por meio de métricas internas ou externas (TABELA 1).

TABELA 1 – SUBCARACTERÍSTICAS DA NBR ISO/IEC 9126-1

CARACTERÍSTICA	SUBCARACTERÍSTICA	SIGNIFICADO
Funcionalidade	Adequação	Propõe-se a fazer o que é apropriado?
	Acurácia	Gera resultados corretos ou conforme acordados?
	Interoperabilidade	É capaz de interagir com os sistemas especificados?
	Segurança de acesso	Evita o acesso não autorizado, acidental ou deliberado a programas e dados?
	Conformidade	Está de acordo com normas e convenções previstas em leis e descrições similares?
Confiabilidade	Maturidade	Com que frequência apresenta falhas?
	Tolerância a falhas	Ocorrendo falhas, como ele reage?
	Recuperabilidade	É capaz de recuperar dados após uma falha?
Usabilidade	Inteligibilidade	É fácil entender os conceitos utilizados?

CARACTERÍSTICA	SUBCARACTERÍSTICA	SIGNIFICADO
	Apreensibilidade	É fácil aprender a usar?
	Operacionalidade	É fácil de operar e controlar a operação?
Eficiência	Comportamento em relação ao tempo	Qual é o tempo de resposta e processamento?
	Comportamento em relação aos recursos	Quanto recurso utiliza?
Manutenibilidade	Analisabilidade	É fácil encontrar uma falha quando ocorre?
	Modificabilidade	É fácil modificar e remover defeitos?
	Estabilidade	Há grandes riscos de *bugs* quando se faz alterações?
	Testabilidade	É fácil testar quando se faz alterações?
Portabilidade	Adaptabilidade	É fácil adaptar a outros ambientes sem aplicar outras ações ou meios além dos fornecidos para esta finalidade no *software* considerado?
	Capacidade de ser instalado	É fácil instalar em outros ambientes?
	Capacidade de substituir	É fácil substituir por outro *software*?
	Conformidade	Está de acordo com os padrões ou convenções de portabilidade?

FONTE: GOMES (2000)

Gomes (2000, p. 4) complementa dizendo que “o modelo proposto pela NBR ISO/IEC 9126-1 tem por objetivo servir de referência básica na avaliação de produto de *software*. Além de ter força de norma internacional, ela cobre os aspectos mais importantes para qualquer produto de *software*”.

2.5.2 NBR ISO/IEC 14598-1

Publicada em 2001, esta parte da NBR ISO/IEC 14598-1 fornece requisitos e recomendações para implementação prática de avaliação de um produto de *software*, quando várias partes envolvidas necessitam entender, aceitar e confiar nos resultados da avaliação. Em especial, ela pode ser usada na aplicação dos conceitos descritos na NBR ISO/IEC 9126-1. O processo de avaliação, descrito na

NBR ISO/IEC 14598-1, descreve 4 etapas que compõem este processo (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001):

1. Estabelecer os requisitos da avaliação;
2. Especificar a avaliação;
3. Projetar a avaliação;
4. Executar a avaliação.

2.5.2.1 Estabelecer os requisitos da avaliação

Na primeira etapa, ou seja, no estabelecimento dos requisitos da avaliação, deve-se estabelecer o propósito da avaliação, os produtos a serem avaliados e especificar o modelo de qualidade (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

Segundo as definições da Associação Brasileira de Normas Técnicas (2001), estabelecer o propósito da avaliação é assegurar que o produto forneça a qualidade requerida e que ele atenda às necessidades implícitas e explícitas dos usuários. A identificação do tipo de produto a ser avaliado refere-se à determinação de qual produto será avaliado, o intermediário ou final, e se uma parte em especifico ou ele como um todo.

Ainda segundo a Associação Brasileira de Normas Técnicas (2001), especificar o modelo de qualidade consiste em “selecionar características de qualidade relevantes, utilizando um modelo de qualidade que desdobre a qualidade do *software* em diferentes características”. A NBR ISO/IEC 9126-1 fornece um modelo de qualidade com seis amplas categorias de características, conforme pode ser visto na seção 2.5.1 deste trabalho. Estas características devem ser subdivididas em subcaracterísticas com atributos mensuráveis. A NBR ISO/IEC 9126-1 fornece esta lista de verificação para um modelo de qualidade, no entanto a NBR ISO/IEC 14598-1 diz que outras maneiras de categorizar a qualidade podem ser utilizadas, dependendo das circunstâncias (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

2.5.2.2 Especificar a avaliação

Nesta segunda etapa, ou seja, na especificação da avaliação, deve-se selecionar as métricas, estabelecer níveis de pontuação e os critérios para julgamento dos resultados. Para selecionar as métricas, a NBR ISO/IEC 14598-1 diz que “é importante que as medições de um produto de *software* possam ser feitas fácil e economicamente e que as medidas resultantes sejam fáceis de usar”. Para a norma, todo atributo interno e externo quantificável interagindo com seu ambiente e que se correlacione com uma característica pode ser definido como uma métrica. Também define uma escala quantitativa e um método que podem ser usados para medição (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

As métricas podem ser externas ou internas, diretas ou indiretas. As externas representam a perspectiva externa de qualidade do *software* quando está em uso, já as métricas internas representam atributos internos relacionados ao projeto e ao código fonte. Métricas diretas estão relacionadas a números, como por exemplo número de campos em uma tela, já a métrica indireta é derivada de medidas de outra ou de várias outras métricas (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

Para estabelecer os níveis de pontuação, a norma define que se deve atribuir uma escala e uma pontuação correspondente, pois um número por si só não mostra o nível de satisfação (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

Quanto ao estabelecimento dos critérios de julgamento, Villas Boas (2007, p. 27) diz que:

> Para julgar a qualidade do produto, o resultado da avaliação de cada característica precisa ser sintetizado. Convém que o avaliador prepare um procedimento para isto, com critérios diferentes para características de qualidade diferentes, onde cada característica poderá estar representada em termos de suas subcaracterísticas ou de uma combinação ponderada de subcaracterísticas. O procedimento normalmente incluirá outros aspectos como tempo e custo, os quais contribuem para o julgamento da qualidade de um produto de *software* num ambiente particular.

Villas Boas (2007) sugere um método para obter resultados sintéticos de avaliação, composto por três etapas:

1. Atribuir aos resultados das métricas uma escala que vai de 0 (zero) a 1 (um), onde 0 significa o pior resultado e 1 significa o melhor;
2. Estabelecer pesos para as subcaracterísticas de qualidade, onde o peso deve representar a importância relativa de cada item no julgamento final de qualidade do produto;
3. Calcular médias ponderadas utilizando os valores das métricas e os pesos de cada subcaracterística.

Dessa forma é possível, por exemplo, calcular médias agrupando as métricas de cada característica, como por exemplo obter a média ponderada apenas da característica usabilidade. Este procedimento também permite confrontar diretamente produtos, no caso de avaliações comparativas, e examinar os resultados em diferentes níveis de detalhe. Mas a comparação somente é possível quando as métricas e os pesos, utilizados na avaliação, forem idênticos (VILLAS BOAS, 2007).

A Associação Brasileira de Normas Técnicas (2001), na norma NBR ISO/IEC 14598-1, define que, para julgar a qualidade do produto, o resultado da avaliação de cada característica precisa ser sintetizado, preparando um procedimento com diferentes critérios para características de qualidades diferentes.

2.5.2.3 Projetar a avaliação

Nesta terceira etapa, ou seja, no projeto da avaliação, deve-se, com base na especificação da avaliação, elaborar um plano de avaliação contendo os componentes do produto de *software* e os métodos de avaliação propostos.

O plano de avaliação deve conter o método de avaliação e o cronograma das ações que o avaliador deverá executar, contemplando, se possível, uma previsão do tempo necessário para a realização da avaliação (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

2.5.2.4 Execução da avaliação

Nesta quarta etapa, ou seja, na execução da avaliação, deve-se realizar a inspeção, medição e teste dos produtos de acordo com o plano de avaliação e, por fim, comparar e julgar os resultados. É nesta etapa que o avaliador vai obter as medidas, e as métricas selecionadas serão aplicadas ao *software* obtendo-se, como resultado, os valores nas escalas das métricas. Posteriormente deve-se comparar os valores medidos com os critérios predeterminados (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

O julgamento dos resultados constitui a etapa final do processo de avaliação. O resultado é quanto o produto de *software* atende aos requisitos de qualidade e o resultado resumido pode ser comparado a outros aspectos como tempo e custo, servindo de apoio a uma decisão gerencial, se for o caso (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2001).

Com base em todo esse referencial teórico foi possível dar início ao processo de avaliação da qualidade dos *softwares*.

3 MATERIAL E MÉTODOS

3.1 CLASSIFICAÇÃO DA PESQUISA

Este trabalho foi desenvolvido tendo como base um referencial teórico publicado de estudos desenvolvidos na área (livros, artigos, dissertações, etc) e em materiais disponíveis na *internet*, desenvolvidos, inclusive, pelos próprios autores das ferramentas analisadas (*sites*, arquivos no formato pdf, etc).

Também foram utilizados conhecimentos obtidos durante reuniões e aulas teóricas com professores que ministram disciplinas de gerência de projetos e metodologias ágeis, na Universidade Federal do Paraná.

Segundo Ander-Egg (1978, apud LAKATOS: MARCONI, 2002), esta pesquisa é de natureza científica aplicada, pois trata da investigação pelo interesse de adquirir novos conhecimentos utilizando os resultados na solução de problemas da realidade.

