

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN
BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

VANDSON SANTOS DE LIMA

PROPOSTA DE UTILIZAÇÃO DO SCRUM COMO METODOLOGIA DE
DESENVOLVIMENTO NO NÚCLEO DE DESENVOLVIMENTO DE SOFTWARE DA
FARN

Monografia apresentada à Faculdade
Natalense para o Desenvolvimento do Rio
Grande do Norte - FARN, como parte dos

**PROPOSTA DE UTILIZAÇÃO DO SCRUM COMO METODOLOGIA DE
DESENVOLVIMENTO NO NÚCLEO DE DESENVOLVIMENTO DE
SOFTWARE DA FARN**

por: **Vandson Santos de Lima**
Cariacica

**NATAL/RN
2011**

VANDSON SANTOS DE LIMA

PROPOSTA DE UTILIZAÇÃO DO SCRUM COMO METODOLOGIA DE
DESENVOLVIMENTO NO NÚCLEO DE DESENVOLVIMENTO DE SOFTWARE DA
FARN

PROPOSTA DE UTILIZAÇÃO DO SCRUM COMO METODOLOGIA DE
DESENVOLVIMENTO NO NÚCLEO DE DESENVOLVIMENTO DE SOFTWARE DA
FARN

Monografia apresentada à Faculdade
Natalense para o Desenvolvimento do Rio
Grande do Norte – FARN, como parte dos
requisitos para obtenção do título de
Bacharel em Sistemas de Informação.

Orientador: Prof. Esp. Romulo Fagundes
Catanhede.

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Lima, Vandson Santos de

Proposta de utilização do scrum como metodologia de desenvolvimento
no núcleo de desenvolvimento de software da FARN / Vandson Santos de
Lima. – Natal, 2011.

50f.

Orientador: Rômulo Fagundes Catanhede (Esp.)

Monografia (Bacharelado em Sistemas de Informação) – Faculdade
Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Software - Monografia. 2. Projeto – Monografia. 3. Gerenciamento –
Monografia. 4. Desenvolvimento – Monografia. I.Título.

RN/FARN/BC

CDU 004.43

BIBLIOTECA UNI-RN

004.43 L732p - RB: 37240 (M002092)

Título: Proposta de utilização do SCRUM ...

Autoria: Lima, Vandson Santos de
2011, ex. 1

VANDSON SANTOS DE LIMA

**PROPOSTA DE UTILIZAÇÃO DO SCRUM COMO METODOLOGIA DE
DESENVOLVIMENTO NO NÚCLEO DE DESENVOLVIMENTO DE SOFTWARE DA
FARN**

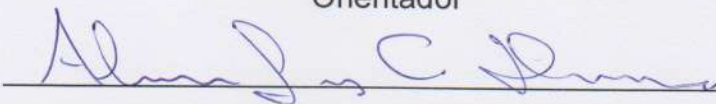
Monografia apresentada à Faculdade
Natalense para o Desenvolvimento do Rio
Grande do Norte – FARN, como parte dos
requisitos para obtenção do título de
Bacharel em Sistemas de Informação.

Aprovado em: ____/____/____

BANCA EXAMINADORA



Prof. Esp. Romulo Fagundes Catanhede
Orientador



Prof. Msc. Alexandre Luiz Galvão Damasceno
Membro examinador

Prof. Msc. Ricardo Wendell Rodrigues da Silveira
Membro examinador

NATAL/RN
2011

AGRADECIMENTOS

Agradeço ao Deus eterno, pela vida que me concedeu sem Ele nada sou. Porque tudo por Ele e para Ele são todas as coisas.

Aos meus pais Valdir Eloi e Maria Nélia que sempre estiveram torcendo por mim, pelo amor dedicado e por serem os meus maiores exemplos de vida em quem me espelho. Aos meus irmãos Nádja e Isaac pelo companheirismo e união.

Agradeço à minha esposa, amiga e companheira, Alisa Nerys, por seu amor, paciência, compreensão e incentivo em todos os momentos.

As minhas filhas Letícia e Gabriela, vocês sempre foram o meu grande incentivo para conquistar essa vitória.

Aos amigos que sempre aguçaram e motivaram a minha busca por esse tão importante momento.