3.2 REVISÃO DA LITERATURA

Para fundamentar teoricamente este trabalho, inicialmente um estudo referente aos modelos tradicionais de processo de *software* (o modelo em cascata, evolucionário e o incremental) foi pesquisado. Em seguida os modelos de desenvolvimento considerados ágeis (Desenvolvimento Orientado a Testes, *Extreme Programming*, *Scrum* e *Kanban*) e o próprio Manifesto Ágil foram estudados.

Pesquisou-se também a respeito dos conceitos e definições em gerência de projetos e o PMBOK, um guia com boas práticas em gerência de projetos muito utilizado nas organizações.

O apoio que um *software* dá aos gerentes foi resumido e a definição de *software* livre foi pesquisada. Mostraram-se também os principais tipos de licença, em se tratando de *software* livre existentes, sendo eles GPL, *Copyleft*, MIT, AGPL, e BSD.

Por fim, o assunto qualidade de *software* foi estudado, incluindo suas definições e considerações. As normas NBR ISO/IEC 9126-1 e NBR ISO/IEC 14598-1 também estiveram presentes na revisão da literatura, pois estas foram fundamentais na realização do objetivo maior deste trabalho, que é a avaliação comparativa de *softwares* livres. Elas orientaram o estabelecimento dos requisitos, especificação, projeto e execução da avaliação no que diz respeito à qualidade de *software*.

3.3 O PROCESSO DE AVALIAÇÃO

A avaliação comparativa dos *softwares* livres iniciou-se pela seleção das ferramentas que seriam avaliadas. Em seguida, seguindo as recomendações da norma ISO/IEC 14598-1, que orienta o planejamento e execução de um processo de avaliação de *software*, foram definidas as etapas a serem seguidas, sendo elas:

- Análise dos Requisitos;
- Especificação da Avaliação;
- Planejamento da Avaliação;
- Execução da Avaliação;
- Resultados.

Para a determinação das características de qualidade que seriam analisadas, utilizou-se a norma ISO/IEC 9126-1, que estabelece um modelo de qualidade contendo métricas internas e externas para avaliação de um produto de *software*.

Com as métricas determinadas, estabeleceu-se um peso para cada métrica e quais seriam as pontuações para cada nível de atendimento às métricas. Com isso, foi feita a avaliação de cada ferramenta, contendo uma pontuação final para cada item. Somando-se a pontuação final de cada item tem-se a pontuação final total da ferramenta. Com a pontuação final total de cada ferramenta fez-se a comparação para determinar o resultado final da avaliação.

3.4 SELEÇÃO DOS *SOFTWARES* LIVRES

Nesta primeira etapa, para a identificação dos *softwares* livres, decidiu-se que a procura seria feita utilizando-se de ferramentas de busca disponíveis na *Internet*, e utilizando como critério palavras, tanto em inglês quanto em português, que abordassem os seguintes assuntos, combinadas de diversas formas:

- Gerenciamento de projetos;
- Gestão de projetos;
- *Scrum*;
- Open Source;
- *Software* Livre;

Também fez parte do critério de seleção dos *softwares* encontrados o fato de, mesmo se houvesse diferenciação de versão, que todas elas atendessem ao requisito de ser considerado *Open Source. Softwares* que possuíssem versões básicas dentro da Licença *Open Source* e versões mais completas, porém pagas, seriam desconsiderados.

Todos os requisitos e recursos necessários para sua instalação e funcionamento deveriam se enquadrar na Licença *Open Source* ou deveriam ser disponibilizados gratuitamente para *download* na *Internet*.

Os programas encontrados deveriam ser voltados especificamente para a gestão de projetos utilizando como base a metodologia ágil *Scrum*. Em suma, os *softwares* deveriam atender basicamente aos seguintes requisitos:

- Estar disponível para *download* na *internet*;
- Se enquadrar, na sua totalidade, na licença *Open Source*;
- Ser baseado na metodologia de desenvolvimento ágil *Scrum*;
- Possuir língua original em inglês ou português;
- Plataforma *web;*
- Funcionar inicialmente no *Windows;*

- Possuir uma versão do sistema para demonstração ou *screenshots* para poder-se visualizar antes de instalar.

Com estes critérios foram encontrados diversos *softwares,* e aqueles que não atenderam aos requisitos foram eliminados. Ao final, quando obteve-se 3 ou mais sistemas, os 3 primeiros que tiveram o maior número de *downloads* no ano de 2011 seriam os selecionados para a avaliação comparativa.

Alguns dos *softwares* encontrados foram descartados, pois devido à falta de instrução/manual que auxiliasse na instalação, ou mesmo uma dificuldade maior em conseguir algum recurso necessário para seu funcionamento, como *link* fora do ar de um *framework* específico para ele, fazia com ele que fosse desconsiderado e a busca fosse retomada.

Depois de muita busca foram selecionadas as ferramentas *Firescrum*, *Agilefant* e *Kunagi*. O QUADRO 1 mostra a quantidade de *downloads* que cada sistema obteve, considerando o ano de 2011, e que foram determinantes para a escolha dos mesmos.

NOME	DOWNLOADS (2011)
Agilefant	13930
Kunagi	5438
Firescrum	4629

QUADRO 1 – QUANTIDADE DE *DOWNLOADS*

FONTE: Sourceforge (2012a, 2012b, 2012c)

3.5 FERRAMENTAS *SCRUM*

3.5.1 *Firescrum*

Firescrum é uma ferramenta de gestão de projetos baseado na metodologia *Scrum* que pode ser encontrada na *internet*, mais especificamente no seguinte endereço: http://www.firescrum.org/download/.

Esta ferramenta é resultado de um trabalho de dissertação de mestrado e suas funcionalidades foram implementadas por cerca de 60 pós-graduandos da disciplina de Engenharia de *Software* na Universidade Federal de Pernambuco (CAVALCANTI, 2009).

Firescrum é uma aplicação *Web*, ou seja, ela pode ser acessada de qualquer lugar que tenha um navegador (*browser*) de *internet*. Utiliza conceitos de *Web 2.0* e de *Rich Internet Applications* (RIA). A arquitetura dessa aplicação está dividida em duas partes: o *front-end (interface)* e o *back-end* (FIGURA 8). O *front-end* foi desenvolvido utilizando o *Adobe Flex*, um *framework* de código aberto que permite a criação de Aplicações de *Internet* Rica (RIA) e que podem ser visualizados por qualquer navegador de *internet* que possua suporte ao *plugin* do *Adobe Flash Player* instalado. O *back-end*, ou seja, onde estão os mecanismos e regras de negócios do sistema, foi desenvolvido utilizando a linguagem de programação *Java* e o banco de dados desta ferramenta utiliza o *PostgreSql*, ambas ferramentas livres, disponíveis gratuitamente na *internet* (CAVALCANTI, 2009).

O que chama a atenção nesta solução desenvolvida pelos pós-graduandos, e que recebeu o nome de *Firescrum*, é o fato de que, desde o início, o seu desenvolvimento foi focado na usabilidade e simplicidade de uso, com uma arquitetura modularizada e independente.

Para o funcionamento deste aplicativo é necessário instalar no servidor o *Java Development Kit* (*JDK*) versão 6 ou superior, o servidor *Apache Tomcat* versão 6 ou superior, o *Red5 Server* 7.0 e o Banco de dados *PostgreSql* 8 ou versão superior.

A versão do *Firescrum* utilizada neste trabalho foi a 1.0 beta 5.

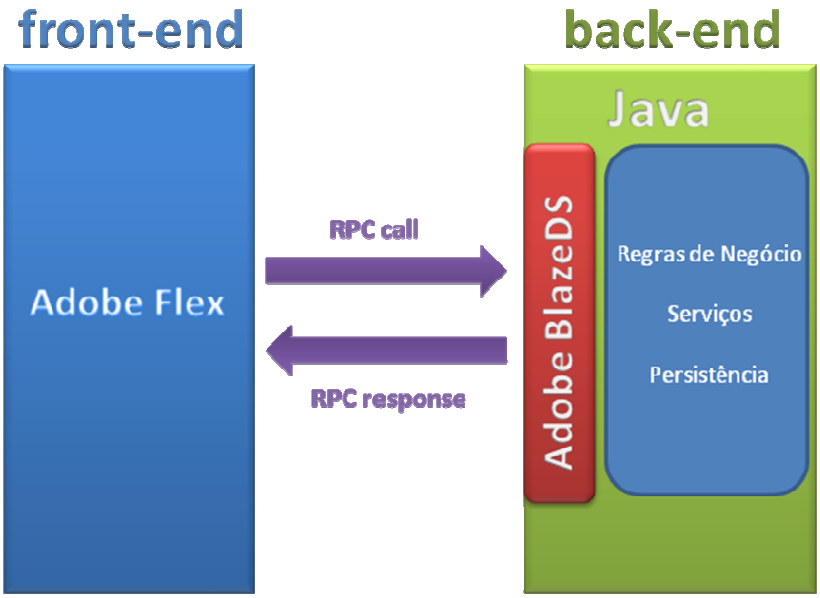


FIGURA 8 – ARQUITETURA MACRO DO *FIRESCRUM*
FONTE: Cavalcanti (2009)

3.5.2 *Agilefant*

Agilefant é uma ferramenta de gestão de projetos baseado na metodologia *Scrum* que pode ser encontrada na *internet*, mais especificamente no seguinte endereço: http://www.agilefant.org/wiki/display/AEF/Downloads.

Esta ferramenta foi desenvolvida por um Grupo de Pesquisa em Processo de *Software* da Universidade Aalto na Finlândia (AGILEFANT, 2011).

Agilefant também é uma aplicação *web*, ou seja, ela pode ser acessada de qualquer lugar que tenha um navegador (*browser*) de *internet*. Desenvolvida para ser simples e atender às atividades de gerenciamento de desenvolvimento ágil de *software,* o *Agilefant* possuiu 13.930 *downloads* no ano de 2011 e, em média, cerca de 200 visitas diárias em seu site, sendo o recordista em *downloads* dentre as três ferramentas avaliadas (AGILEFANT, 2011; SOURCEFORGE, 2012a).

Um ponto positivo desta ferramenta é o fato de estar sempre sendo desenvolvida uma nova versão melhorada. Em março de 2011, o grupo de pesquisa responsável pelo seu desenvolvimento publicou no *website* da ferramenta que já estava trabalhando em uma nova versão para o *Agilefant.*

Para o funcionamento deste aplicativo é necessário instalar no servidor o *Java Development Kit* (*JDK*) versão 6 ou superior, o servidor *Apache Tomcat* versão 5.5 ou superior, o banco de dados *Mysql* 5 ou versão superior e um navegador de *internet Mozilla Firefox* ou *Google Chrome*.

A versão do *Agilefant* utilizada neste trabalho foi a 2.0.5.

3.5.3 *Kunagi*

Kunagi é uma ferramenta de gestão de projetos baseado na metodologia *Scrum* que pode ser encontrada, na *internet*, para *download* no seguinte endereço: http://www.kunagi.org/download.html.

Kunagi é uma ferramenta *web* de gestão completa para projetos baseados em *Scrum*. Além das melhores práticas do *Scrum*, esta ferramenta oferece informações adicionais do projeto, como requisitos não-funcionais, arquivos, últimos eventos, etc (KUNAGI, 2012).

Kunagi foi desenvolvida por um grupo de 5 profissionais da área de projetos de desenvolvimento de *software* e recebeu a colaboração de outros profissionais, de diversas nacionalidades, também da área.

Para o funcionamento deste aplicativo é necessário instalar a última versão do servidor *Apache Tomcat* e baixar o arquivo contendo a aplicação. O site da ferramenta também oferece a opção de um arquivo executável que faz todo o processo de instalação do servidor *Apache Tomcat* e dos arquivos necessários para o programa.

A versão do *Kunagi* utilizada neste trabalho foi a 0.22.3.

3.6 QUADRO COMPARATIVO

O QUADRO 2 foi elaborado com base nas informações obtidas nos *websites* dos autores de cada *software*.

Dois fatos chamam a atenção para as três ferramentas selecionadas: (1) todas elas possuem o inglês como idioma original e (2) os três utilizavam a linguagem de programação *Java* como uma das linguagens predominantes no seu desenvolvimento.

Sobre a facilidade de instalação, o *Kunagi* mostrou ser o mais fácil. Precisou apenas executar um único arquivo para que todo o sistema funcionasse, já o *Firescrum* precisou de 4 pré-requisitos instalados para que o sistema funcionasse adequadamente.

Por fim, o último item compara o tipo da licença de cada *software.* Cada um possui uma licença específica, no entanto, todas as licenças enquadram os *softwares* como livres.

	FIRESCRUM	**AGILEFANT**	**KUNAGI**
BANCO DE DADOS	PostgreSql	MySql	XML
IDIOMA ORIGINAL	Inglês	Inglês	Inglês
SISTEMA OPERACIONAL	*Windows* ou *Linux*	Windows *ou* Linux	Windows ou *Linux*
INTERFACE	*Web*	*Web*	*Web*
ANO DA PRIMEIRA VERSÃO	2009	2008	2010
ÚLTIMA VERSÃO ESTÁVEL	1.0 Beta 5	2.0.5	0.22.3
LINGUAGEM DE PROGRAMAÇÃO	Java	Java	Java
QUANTIDADE DE PRÉ-REQUISITOS PARA INSTALAÇÃO	4	3	1
LICENÇA	GPL	MIT	AGPL

QUADRO 2 – CARACTERÍSTICAS *VERSUS SOFTWARES*
FONTE: O autor (2012)

3.7 ANÁLISE DOS REQUISITOS

Esta seção estabelece um conjunto de características e subcaracterísticas de qualidade baseados na norma NBR ISO/IEC 9126-1, dentro do contexto do método ágil *Scrum,* que serão utilizados, posteriormente, para avaliar se as três ferramentas escolhidas possuem a qualidade adequada, levando em consideração estes critérios e comparando, ao final, o resultado entre elas.

A avaliação será feita em cima da versão final dos três produtos, disponíveis no momento da realização deste trabalho, e abrangerá o produto como um todo, não focando apenas em um módulo ou funcionalidade.

Para a determinação das características de qualidade que serão analisadas, utilizou-se a norma NBR ISO/IEC 9126-1, que define seis características de qualidade para avaliação de produto de *software*. A composição da lista com os itens da avaliação está disposta da seguinte forma: característica, subcaracterística e descrição. Cada item da tabela possui um número identificador único, que permite que este possa ser referenciado a qualquer momento neste trabalho, se necessário, facilitando assim, sua identificação.

A TABELA 2 mostra os critérios utilizados neste trabalho para a característica de funcionalidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 2 – CRITÉRIOS DE QUALIDADE - FUNCIONALIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
1 -FUNCIONALIDADE	1.1 - Adequação	1.1.1 - Exportar o projeto.
		1.1.2 - Importar o projeto.
		1.1.3 - *Planning Poker.*
		1.1.4 - Meio de comunicação entre os usuários.
		1.1.5 - Criar e manter *Product Backlog.*
		1.1.6 - Atribuir prioridades para os itens do *Product Backlog.*
		1.1.7 - Estimar tempo de execução para os itens do *Product Backlog.*
		1.1.8 - Atribuir responsáveis para os itens do *Product Backlog.*
		1.1.9 - Criar e manter um *Sprint.*
		1.1.10 - Criar e manter uma lista de impedimentos.
		1.1.11 - Criar e manter mais de um projeto.
	1.2 - Acurácia	1.2.1 - Emissão de relatório de atividades do *Product Backlog.*
		1.2.2 - Emissão de relatório de atividades do *Sprint Backlog.*
		1.2.3 - Emissão de relatório/gráfico *Sprint Burndown.*
	1.3 - Segurança	1.3.1 – Permite customização do tipo de acesso do usuário.
	1.4 - Interoperabilidade	1.4.1 - Interage com outros sistemas.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

A TABELA 3 mostra os critérios utilizados neste trabalho para a característica de confiabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1 e no conteúdo da fundamentação teórica referente ao método ágil *Scrum* constante na seção 2.2.4 deste trabalho.

TABELA 3 – CRITÉRIO DE QUALIDADE - CONFIABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
2 - CONFIABILIDADE	2.1 - Maturidade	2.1.1 - O *software* evita falhas.
	2.2 - Tolerância a falhas	2.2.1 - O *software* mantém seu funcionamento quando da ocorrência de uma falha.
	2.3 - Recuperabilidade	2.3.1 - O *software* é capaz de se recuperar após uma falha.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

A TABELA 4 mostra os critérios utilizados neste trabalho para a característica de usabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 4 – CRITÉRIO DE QUALIDADE - USABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
3 - USABILIDADE	3.1 - Inteligibilidade	3.1.1 - A impressão inicial que o usuário tem do *software* permite compreender se ele é apropriado para a tarefa desejada.
		3.1.2 - Existe documentação descritiva do *software.*
	3.2 - Apreensibilidade	3.2.1 - Em geral, o *software* é de fácil utilização.
	3.3 - Operacionalidade	3.3.1 - Tolera erros de operação do(s) usuário(s).
		3.3.2 - Utiliza funcionalidades que induzem o funcionamento ao usuário.
	3.4 - Atratividade	3.4.1 - Utiliza recursos gráficos atraentes aos usuários.
		3.4.2 - Presta informações adequadas aos usuários.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

A TABELA 5 mostra os critérios utilizados neste trabalho para a característica de eficiência e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 5 – CRITÉRIO DE QUALIDADE - EFICIÊNCIA

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
4 - EFICIÊNCIA	4.1 - Comportamento em relação ao tempo	4.1.1 - Tempo de resposta e processamento apropriados quando executa suas funções.
	4.2 - Utilização de recursos	4.2.1 - Utiliza diversos tipos e quantidades de recursos quando executa suas funções estabelecidas.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

A TABELA 6 mostra os critérios utilizados neste trabalho para a característica de manutenibilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 6 – CRITÉRIO DE QUALIDADE - MANUTENIBILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
5 - MANUTENIBILIDADE	5.1 - Analisabilidade	5.1.1 - Permite o diagnóstico de deficiência ou causas de falhas, ou de partes a serem modificadas.
	5.2 - Modificabilidade	5.2.1 - Permite implementar uma modificação especificada.
	5.3 - Estabilidade	5.3.1 - Evita efeitos inesperados decorrentes de modificações no *software.*
	5.4 - Testabilidade	5.4.1 - Permite avaliação do *software* quando modificado.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

A TABELA 7 mostra os critérios utilizados neste trabalho para a característica de portabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 7 – CRITÉRIO DE QUALIDADE - PORTABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
6 - PORTABILIDADE	6.1 - Adaptabilidade	6.1.1 - Funciona em diversos navegadores (*browsers*) sem a necessidade de configuração adicional ou instalação de outros recursos (*plug-in*).
	6.2 - Capacidade para ser instalado	6.2.1 - O *software* é de fácil instalação.
	6.3 - Coexistência	6.3.1 - Funciona, sem problema, ao mesmo tempo com outros *softwares* que utilizam os mesmos recursos.