A todos os professores que contribuíram para a minha formação.

Dedico este trabalho à minha esposa e
minhas filhas. Vocês são meu maior
incentivo.

AGRADECIMENTOS

Agradeço ao Deus eterno, pela vida que me concedeu; sem Ele nada sou. Porque dEle, por Ele e para Ele são todas as coisas.

Aos meus pais Valdir Eloi e Maria Neide que sempre estiveram torcendo por mim, pelo amor dedicado e por serem os meus maiores exemplos de vida em quem me espelho. Aos meus irmãos Nadja e Isaac pelo companheirismo e união.

Agradeço à minha esposa, amiga e companheira Nilse Nayara, por seu amor, paciência, compreensão e incentivo em todos os momentos.

Às minhas filhas Letícia e Gabriela, vocês sempre foram o meu grande incentivo para conquistar essa vitória.

Aos amigos que sempre ajudaram e motivaram a minha busca por este tão maravilhoso momento.

A todos os professores que contribuíram para a minha formação.

RESUMO

O presente trabalho pretende elaborar uma proposta de utilização da metodologia de desenvolvimento de software Scrum no núcleo de desenvolvimento de software da FARN e analisar as possíveis mudanças em relação ao processo de desenvolvimento de softwares. São evidenciadas as principais características do Scrum, adaptando-as ao núcleo de desenvolvimento da FARN. Para isso foram realizadas várias pesquisas sobre metodologias de software, iniciando pelas metodologias tradicionais, analisando as principais metodologias ágeis especialmente a metodologia Scrum devido à facilidade de adaptação para o ambiente em análise. Através de uma análise junto ao núcleo, foi possível identificar que o grupo necessita reavaliar os atuais procedimentos utilizados, para que assim, possa obter sucesso nos projetos relacionados ao desenvolvimento de softwares. A proposta de implantação da metodologia Scrum visa resolver alguns problemas encontrados no núcleo, bem como propor uma nova forma de conduzir os projetos, com prazos mais definidos, análise mais desenvolvida e documentação objetiva, com foco no resultado final e na satisfação para o cliente.

Palavras-Chave: Software. Projeto. Gerenciamento. Desenvolvimento.

LISTA ABSTRACT

This work intends to develop a proposal for use of the software development methodology Scrum at the core of software development FARN and analyze the possible changes in relation to the process of software development. Highlighting the main features of Scrum and adapt them to the core development FARN. For this, a research was conducted about different software methodologies. Starting with the traditional methodologies, analyzing especially the Scrum methodology due to the ease of adaptation to the environment under review. Through an analysis in the core, it was possible to identify what the group needs to review the current procedures used to succeed in projects related to software development. The proposed implementation of the Scrum methodology aims to address some problems found in the nucleus as well as providing a new way of doing projects with deadlines more defined, more developed analysis and objective documentation, focusing on the outcome and satisfaction for the customer.

Keywords: Software. Project. Management. Development.

LISTA DE IMAGENS

Figura 1: Modelo cascata.	14
Figura 2: O modelo espiral.	15
Figura 3: As quatro dimensões do XP.....	21
Figura 4: Práticas do XP.....	24
Figura 5: Ciclo de vida do FDD.	26
Figura 6: História sobre o comprometimento do Scrum Team.....	30
Figura 7: Exemplo de Sprint Backlog.	34
Figura 8: Exemplo de Scrum <i>Burndown Chart</i>	35
Figura 9: Ciclo de desenvolvimento com Scrum	38

LISTA DE TABELAS

1. INTRODUÇÃO	11
Tabela 1: Características da metodologia ágil XP.....	22
Tabela 2: Exemplo de <i>Product Backlog</i>	33
Tabela 3: Práticas e características da metodologia ágil Scrum.....	40
2.2 Objetivos específicos	13
3. REFERENCIAL TEÓRICO	14
3.1 Metodologias tradicionais de desenvolvimento de software	14
3.1.1 Modelo clássico	14
3.1.2 Modelo espiral.....	15
3.1.3 Falhas no Desenvolvimento Tradicional	16
3.2 Metodologias ágeis de desenvolvimento de software	17
3.2.1 Histórico do manifesto ágil	17
3.2.2 Princípios do desenvolvimento ágil de software	19
3.3 Abordagens da metodologia ágil	21
3.3.1 Extreme Programming	21
3.3.2 Feature Driven Development (FDD)	24
3.3.3 Metodologia Scrum	28
3.3.1 Papéis e responsabilidades	30
3.3.2 Artefatos	32
3.3.3 Eventos do Scrum	34
3.3.4 Ciclo de desenvolvimento	37
3.3.5 Ferramentas de gerenciamento Redmine	38
3.3.6 Aplicações práticas	39
4. ELABORAÇÃO DE PROPOSTA DE IMPLANTAÇÃO DO MÉTODO SCRUM	41
4.1 Caracterizações do ambiente	41
4.1.1 Processo de desenvolvimento	41
4.1.2 Equipe de desenvolvimento	42
4.2 Proposta de implantação do Scrum	43
4.2.1 Artefatos	43
4.2.2 Redefinição de papéis e responsabilidades	44
4.2.3 Definição de Eventos	45
4.3 Resultados esperados	46
5. CONSIDERAÇÕES FINAIS	47
REFERÊNCIAS	48

SUMÁRIO

1. INTRODUÇÃO.....	11
1.2 Justificativa.....	12
2. OBJETIVO.....	13
2.1 Objetivo geral.....	13
2.2 Objetivos específicos	13
3. REFERÊNCIAL TEÓRICO	14
3.1 Metodologias tradicionais de desenvolvimento de software.....	14
3.1.1 Modelo clássico	14
3.1.2 Modelo espiral.....	15
3.1.3 Falhas no Desenvolvimento Tradicional	16
3.2 Metodologias ágeis de desenvolvimento de software	17
3.2.1 Histórico do manifesto ágil	17
3.2.2 Princípios do desenvolvimento ágil de software.....	19
3.3 Abordagens da metodologia ágil	21
3.3.1 Extreme Programming.....	21
3.3.2 Feature Driven Development (FDD).....	24
3.3 Metodologia Scrum.....	28
3.3.1 Papéis e responsabilidades	30
3.3.2 Artefatos	32
3.3.3 Eventos do Scrum	35
3.3.4 Ciclo de desenvolvimento	37
3.3.5 Ferramenta de gerenciamento <i>Redmine</i>	38
3.3.6 Aplicações práticas.....	39
4. ELABORAÇÃO DE PROPOSTA DE IMPLANTAÇÃO DO MÉTODO SCRUM	41
4.1 Caracterizações do ambiente	41
4.1.1 Processo de desenvolvimento.....	41
4.1.2 Equipe de desenvolvimento.....	42
4.2 Proposta de implantação do Scrum.....	43
4.2.1 Artefatos	43
4.2.2 Redefinição de papéis e responsabilidades.....	44
4.2.3 Definição de Eventos	45
4.4 Resultados esperados.....	46
5. CONSIDERAÇÕES FINAIS	47
REFERÊNCIAS.....	48

1. INTRODUÇÃO

Nos últimos anos, o software tem sido um componente essencial a uma organização, principalmente como ferramenta de auxílio a tomadas de decisão em meio a um mercado extremamente competitivo. Segundo dados da consultoria *Gartner*, no ano de 2009 a receita obtida por fabricantes de software em todo o mundo foi de U\$ 224,4 bilhões. O mercado de software para empresas adicionou cerca de U\$ 10 bilhões em negócios no ano de 2010 e estima-se que em 2014 esse mercado movimentará U\$ 297 bilhões por ano (BARIFOUSE, 2010).

A globalização faz com que a concorrência entre empresas de software se torne muito acirrada. Por isso, atrasos, tolerância a falhas, softwares com tecnologias obsoletas e projetos cancelados estão cada vez menores, e as empresas precisam se enquadrar nesse novo contexto para poderem se destacar no mercado (SAVOINE ET AL., 2009).

Analisando o cenário de produção de software, é perceptível que os maiores problemas enfrentados por empresas de desenvolvimento de softwares são o aumento significativo do custo em relação ao custo inicial, alteração de cronograma e produtos de baixa qualidade que apresentam dificuldade de manutenção e frequentes disparidades. Para suprir tais necessidades, empresas utilizam de abordagens para o desenvolvimento de softwares que possam garantir uma qualidade dos seus produtos em um curto tempo.