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO
	6.4 - Capacidade para substituir versões	6.4.1 - Permite atualização de versão mantendo o conteúdo do banco de dados existente.

FONTE: O autor (baseado em NBR ISO/IEC 9126-1, 2012)

3.8 ESPECIFICAÇÃO DA AVALIAÇÃO

Nesta etapa do trabalho pretende-se especificar as métricas que serão utilizadas na avaliação comparativa das ferramentas, os níveis de pontuação para essas métricas e por fim o critério de julgamento para os resultados encontrados.

Para avaliar o nível de atendimento de cada item das subcaracterísticas dos critérios de qualidade utilizados, será atribuída uma descrição do nível de atendimento, que possui valores que variam em uma escala de 0 (zero) a 2 (dois) conforme pode ser visto na TABELA 8, onde 0 (zero) constitui o pior valor e 2 (dois) o melhor para aquela métrica.

TABELA 8 – VALORES DAS MÉTRICAS

DESCRIÇÃO	VALOR
Não Atende	0
Atende Parcialmente	1
Atende	2

FONTE: O autor (2012)

Para a pontuação final de cada item dos critérios de qualidade, que compõem o resultado sintético da avaliação, são atribuídos pesos para esses critérios. O valor de cada peso varia em uma escala de 1 (um) a 3 (três), conforme pode ser visto na TABELA 9.

TABELA 9 – PESO DA MÉTRICA

DESCRIÇÃO	VALOR
Essencial	3
Importante	2
Desejável	1

FONTE: O autor (2012)

A TABELA 10 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de funcionalidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

A atribuição dos pesos para cada item foi feita pelo autor deste trabalho analisando a necessidade daquele item para um bom atendimento do produto segundo os princípios da metodologia de desenvolvimento ágil *Scrum*.

TABELA 10 – PESOS DO CRITÉRIO DE QUALIDADE - FUNCIONALIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
1 -FUNCIONALIDADE	1.1 - Adequação	1.1.1 - Exportar o projeto.	2
		1.1.2 - Importar o projeto.	2
		1.1.3 - Planning Poker.	2
		1.1.4 - Meio de comunicação entre os usuários.	1
		1.1.5 - Criar e manter *Product Backlog.*	3
		1.1.6 - Atribuir prioridades para os itens do *Product Backlog.*	2
		1.1.7 - Estimar tempo de execução para os itens do *Product Backlog.*	3
		1.1.8 - Atribuir responsáveis para os itens do *Product Backlog.*	3
		1.1.9 - Criar e manter um *Sprint.*	3
		1.1.10 - Criar e manter uma lista de impedimentos.	3
		1.1.11 - Criar e manter mais de um projeto.	3
	1.2 - Acurácia	1.2.1 - Emissão de relatório de atividades do *Product Backlog.*	1
		1.2.2 - Emissão de relatório de atividades do *Sprint Backlog.*	1
		1.2.3 - Emissão de relatório/gráfico *Sprint Burndown.*	3
	1.3 - Segurança	1.3.1 – Permite customização do tipo de acesso do usuário.	2
	1.4 - Interoperabilidade	1.4.1 - Interage com outros sistemas.	1

FONTE: O autor (2012)

A TABELA 11 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de confiabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1 e no conteúdo da fundamentação teórica referente ao método ágil *Scrum* constante na seção 2.2.4 deste trabalho.

TABELA 11 – PESOS DO CRITÉRIO DE QUALIDADE - CONFIABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
2 - CONFIABILIDADE	2.1 - Maturidade	2.1.1 - O *software* evita falhas.	2
	2.2 - Tolerância a falhas	2.2.1 - O *software* mantém seu funcionamento quando da ocorrência de uma falha.	3
	2.3 - Recuperabilidade	2.3.1 - O *software* é capaz de se recuperar após uma falha.	3

FONTE: O autor (2012)

A TABELA 12 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de usabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 12 – PESOS DO CRITÉRIO DE QUALIDADE - USABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
3 - USABILIDADE	3.1 - Inteligibilidade	3.1.1 - A impressão inicial que o usuário tem do *software* permite compreender se ele é apropriado para a tarefa desejada.	2
		3.1.2 - Existe documentação descritiva do *software.*	2
	3.2 - Apreensibilidade	3.2.1 - Em geral, o *software* é de fácil utilização.	3
	3.3 - Operacionalidade	3.3.1 - Tolera erros de operação do(s) usuário(s).	2
		3.3.2 - Utiliza funcionalidades que induzem o funcionamento ao usuário.	2
	3.4 - Atratividade	3.4.1 - Utiliza recursos gráficos atraentes aos usuários.	2
		3.4.2 - Presta informações adequadas aos usuários.	2

FONTE: O autor (2012)

A TABELA 13 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de eficiência e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 13 – PESOS DO CRITÉRIO DE QUALIDADE - EFICIÊNCIA

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
4 - EFICIÊNCIA	4.1 - Comportamento em relação ao tempo	4.1.1 - Tempo de resposta e processamento apropriados quando executa suas funções.	2
	4.2 - Utilização de recursos	4.2.1 - Utiliza diversos tipos e quantidades de recursos quando executa suas funções estabelecidas.	1

FONTE: O autor (2012)

A TABELA 14 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de manutenibilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 14 – PESOS DO CRITÉRIO DE QUALIDADE - MANUTENIBILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
5 - MANUTENIBILIDADE	5.1 - Analisabilidade	5.1.1 - Permite o diagnóstico de deficiência ou causas de falhas, ou de partes a serem modificadas.	1
	5.2 - Modificabilidade	5.2.1 - Permite implementar uma modificação especificada.	2
	5.3 - Estabilidade	5.3.1 - Evita efeitos inesperados decorrentes de modificações no *software*.	3
	5.4 - Testabilidade	5.4.1 - Permite avaliação do *software* quando modificado.	3

FONTE: O autor (2012)

A TABELA 15 mostra os critérios e seus respectivos pesos, utilizados neste trabalho para a característica de portabilidade e suas subcaracterísticas, baseados no modelo de qualidade definido pela norma NBR ISO/IEC 9126-1.

TABELA 15 – PESOS DO CRITÉRIO DE QUALIDADE - PORTABILIDADE

CARACTERÍSTICA	SUBCARACTERÍSTICA	DESCRIÇÃO	PESO
6 - PORTABILIDADE	6.1 - Adaptabilidade	6.1.1 - Funciona em diversos navegadores (*browsers*) sem a necessidade de configuração adicional ou instalação de outros recursos (*plug-in*).	3
	6.2 - Capacidade para ser instalado	6.2.1 - O *software* é de fácil instalação	2
	6.3 - Coexistência	6.3.1 - Funciona, sem problema, e ao mesmo tempo com outros *softwares* que utilizam os mesmos recursos	3
	6.4 - Capacidade para substituir versões	6.4.1 - Permite atualização de versão mantendo o conteúdo do banco de dados existente	3

FONTE: O autor (2012)

Após determinar os níveis de atendimento para cada item dos critérios de qualidade, em todas as ferramentas, será calculada a pontuação final de cada item utilizando o valor daquele nível de atendimento (TABELA 8) multiplicado pelo seu peso (TABELA 9) conforme pode ser visto na seguinte fórmula:

$$Pi = V \times P$$

onde:

Pi = Pontuação total do item
V = Valor atribuído à métrica
P = Peso da métrica

Somando-se todas as pontuações finais de todos os itens, tem-se a pontuação final daquela ferramenta conforme pode ser visto na seguinte fórmula:

$$Pf = \sum_{i=1}^{n} Pi$$

onde:

Pf = Pontuação total final da ferramenta
n = Quantidade de critérios da ferramenta
i = Número do critério, variando de 1 a n
Pi = Pontuação total do item

Com a pontuação final de todas as ferramentas será possível realizar a comparação dos produtos, utilizando-se um quadro comparativo com todos os itens e sua pontuação final. A ferramenta que tiver maior pontuação final será considerada de melhor qualidade de atendimento do que aquela que tem menor pontuação.

3.9 PLANEJAMENTO DA AVALIAÇÃO

Para a realização desta avaliação, os procedimentos constantes na TABELA 16 serão realizados na ordem especificada, para a avaliação de cada ferramenta.

Deve-se realizar todos os procedimentos para um primeiro *software* e, assim que for concluído, deve-se reiniciar os procedimentos para o outro *software*, e assim consecutivamente até que todas as ferramentas sejam avaliadas. Um total de 51 procedimentos deverá ser realizado, totalizando 708 minutos (aproximadamente 11,8 horas) de tempo previsto para cada sistema.

Como nesta avaliação serão analisados três sistemas, estima-se utilizar um total máximo de 36 horas para a realização dos procedimentos para as três ferramentas.

TABELA 16 – PROCEDIMENTOS

ORDEM	PROCEDIMENTO	TEMPO PREVISTO (minutos)
1	Instalar o *software* e os outros recursos para seu funcionamento, se necessário.	40
2	Avaliar a capacidade do *software* de ser instalado (item 6.2.1).	2
3	Percorrer o sistema analisando se é possível, em uma primeira impressão, compreender a finalidade de cada módulo ou funcionalidade existente.	30
4	Avaliar o item 3.1.1.	2
5	Procurar a existência de manual, documentação ou ajuda para a utilização da ferramenta.	30
6	Avaliar o item 3.1.2.	2
7	Cadastrar um novo projeto no sistema com a descrição “Sistema Financeiro”.	1
8	Dentro deste projeto, criar um *Product Backlog* com no mínimo 5 requisitos, especificando suas prioridades, tempo estimado de execução e responsáveis.	10
9	Avaliar os itens 1.1.5 ao 1.1.8.	6
10	Criar um “*Sprint*” com 5 atividades para o primeiro requisito do *Product Backlog*.	10
11	Criar mais um “*Sprint” com mais* 5 atividades no segundo requisito do *Product Backlog.*	10
12	Lançar uma data de execução, simulando a execução, para cada atividade, do primeiro *Sprint* criado.	10
13	Avaliar o item 1.1.9.	2
14	Verificar se existe algum meio de comunicação entre os usuários do sistema (*chat*, mensagens, etc) e testar seu funcionamento, caso exista.	20
15	Avaliar o item 1.1.4.	2
16	Verificar se existe a possibilidade de utilizar *Planning Poker* e testar seu funcionamento, caso exista.	30
17	Avaliar o item 1.1.3.	2
18	Testar possibilidade de se criar um projeto novo mantendo o já existente.	10
19	Avaliar o item 1.1.10.	2
20	Verificar se o sistema permite exportar e importar um projeto.	10
21	Avaliar os itens 1.1.1 e 1.1.2.	2
22	Emitir um relatório de *Product Backlog.*	10
23	Avaliar o item 1.2.1.	2
24	Emitir um relatório de atividades do primeiro *Sprint Backlog* criado.	10
25	Avaliar o item 1.2.2.	2
26	Emitir um relatório/gráfico de *Sprint Burndown.*	10
27	Avaliar o item 1.2.3.	2
28	Verificar se o sistema permite criar diferentes níveis de acesso de usuário.	10
29	Avaliar o item 1.3.1.	2
30	Verificar se é citada, em alguma documentação, a possibilidade de se modificar o sistema para ele se comunicar com outros, se for necessário.	15
31	Avaliar o item 1.4.1.	2
32	Acessar novamente os formulários utilizados e avaliar o comportamento do sistema quanto à obrigatoriedade de preenchimento dos campos.	30
33	Acessar novamente os formulários de cadastro, utilizados nas outras etapas, e avaliar o comportamento do sistema quanto ao preenchimento errado de informações em campos específicos, como aqueles onde exige o preenchimento de apenas números, data e etc.	40
34	Avaliar o item 2.1.1, 2.2.1 e 2.3.1.	10
35	Avaliar a facilidade de utilização do *software* (item 3.2,1).	10
36	Avaliar a tolerância a erros apresentada pelo *software* (item 3.3.1).	10
37	Avaliar a capacidade de indução ao funcionamento apresentada, em geral, pelo *software* (item 3.3.2).	10
38	Avaliar o nível de utilização de recursos gráficos (item 3.4.1).	10
39	Avaliar se o *software* oferece informações que auxiliam a operação do sistema pelos usuários (item 3.4.2).	10

ORDEM	PROCEDIMENTO	TEMPO PREVISTO (minutos)
40	Avaliar o tempo geral de resposta do sistema com relação às operações realizadas (item 4.1.1).	10
41	Avaliar a utilização geral de recursos do sistema (item 4.2.1).	10
42	Avaliar se as mensagens de erros, quando ocorrer um erro, possuem informações da causa de uma falha (item 5.1.1).	10
43	Verificar se é possível fazer uma modificação (customização, parametrização) de algum módulo ou funcionalidade (item 5.2.1).	10
44	Avaliar, caso seja possível a customização/parametrização, se o sistema evita falhas decorrentes dessa ação (item 5.3.1).	10
45	Avaliar se é possível testar as modificações realizadas (item 5.4.1).	10
46	Testar o acesso e utilização do sistema no navegador *Internet Explorer*.	60
47	Testar o acesso e utilização do sistema no navegador *Mozilla Firefox*.	60
48	Testar o acesso e utilização do sistema no navegador *Google Chrome.*	60
49	Avaliar a adaptabilidade do sistema quanto ao funcionamento nos três principais navegadores (item 6.1.1).	10
50	Avaliar o funcionamento do sistema quando em uso em conjunto com outros *softwares* existentes no computador (item 6.3.1).	15
51	Verificar e avaliar se é possível fazer a atualização de versão do *software* sem perder os dados já cadastrados (item 6.4.1).	15
	TEMPO TOTAL =	**708**

FONTE: O autor (2012)

Quando o nível de atendimento de um *software* tiver sido avaliado seguindo os procedimentos da TABELA 16, recebendo, cada item, a descrição referente aos valores da TABELA 8, calcular-se-á a pontuação total final daquele *software* somando-se a pontuação final de todos os itens. Com a pontuação final dos três sistemas estabelecida, será feita a comparação final que resultará na avaliação comparativa desejada.

3.10 EXECUÇÃO DA AVALIAÇÃO

Nesta seção será feita a avaliação dos *softwares* livres para suporte ao desenvolvimento ágil baseado no método ágil *Scrum.*

As configurações de *hardware* do computador, onde foram instalados os sistemas, é a seguinte:

- Processador *Core* 2 *Quad* 2.4GHz;
- Memória *RAM* de 4Gb;

- Adaptador de Vídeo com 512 Mega bytes;
- *Windows* XP *Professional* (SP3);

Para a avaliação do item 6.1.1, sobre o funcionamento nos diversos navegadores de *internet*, foram testados nos seguintes *browsers*:

- *Mozilla Firefox* versão 10.0.2;
- *Internet Explorer* versão 8.0.6001.18702;
- *Google Chrome* versão 17.0.963.79.

3.10.1 Avaliação das ferramentas

Com base nos princípios do método ágil *Scrum* e nos princípios de funcionamento geral de um *software,* as ferramentas selecionadas foram avaliadas e sua pontuação final obtida. Conforme citado na seção 3.2 deste trabalho, as ferramentas avaliadas foram:

- *Firescrum*;
- *Agilefant*;
- *Scrumpy*.

Os resultados da avaliação feita na ferramenta *Firescrum* podem ser vistos na TABELA 17.

TABELA 17 – AVALIAÇÃO DA FERRAMENTA *FIRESCRUM*

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
1.1.1	Exportar o projeto	0	2	0	Não possui esta funcionalidade
1.1.2	Importar o projeto	0	2	0	Não possui esta funcionalidade
1.1.3	*Planning Poker*	2	2	4	
1.1.4	Meio de comunicação entre os usuários	2	1	2	
1.1.5	Criar e manter um *Product Backlog*	2	3	6	
1.1.6	Atribuir prioridades para as atividades do *Product Backlog*	2	2	4	
1.1.7	Estimar tempo de execução para as atividades do *Product Backlog*	1	3	3	Permite estimar o esforço necessário apenas para determinadas situações
1.1.8	Atribuir responsáveis para as atividades do *Product Backlog*	2	3	6	
1.1.9	Criar e manter um *Sprint*	2	3	6	
1.1.10	Criar e manter uma lista de impedimentos	2	3	6	
1.1.11	Criar e manter mais de um projeto	2	3	6	
1.2.1	Emissão de relatório de atividades do *Product Backlog*	0	1	0	Não possui relatório de *Product Backlog*
1.2.2	Emissão de relatório de atividades do *Sprint Backlog*	0	1	0	Não possui relatório de atividades do *Sprint Backlog*
1.2.3	Emissão de relatório/gráfico *Sprint Burndown*	2	2	4	
1.3.1	Customização do tipo de acesso do usuário	1	2	2	Permite apenas dois tipos de perfis de usuário
1.4.1	Interage com outros sistemas	0	1	0	Nenhuma informação a respeito foi encontrada
2.1.1	O *software* evita falhas	1	2	2	Algumas informações não se atualizaram sozinhas na tela quando atualizadas
2.2.1	O *software* mantém seu funcionamento quando da ocorrência de uma falha	2	3	6	
2.3.1	O *software* é capaz de se recuperar após uma falha	1	3	3	Em diversas vezes foi preciso usar o recurso de atualizar a tela, pois não mostrava modificações realizadas em informações
3.1.1	A impressão inicial que o usuário tem do *software* permite compreender se ele é apropriado para a tarefa desejada	2	2	4	
3.1.2	Existe documentação descritiva do *software*	1	2	2	

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
3.2.1	Em geral, o *software* é de fácil utilização	2	3	6	
3.3.1	Tolera erros de operação do(s) usuário(s)	2	2	4	
3.3.2	Utiliza funcionalidades que induzem o funcionamento ao usuário	1	2	2	Possui uma funcionalidade de arrastar itens do *Backlog* que não induz o funcionamento com facilidade
3.4.1	Utiliza recursos gráficos atraentes aos usuários	1	2	2	Não utiliza figuras e ícones que facilitam a utilização do usuário
3.4.2	Presta informações adequadas aos usuários	1	2	2	
4.1.1	Tempo de resposta e processamento apropriados quando executa suas funções	2	2	4	
4.2.1	Utiliza diversos tipos e quantidades de recursos quando executa suas funções estabelecidas	1	1	1	Poderia utilizar mais recursos para uma atualização mais dinâmica da tela
5.1.1	Permite o diagnóstico de deficiência ou causas de falhas, ou de partes a serem modificadas	0	1	0	O sistema apresentou algumas telas de erro sem nenhuma informação
5.2.1	Permite implementar uma modificação especificada	1	2	2	Permite parametrizar algumas configurações de tela
5.3.1	Evita efeitos inesperados decorrentes de modificações no *software*	2	3	6	
5.4.1	Permite avaliação do *software* quando modificado	2	3	6	
6.1.1	Funciona em diversos navegadores (*browsers*) sem a necessidade de configuração adicional ou instalação de outros recursos (*plug-in*)	2	3	6	
6.2.1	O *software* é de fácil instalação	0	2	0	Necessita de instalação de diversos componentes, além de configurações avançadas
6.3.1	Funciona, sem problema, ao mesmo tempo com outros *softwares* que utilizam os mesmos recursos	2	3	6	
6.4.1	Permite atualização de versão mantendo o conteúdo do banco de dados existente	2	3	6	
			Pontuação Final =	**119**	

FONTE: O autor (2012)

Os resultados da avaliação feita na ferramenta *Agilefant* podem ser vistos na TABELA 18.