A abordagem para o desenvolvimento de softwares existe há várias décadas e ao longo dos últimos 25 anos muitas surgiram, porém, apenas algumas sobrevivem e permanecem utilizadas nos dias atuais. Dentre várias abordagens utilizadas, o desenvolvimento ágil tem recebido grande destaque, principalmente no contexto de mudanças, oferecendo, em muitas situações, respostas rápidas às novas exigências de desenvolvimento: requisitos mutáveis e não totalmente esclarecidos e entrega do produto com valor aceitável. (LEITÃO, 2010).

O Scrum, objeto principal deste trabalho, destaca-se como sendo uma das principais metodologias ágeis. Segundo Schwaber (2006) o Scrum é um processo ágil ou *framework* para gerenciar projetos ágeis. Esse método é comumente utilizado em pequenas equipes e em ambientes onde é constante mudanças de escopo e de requisitos, sendo o foco a maximização da habilidade da equipe em dar respostas de forma ágil a mudanças e alterações. Visando evidenciar as virtudes e pontos críticos desse *framework*, elaboramos uma proposta para a utilização dessa metodologia em

um grupo de desenvolvimento, avaliando os resultados efetivamente obtidos com a utilização da metodologia de desenvolvimento ágil em foco.

2.1 Objetivo geral

1.2 Justificativa

O objetivo geral é elaborar uma proposta de utilização da metodologia de de-

A Justificativa para este trabalho parte da premissa de que o cenário atual de software necessita se adequar às exigências do mercado e apresenta o Scrum como uma solução, sendo aplicado dentro de um cenário real, analisando as atividades do grupo de desenvolvimento da Faculdade Natalense Para o Desenvolvimento do Rio Grande do Norte (FARN), caracterizando assim um estudo de caso.

Atualmente o grupo de desenvolvimento desempenha atividades de análise e desenvolvimento de sistemas utilizando diversas tecnologias. Porém não há nenhuma metodologia adotada para a realização de planejamentos, prazos, ciclos de atividades e definição de papéis desempenhados pelos integrantes do grupo. Isso resulta em um maior tempo para a execução das atividades, prazos e responsabilidades mal definidos, e um processo longo e desestimulante de atividades de análise e codificação do software.

De acordo com o cenário descrito, apresentamos uma proposta de solução através da implantação do Scrum, que deverá trazer excelentes resultados no que diz respeito a todo o processo de software e também à qualidade do produto final.

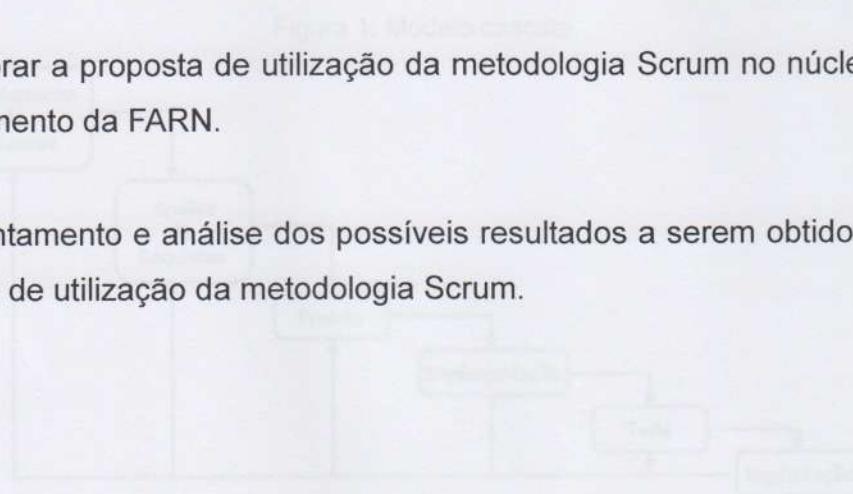
2. OBJETIVO TEÓRICO

2.1 Objetivo geral

O objetivo geral é elaborar uma proposta de utilização da metodologia de desenvolvimento de softwares Scrum em um cenário real e analisar as possíveis mudanças em relação ao processo de desenvolvimento de softwares.