TABELA 18 – AVALIAÇÃO DA FERRAMENTA *AGILEFANT*

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
1.1.1	Exportar o projeto	0	2	0	Não permite exportar o projeto
1.1.2	Importar o projeto	0	2	0	Não permite importar o projeto
1.1.3	*Planning Poker*	0	2	0	Não possui *Planning Poker*
1.1.4	Meio de comunicação entre os usuários	0	1	0	Não possui chat nem outro tipo de comunicação entre usuários
1.1.5	Criar e manter um *Product Backlog*	2	3	6	
1.1.6	Atribuir prioridades para as atividades do *Product Backlog*	2	2	4	
1.1.7	Estimar tempo de execução para as atividades do *Product Backlog*	2	3	6	
1.1.8	Atribuir responsáveis para as atividades do *Product Backlog*	2	3	6	
1.1.9	Criar e manter um *Sprint*	2	3	6	
1.1.10	Criar e manter uma lista de impedimentos	0	3	0	Não permite manter uma lista de impedimentos
1.1.11	Criar e manter mais de um projeto	2	3	6	
1.2.1	Emissão de relatório de atividades do *Product Backlog*	1	1	1	Não possui impressão em forma de relatório, apenas consulta no sistema
1.2.2	Emissão de relatório de atividades do *Sprint Backlog*	1	1	1	Não possui impressão em forma de relatório, apenas consulta no sistema
1.2.3	Emissão de relatório/gráfico *Sprint Burndown*	2	2	4	
1.3.1	Customização do tipo de acesso do usuário	1	2	2	Permite apenas dois tipos de perfis de usuário
1.4.1	Interage com outros sistemas	0	1	0	Nenhuma informação a respeito foi encontrada
2.1.1	O *software* evita falhas	2	2	4	
2.2.1	O *software* mantém seu funcionamento quando da ocorrência de uma falha	2	3	6	
2.3.1	O *software* é capaz de se recuperar após uma falha	2	3	6	
3.1.1	A impressão inicial que o usuário tem do *software* permite compreender se ele é apropriado para a tarefa desejada	2	2	4	
3.1.2	Existe documentação descritiva do *software*	2	2	4	

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
3.2.1	Em geral, o *software* é de fácil utilização	2	3	6	
3.3.1	Tolera erros de operação do(s) usuário(s)	2	2	4	
3.3.2	Utiliza funcionalidades que induzem o funcionamento ao usuário	1	2	2	Em alguns campos falta indicação do tipo de dados que deve ser inserido
3.4.1	Utiliza recursos gráficos atraentes aos usuários	2	2	4	
3.4.2	Presta informações adequadas aos usuários	1	2	2	Em alguns campos não se sabe ao certo que tipo de valor preencher
4.1.1	Tempo de resposta e processamento apropriados quando executa suas funções	2	2	4	
4.2.1	Utiliza diversos tipos e quantidades de recursos quando executa suas funções estabelecidas	2	1	2	
5.1.1	Permite o diagnóstico de deficiência ou causas de falhas, ou de partes a serem modificadas	0	1	0	
5.2.1	Permite implementar uma modificação especificada	1	2	2	Permite parametrizar algumas configurações de tela
5.3.1	Evita efeitos inesperados decorrentes de modificações no *software*	2	3	6	
5.4.1	Permite avaliação do *software* quando modificado	2	3	6	
6.1.1	Funciona em diversos navegadores (*browsers*) sem a necessidade de configuração adicional ou instalação de outros recursos (*plug-in*)	2	3	6	
6.2.1	O *software* é de fácil instalação	1	2	2	Precisa instalar alguns componentes
6.3.1	Funciona, sem problema, ao mesmo tempo com outros *softwares* que utilizam os mesmos recursos	2	3	6	
6.4.1	Permite atualização de versão mantendo o conteúdo do banco de dados existente	2	3	6	
			Pontuação Final =	**124**	

FONTE: O autor (2012)

Os resultados da avaliação feita na ferramenta *Kunagi* podem ser vistos na TABELA 19.

TABELA 19 – AVALIAÇÃO DA FERRAMENTA *KUNAGI*

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
1.1.1	Exportar o projeto	0	2	0	
1.1.2	Importar o projeto	0	2	0	
1.1.3	*Planning Poker*	2	2	4	
1.1.4	Meio de comunicação entre os usuários	2	1	2	
1.1.5	Criar e manter um *Product Backlog*	2	3	6	
1.1.6	Atribuir prioridades para as atividades do *Product Backlog*	0	2	0	Não permite atribuir prioridades
1.1.7	Estimar tempo de execução para as atividades do *Product Backlog*	2	3	6	
1.1.8	Atribuir responsáveis para as atividades do *Product Backlog*	0	3	0	Não permite atribuir responsáveis
1.1.9	Criar e manter um *Sprint*	2	3	6	
1.1.10	Criar e manter uma lista de impedimentos	2	3	6	
1.1.11	Criar e manter mais de um projeto	2	3	6	
1.2.1	Emissão de relatório de atividades do *Product Backlog*	2	1	2	
1.2.2	Emissão de relatório de atividades do *Sprint Backlog*	2	1	2	
1.2.3	Emissão de relatório/gráfico *Sprint Burndown*	2	2	4	
1.3.1	Customização do tipo de acesso do usuário	2	2	4	
1.4.1	Interage com outros sistemas	0	1	0	Nenhuma informação a respeito foi encontrada
2.1.1	O *software* evita falhas	2	2	4	
2.2.1	O *software* mantém seu funcionamento quando da ocorrência de uma falha	2	3	6	
2.3.1	O *software* é capaz de se recuperar após uma falha	2	3	6	
3.1.1	A impressão inicial que o usuário tem do *software* permite compreender se ele é apropriado para a tarefa desejada	2	2	4	
3.1.2	Existe documentação descritiva do *software*	2	2	4	

ID	DESCRIÇÃO	AVAL.	PESO	PONTUAÇÃO	OBSERVAÇÃO
3.2.1	Em geral, o *software* é de fácil utilização	1	3	3	
3.3.1	Tolera erros de operação do(s) usuário(s)	2	2	4	
3.3.2	Utiliza funcionalidades que induzem o funcionamento ao usuário	2	2	4	
3.4.1	Utiliza recursos gráficos atraentes aos usuários	2	2	4	
3.4.2	Presta informações adequadas aos usuários	2	2	4	
4.1.1	Tempo de resposta e processamento apropriados quando executa suas funções	2	2	4	
4.2.1	Utiliza diversos tipos e quantidades de recursos quando executa suas funções estabelecidas	2	1	2	
5.1.1	Permite o diagnóstico de deficiência ou causas de falhas, ou de partes a serem modificadas	2	1	2	
5.2.1	Permite implementar uma modificação especificada	2	2	4	
5.3.1	Evita efeitos inesperados decorrentes de modificações no *software*	2	3	6	
5.4.1	Permite avaliação do *software* quando modificado	2	3	6	
6.1.1	Funciona em diversos navegadores (*browsers*) sem a necessidade de configuração adicional ou instalação de outros recursos (*plug-in*)	2	3	6	
6.2.1	O *software* é de fácil instalação	2	2	4	
6.3.1	Funciona, sem problema, ao mesmo tempo com outros *softwares* que utilizam os mesmos recursos	2	3	6	
6.4.1	Permite atualização de versão mantendo o conteúdo do banco de dados existente	2	3	6	
			Pontuação Final =	**137**	

FONTE: O autor (2012)

Com os valores da avaliação das três ferramentas, foi possível partir para a próxima etapa do trabalho, onde se comparou os valores obtidos para se obter os resultados.

4 RESULTADOS

Com base nos valores obtidos na avaliação das três ferramentas foi possível fazer a comparação para então chegar-se a uma conclusão da ferramenta que oferece o melhor suporte ao desenvolvimento ágil baseado no método ágil *Scrum*.

	FIRESCRUM	AGILEFANT	KUNAGI	MÁXIMO POSSÍVEL
FUNCIONALIDADE	49	42	48	68
CONFIABILIDADE	11	16	16	16
USABILIDADE	22	26	27	30
EFICIÊNCIA	5	6	6	6
MANUTENIBILIDADE	14	14	18	18
PORTABILIDADE	18	20	22	22
PONTUAÇÃO TOTAL	**119**	**124**	**137**	**160**

QUADRO 3 – QUADRO COMPARATIVO FINAL

FONTE: O autor (2012)

O QUADRO 3 mostra os resultados para cada sistema separado pelas seis características, definidas pela norma NBR ISO/IEC 9126-1, e a sua pontuação final quanto ao nível de atendimento.