2.2 Objetivos específicos

- Estudo das metodologias ágeis, com destaque para a metodologia Scrum.
- Análise do cenário atual do núcleo de desenvolvimento da FARN, destacando a necessidade de metodologias ágeis para o desenvolvimento de softwares.
- Elaborar a proposta de utilização da metodologia Scrum no núcleo de desenvolvimento da FARN.
- Levantamento e análise dos possíveis resultados a serem obtidos com a proposta de utilização da metodologia Scrum.



Fonte: Pressman (2002).

3. REFERÊNCIAL TEÓRICO

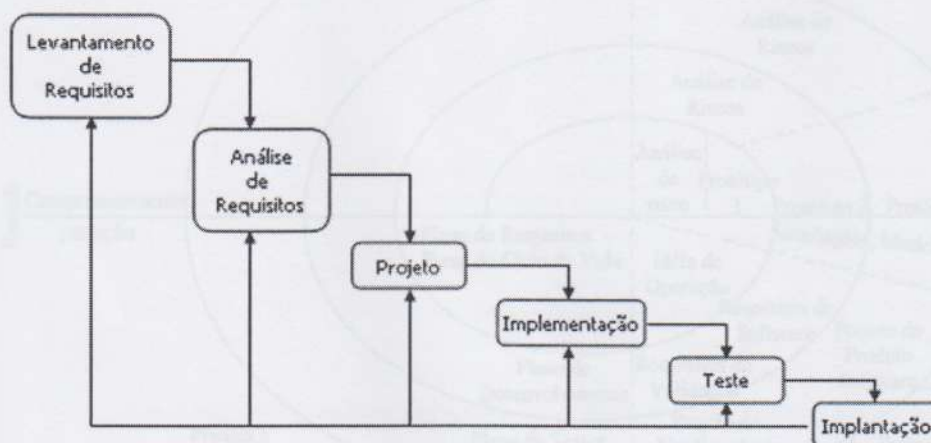
3.1 Metodologias tradicionais de desenvolvimento de software

Neste trabalho, utilizamos o termo metodologias tradicionais para identificar metodologias de desenvolvimento que há várias décadas são utilizadas nos mais diversos processos de desenvolvimento de software, e ainda continuam sendo os mais empregados atualmente.

3.1.1 Modelo clássico

O modelo clássico “é um paradigma que utiliza um método sistemático e seqüencial, em que o resultado de uma fase se constitui na entrada de outra, também conhecido como modelo *cascata* (...)” (CARVALHO e CHIOSSI, 2001, P.29).

Figura 1: Modelo cascata.



Fonte: Pressman (2007).

Esse modelo trouxe ainda várias contribuições importantes para o processo de desenvolvimento de software, sendo as principais, o processo de desenvolvimento de software deve ser sujeito à disciplina, planejamento e gerenciamento; e a implementação do produto deve ser postergada até que os objetivos tenham sido completamente atendidos.

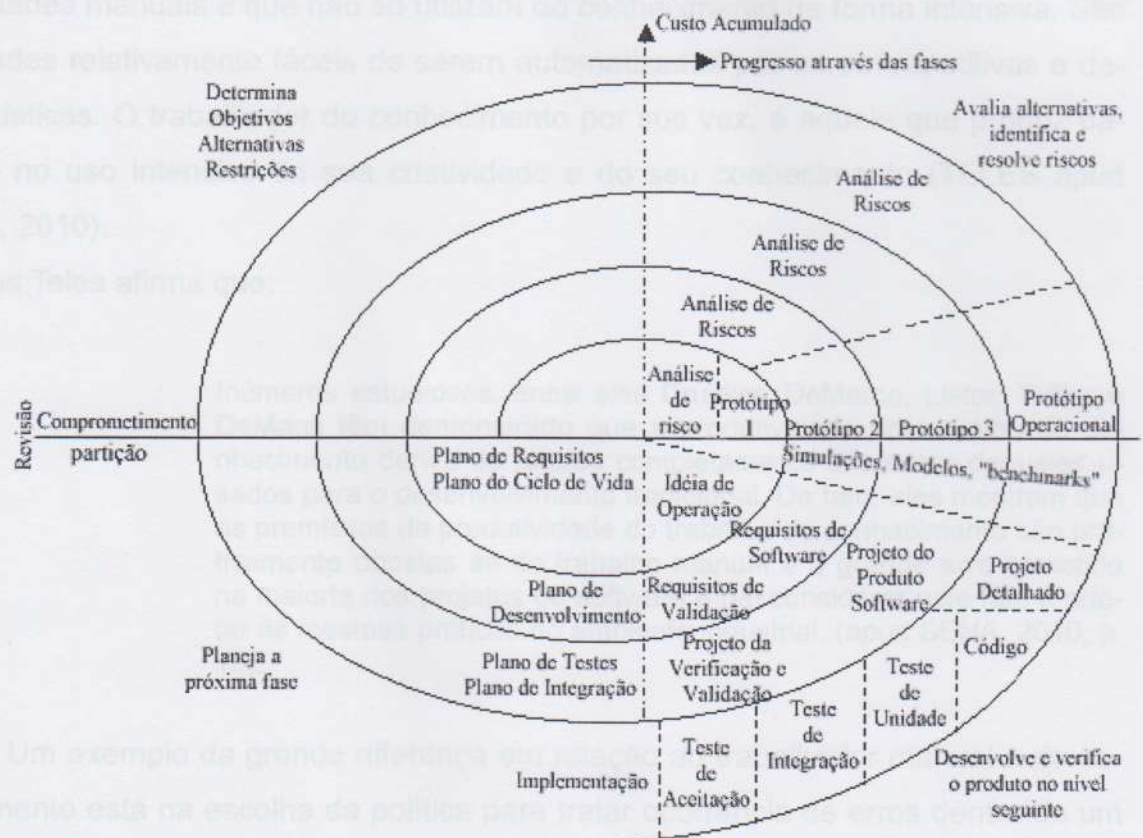
Por outro lado, existem vários problemas com essa metodologia, dentre eles, é que os projetos que a utilizam raramente seguem o fluxo seqüencial proposto pelo modelo, pois sempre é necessário utilizar iterações que trazem problemas na aplicação do paradigma. Outro problema é que o modelo clássico requer que o cliente declare todas as exigências explicitamente. O modelo possui dificuldades até mesmo

de acomodar a incerteza natural que existe no início de vários projetos. Por fim, o cliente necessita ter paciência, pois uma versão, ou protótipo do produto só estará disponível num ponto tardio do cronograma do projeto (PRESSMAN, 2007).

3.1.2 Modelo espiral

O modelo espiral surgiu em 1988 e propõe uma organização em espiral composta por diversos ciclos. Esse modelo também abrange as melhores características do modelo cascata, acrescentando um novo elemento – a análise de riscos – que falta a esse paradigma (PRESSMAN, 2007).

Figura 2: O modelo espiral.



Fonte: PRESSMAN (2007)

A figura 2 apresenta quatro importantes atividades representadas pelos quadrantes da figura. Cada ciclo na espiral inicia com a identificação dos objetivos e descrição de alternativas para atingir os objetivos propostos assim como as restrições impostas. O próximo passo no ciclo é a avaliação das diferentes alternativas com base nos objetivos, o que vai permitir também definir incertezas e riscos de cada alternativa. No passo seguinte, o desenvolvimento de estratégias permitindo resolver ou eliminar as incertezas levantadas anteriormente, o que pode envolver ativi-

dades de prototipação, simulação, avaliação de desempenho, etc. Finalmente, o software é desenvolvido e o planejamento dos próximos passos é realizado (MAZZOLA, 2010).

3.1.3 Falhas no Desenvolvimento Tradicional

Segundo Sena (2010) “A falha no desenvolvimento tradicional é que ele baseia-se em premissas que não se aplicam ao trabalho de desenvolvimento de software”. Para Drucker (DRUCKER apud SENA, 2010, P.18), existem duas categorias de trabalhadores: o trabalhador manual e o trabalhador do conhecimento. O trabalhador manual caracteriza-se por efetuar trabalhos que dependem basicamente de habilidades manuais e que não se utilizam do conhecimento de forma intensiva. São atividades relativamente fáceis de serem automatizadas por serem repetitivas e determinísticas. O trabalhador do conhecimento por sua vez, é aquele que produz baseado no uso intensivo da sua criatividade e do seu conhecimento (TELES apud SENA, 2010).