De um total máximo de 160 pontos que poderia ser obtido, a ferramenta *Firescrum* obteve 119, portanto, 74,37% de índice de atendimento contra 77,5% da ferramenta *Agilefant* e 85,63% da ferramenta *Kunagi.*

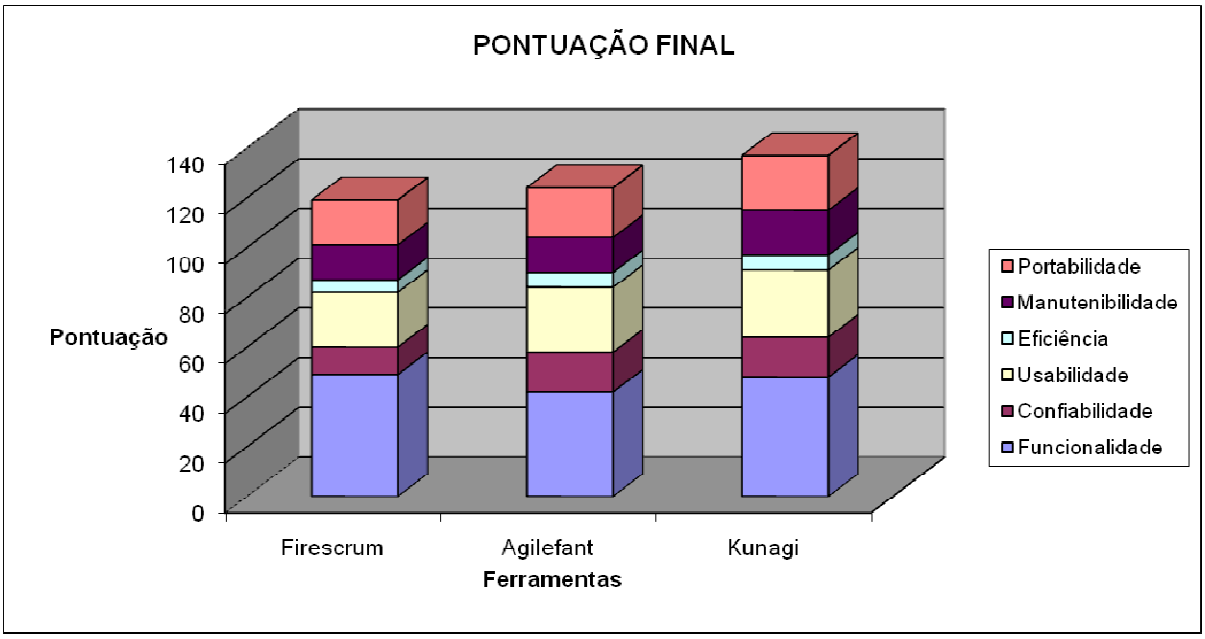


FIGURA 9 – GRÁFICO COMPARATIVO DA PONTUAÇÃO FINAL
FONTE: O Autor (2012)

A FIGURA 9 demonstra a comparação entre a pontuação final obtida por cada ferramenta. Nela é possível observar que a ferramenta *Kunagi,* obteve a maior pontuação (137), face aos critérios estabelecidos neste trabalho.

Embora a ferramenta *Kunagi* tenha sido a que atingiu a maior pontuação, curiosamente na característica funcionalidade esta ferramenta obteve um ponto a menos do que o último colocado, o *Firescrum.* Foram 48 pontos para *Kunagi* contra 49 pontos do *Firescrum*. É nesta característica que estão os critérios avaliados que mais estão relacionados aos princípios do *Scrum*, mas como para o resultado final desta avaliação o que importa é o resultado sintético de todas as características, a ferramenta *Kunagi* foi a que melhor ofereceu suporte ao desenvolvimento ágil baseado no método *Scrum.*

Kunagi também demonstrou ser uma ferramenta que possui muitas funcionalidades exigidas pelos princípios do *Scrum,* com ótimos recursos gráficos e de fácil instalação.

No segundo lugar, *Agilefant* aparece com 124 pontos, mostrando que, no geral, é outra ótima opção para quem deseja ter o suporte de uma ferramenta livre. *Agilefant* perdeu preciosos pontos, que lhe tiraram a liderança, devido ao fato de ser

a única a não possuir recurso para o *Planning Poker,* não possuir um meio de comunicação entre os usuários e também por não permitir criar e manter uma lista de impedimentos.

A ferramenta *Firescrum* foi a que obteve a menor pontuação dentre as três avaliadas. Esta ferramenta obteve a maior pontuação na característica funcionalidade, no entanto perdeu bons pontos em confiabilidade e usabilidade pois, embora tenha funcionalidades coerentes com os princípios ágeis do *Scrum*, ainda falta utilizar mais recursos gráficos atraentes ao usuário, como ícones, e também precisa melhorar a atualização de algumas telas conforme os dados são alterados.

Por fim, tem-se o resultado final, em ordem de colocação, para a avaliação comparativa de *softwares* livres para suporte ao desenvolvimento ágil baseado no método ágil *Scrum:*

1° colocado: *Kunagi*;
2° colocado: *Agilefant*;
3° colocado: *Firescrum*.

5 CONCLUSÃO

Após a realização deste trabalho pode-se afirmar que, dentre os três *softwares* avaliados, a ferramenta, disponível atualmente na *internet,* que melhor suporte oferece ao desenvolvimento de sistemas baseado no método ágil *Scrum,* é a ferramenta denominada *Kunagi*.

Vale ressaltar que esta conclusão se dá diante dos procedimentos de avaliação adotados por este trabalho, que foram baseados na norma NBR ISO/IEC 14598-1 e nas características de qualidade definidas pela norma NBR ISO/IEC 9126-1, considerando apenas os três *softwares* selecionados, sendo eles: *Agilefant*, *Kunagi* e *Firescrum*.

Para dar continuidade a esta conclusão convém recordar outro problema de pesquisa do presente trabalho: quanto um *software* livre atende aos princípios básicos do *Scrum* e aos requisitos de qualidade de *software?*

Observando-se os valores obtidos pela avaliação comparativa deste trabalho, pode-se observar que o menor índice de atendimento, obtido pela ferramenta *Firescrum*, foi de 74,37%, e o maior índice, obtido pela ferramenta *Kunagi,* foi de 85,63%, portanto é possível afirmar que, no mínimo, mais de 70% dos requisitos de qualidade de *software* são atendidos pelos *softwares* livres avaliados. Com índices altos como os obtidos, nem mesmo o último colocado da avaliação, o *Firescrum,* pode ser considerado um *software* de baixa qualidade e muito menos deixam a desejar quanto ao nível de atendimento aos princípios básicos a que se destinam.

Outro ponto interessante, observado no desenvolvimento deste trabalho, é o fato de perceber que a grande maioria dos *softwares* livres está sim em constante atualização e melhoria, como é o caso do *Kunagi,* primeiro colocado da avaliação. Em seu *site* é possível reportar à equipe de desenvolvedores possíveis problemas encontrados na ferramenta, verificar as melhorias que estão sendo implementadas e ter conhecimento da data prevista de disponibilização da próxima versão atualizada.

Diante das informações colocadas, pode-se afirmar que o motivo de o percentual de 56%, em média, de utilização de *softwares* livres nas empresas não ser maior não é o fato de não existir muitas opções para *download* na *internet*. Muitas ferramentas foram encontradas para a realização deste trabalho, mas apenas as três com o maior número de *downloads* foram consideradas, o que demonstra

que a comunidade está sempre procurando desenvolver novas soluções para a gestão de projetos de *software* baseados no *Scrum*.

Um ponto negativo para estas ferramentas é o fato de geralmente estar disponível em apenas um idioma, o inglês. Este pode ser um dos fatores que intimida quem deseja adquirir uma ferramenta livre e não tem facilidade com a língua inglesa.

Também foi possível verificar que ainda falta utilizar mais recursos gráficos, como imagens e ícones, para facilitar a assimilação das funcionalidades oferecidas e a utilização pelos usuários.

Em resumo, a baixa utilização de recursos gráficos e o não suporte a mais de um idioma, dentre eles o português, podem ser considerados fatores que influenciam a não utilização de *softwares* livres pelas organizações.

Por fim, pode-se afirmar, para aqueles que desejam adquirir um *software* livre para o suporte ao desenvolvimento baseado no método ágil *Scrum*, que *Kunagi* é a melhor ferramenta livre disponível atualmente para tal fim, pois obteve melhor colocação na avaliação comparativa realizada neste trabalho. *Agilefant,* segundo colocado, e *Firescrum,* terceiro colocado, também tiveram ótimos índices de atendimento, demonstrando ser possível utilizar qualquer uma dessas duas ferramentas, caso o primeiro colocado não esteja disponível, por qualquer que seja o motivo, no momento que for procurado na *internet.*

Espera-se, com o resultado deste trabalho, poder contribuir com os gerentes de projetos de *software,* seja encorajando-os a adquirir e usar uma ferramenta livre em sua organização, ou poupando tempo dos mesmos, na pesquisa por novas ferramentas, servindo de auxilio na tomada de decisões futuras e cooperando para o aumento do número de *softwares* livres nas organizações.

6 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Como sugestões para trabalhos futuros, desenvolvidos sobre o mesmo tema, ficam as seguintes colocações:

- Pesquisar a satisfação de uso de *softwares* livres nas organizações;
- Avaliar e comparar ferramentas para o suporte ao desenvolvimento abrangendo outras metodologias ágeis;
- Realizar um estudo de caso de implantação de um *software* livre de gestão de desenvolvimento ágil.

REFERÊNCIAS

ABNT. **Associação Brasileira de Normas Técnicas.** Disponível em: <http://www.abnt.org.br/m3.asp?cod_pagina=929>. Acesso em: 06/04/2012.

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. **Agile Software Development Methods:** Review and analysis. Espoo: VTT Publications, 2002. Disponível em: <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>. Acesso em: 17/03/2012.

AGILEFANT. Disponível em: <http://www.agilefant.org>. Acesso em: 14/10/2011.

AGILE MANIFESTO. Disponível em: <http://agilemanifesto.org/history.html> Acesso em: 17/03/2012.

AGILE PLANNING POKER | VIEW FROM THE POTTING SHED. Disponível em <http://blog.garethjmsaunders.co.uk/archives/2010/04/17/agile-planning-poker/>. Acesso em: 16/04/2012.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 9126-1**: Engenharia de Software – Qualidade de Produto – Parte 1: Modelo de qualidade. Rio de Janeiro, 2003.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 14598-1**: Tecnologia da Informação – Avaliação de produto de software – Parte 1: Visão Geral. Rio de Janeiro, 2001.

BARBI, F. C. **Os 7 passos do gerenciamento de projetos.** Disponível em: <http://www.microsoft.com/brasil/msdn/tecnologias/carreira/gerencprojetos.mspx>. Acesso em: 16/03/2012.

BECK, K. **Programação Extrema explicada:** acolha as mudanças, Porto Alegre: Bookman, 2004.

BRUZZI, D. G. **Gerência de Projetos.** Brasília: Editora Senac Distrito Federal, 2008.

CAVALCANTI, E. **Firescrum:** Ferramenta de Apoio à Gestão Ágil de Projetos utilizando Scrum**.** Centro de Estudos e Sistemas Avançados do Recife (CESAR), Recife, 2009.

FREE SOFTWARE FOUNDATION. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>. Acesso em: 17/03/2012.

GNU GENERAL PUBLIC LICENSE. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2012. Disponível em: <http://pt.wikipedia.org/w/index.php?title=GNU_General_Public_License&oldid=28634303>. Acesso em: 12/01/2012.

GNU PROJECT. Disponível em: <http://www.gnu.org/licenses/licenses.html>. Acesso em: 11/04/2012.

GOMES, N. da S. **Qualidade de Software – Uma Necessidade**, 2000. Disponível em <http://www.fazenda.gov.br/ucp/pnafe/cst/arquivos/Qualidade_de_Soft.pdf>. Acesso em: 05/04/2012.

HEPTAGON. **Tecnologia da Informação.** Disponível em <http://www.heptagon.com.br/kanban>. Acesso em: 19/04/2012.

HEXSEL, R. A. **Software Livre:** Propostas de ações de governo para incentivar o uso de software livre**.** Curitiba: [s.n.], 2002, 48 p. Disponível em <http://www.inf.ufpr.br/info/techrep/RT_DINF004_2002.pdf>. Acesso em: 10/04/2012.

HIRANABE, K. **Kanban applied to software development:** from agile to lean, 2008. Disponível em <http://www.infoq.com/articles/hiranabe-lean-agile-kanban>. Acesso em: 15/04/2012.

IEC. **International Electrotechnical Commission.** Disponível em <http://www.iec.ch/about>. Acesso em: 05/04/2012.

ISO. **International Organization for Standardization.** Disponível em <http://www.iso.org/iso/about.htm>. Acesso em: 05/04/2012.

KERZNER, H. **Gestão de projetos – As melhores práticas.** 2. ed. Porto Alegre: Artmed Editora S.A., 2004.

KERZNER, H. **Project management:** A systems approach to planning, scheduling and controlling. Ninth Edition. New York: John Wiley and Sons, 2006. Disponível em: <http://books.google.com.br/books?id=B1u9e0Dgx80C&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 16/03/2012.

KNIBERG, H; SKARIN, M. **Kanban e Scrum – obtendo o melhor de ambos.** C4 Media, 2009. Disponível em <http://www.infoq.com/minibooks/kanban-scrum-minibook>. Acesso em: 15/04/2012.

KUNAGI. **Agile Project Management based on Scrum**. Disponível em: <http://www.kunagi.org>. Acesso em: 21/03/2012.

LACY, M. K. B. Comparativo de gerenciadores de projetos: controle total. **Linux Magazine,** 18 ed., p. 46-51, 2006. Disponível em <http://www.konsultex.com.br/solucoes-livres/arquivos/gerenciamento-projetos-linux.pdf>. Acesso em: 10/04/2012.

LAKATOS, E. M.; MARCONI, M. de A. **Técnicas de Pesquisa:** planejamento e execução de pesquisas; amostragens e técnicas de pesquisa; elaboração, análise e interpretação de dados. 5 ed. São Paulo: Editora Atlas, 2002.

MARÇAL, A. S. C.; FREITAS, B. C. C.; SOARES, F. S. F.; MACIEL, T. M. M.; BELCHIOR, A. D. **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI**, 2007. Disponível em: <http://www.inf.ufes.br/~monalessa/PaginaMonalessa-NEMO/ES_Mestrado/Artigos/EstendendoScrumSegundoAreasDeGerenciaNoCMMI.pdf>. Acesso em: 17/03/2012.

MIT LICENSE. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2012. Disponível em: <http://en.wikipedia.org/w/index.php?title=MIT_License&oldid=483935005>. Acesso em: 18/04/2012.

MUNDOPM. **Pesquisa revela:** gestão de projetos assume papel estratégico nas empresas, 2008. Disponível em: <http://www.mundopm.com.br/noticia.jsp?id=262>. Acesso em: 15/03/2012.

OPEN SOURCE INITIATIVE. Disponível em <http://www.opensource.org/licenses/mit-license.php>. Acesso em: 11/04/2012.

PEDROSO, L. S. **Ferramentas e técnicas em *software* livre para o gerenciamento de projetos.** 55 f. Monografia (Especialização em Informática) – Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2008.

PFLEEGER, S. L. **Engenharia de software:** teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004. 537 p.

PLANNING POKER. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2012. Disponível em: <http://en.wikipedia.org/w/index.php?title=Planning_poker&oldid=480804209>. Acesso em: 16/04/2012.

PORTAL DA ADMINISTRAÇÃO. **Empresas Brasileiras Preferem Softwares Livres**, 2008. Disponível em: <http://www.administradores.com.br/informe-se/informativo/empresas-brasileiras-preferem-softwares-livres/14702/> Acesso em: 17/03/2012.

PRESSMAN, R. S.; LOWE, D. **Engenharia Web**. Rio de Janeiro: LTC, 2009. 416 p.

PROJECT MANAGEMENT INSTITUTE. Disponível em <http://www.pmi.org/PMBOK-Guide-and-Standards/Standards-Library-of-PMI-Global-Standards.aspx>. Acesso em: 09/04/2012.

PROJECT MANAGEMENT INSTITUTE SANTA CATARINA. Disponível em <http://www.pmisc.org.br/open.php?pk=18&id_ses=4>. Acesso em: 16/03/2012.

RIBEIRO, C. L. **A Relação entre TDD e Qualidade de Software**, 2010. Disponível em: <http://www.infoq.com/br/articles/relacao-tdd-qualidade>. Acesso em: 16/042012.

ROSITO, M. C.; CALLEGARI, D. A.; BASTOS, R. M. **Gerência de Projetos e Processos de Desenvolvimento de Software:** uma proposta de integração, 2007. Disponível em: <http://www.sbc.org.br/sbsi/2008/doc/SBSI2008ArtigoGerenciaProjetos.pdf>. Acesso em: 17/03/2012.

ROSITO, M. C.; CALLEGARI, D. A.; BASTOS, R. M. Project Management and Software Development Processes: Integrating PMBOK and OPEN. **World Academy of Science, Engineering and Technology**, New México, v. 62, n. 68, p. 375-383, fevereiro 2012. Disponível em: <http://www.waset.org/journals/waset/v62/v62-68.pdf>. Acesso em: 07/04/2012.

SANCHEZ, I. **Introdução ao Desenvolvimento Orientado a Testes (TDD)**, 2006. Disponível em: <http://dojofloripa.wordpress.com/2006/11/07/introducao-ao-desenvolvimento-orientado-a-testes>. Acesso em: 18/04/2012.

SOARES, M. S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Softwares. **Journal of Computer Science**, Minas Gerais, 2004a, Disponível em: <http://wiki.dcc.ufba.br/pub/Aside/ProjetoBitecIsaac/Met._Ageis.pdf>. Acesso em: 03/04/2012.

SOARES, M. S. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. **Revista Eletrônica de Sistemas de Informação**, Minas Gerais, 2004b, Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/viewFile/146/38>. Acesso em: 16/03/2012.

SOMMERVILLE, I. **Engenharia de Software.** 6. ed. São Paulo: Pearson Addison Wesley, 2003. 538 p.

SOURCEFORGE. Disponível em <http://sourceforge.net/projects/agilefant/files/stats/timeline?dates=2011-01-01+to+2011-12-31>. Acesso em: 19/03/2012a.

SOURCEFORGE. Disponível em <http://sourceforge.net/projects/firescrum/files/stats/timeline?dates=2011-01-01+to+2011-12-31>. Acesso em: 19/03/2012b.

SOURCEFORGE. Disponível em <http://sourceforge.net/projects/kunagi/files/stats/timeline?dates=2011-01-01+to+2011-12-31>. Acesso em: 19/03/2012c.

SUTHERLAND, J. **Agile Development in the Enterprise**, 2009. Disponível em: <http://jeffsutherland.com/AgileDevelopmentEnterprisev8.pdf>. Acesso em: 17/03/2012.

SUTHERLAND, J. **The Scrum Papers:** Nut, Bolts, and Origins of an Agile Framework, 2011. Disponível em: <http://jeffsutherland.com/ScrumPapers.pdf>. Acesso em: 15/04/2012.

TEST DRIVEN DEVELOPMENT. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2012. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Test_Driven_Development&oldid=29474685>. Acesso em: 16/04/2012.

VILLAS BOAS, A. L. de C. **Qualidade e Avaliação de Produto de Software.** Minas Gerais: UFLA/FAEPE, 2007, 141 p.