Vinicius Teles afirma que:

Inúmeros estudiosos, entre eles Drucker, DeMarco, Lister, Toffler e DeMarsi têm demonstrado que a produtividade do trabalho do conhecimento deriva de fatores completamente diferentes daqueles usados para o desenvolvimento tradicional. De fato, eles mostram que as premissas da produtividade do trabalho do conhecimento são praticamente opostas às do trabalho manual e o grande erro cometido na maioria dos projetos de software é desconsiderar este fato e adotar as mesmas práticas do ambiente industrial. (apud SENA, 2010, p. 18).

Um exemplo da grande diferença em relação ao trabalhador manual e do conhecimento está na escolha da política para tratar ocorrência de erros dentro de um projeto. Para projetos que envolvem trabalhadores manuais, os erros devem ser evitados sempre, e os processos devem ser rígidos e determinísticos para que auxiliem na busca por este objetivo. Já para projetos em que participam trabalhadores do conhecimento, erros devem ser encarados de forma natural, saudável e até inevitáveis. Neste caso, tentar sistematizar ou impor metodologias rígidas ao processo de desenvolvimento de softwares, visando o determinismo, torna as pessoas envolvidas defensivas e pouco criativas. (TELES apud SENA, 2010).

3.2 Metodologias ágeis de desenvolvimento de software

As metodologias ágeis de desenvolvimento de software foram criadas como uma alternativa aos processos tradicionais, suprimindo a necessidade de desenvolvimento mais flexível a mudanças e menos custoso em relação aos métodos tradicionais que necessitam de artefatos de projeto tão completos e precisos quanto possível.

A proposta dessa metodologia, para ser ágil, é aceitar mudança em vez de tentar prever o futuro. De acordo com Teles (2008), este novo conceito de metodologia surgiu nos Estados Unidos na década de 90, na tentativa de criar sistemas em um menor tempo, com uma qualidade superior e de forma mais econômica que o habitual. Esses objetivos podem ser alcançados através de um conjunto de valores, princípios e práticas que diferem essencialmente da metodologia tradicional de se desenvolver software (PAULA, 2010).

Para Cisneiros (2009), nos modelos de Desenvolvimento Ágil, o objetivo principal é desenvolver um produto rapidamente e com qualidade. Para isto, as metodologias ágeis se caracterizam por um gerenciamento de projeto em que um líder de time esteja sempre organizando, inspecionando, apoiando e garantindo que o time esteja bem, ao mesmo tempo em que os resultados do projeto de software vão atendendo as necessidades do cliente.

3.2.1 Histórico do manifesto ágil

Em fevereiro de 2001, um grupo de cerca de 17 metodologistas fundou a *Agile Software Development Alliance*, mais conhecida como *Agile Alliance* (www.agilealliance.org) com o objetivo de compartilhar valores e princípios que utilizavam na prática, e abolindo métodos tradicionais que não traziam benefícios. Este grupo de pessoas definiu um manifesto para encorajar melhores meios para se desenvolver software. Trata-se de um conjunto de princípios que definem critérios para o processo de desenvolvimento de software ágil, que seguem:

1. Indivíduos e iterações valem mais que processos e ferramentas.

Com esse princípio, o movimento ágil destaca a valorização do indivíduo e de suas relações dentro de um projeto. É importante ressaltar que os fatores mais importantes são as pessoas e como elas trabalham juntas. Sem esse capital humano, um projeto não será bem sucedido por melhores processos que se tenha adotado, ou por melhores ferramentas que se tenha adquirido (AMBLER, 2002).

2. Um software funcionando vale mais que uma documentação extensa

A proposta é priorizar o desenvolvimento do software, de forma que a cada ciclo de atividades se tenha algo funcional. Para Ambler (2002, p.24) “a documentação tem o seu lugar. Escrita de maneira correta, torna-se um guia importante para as pessoas entenderem como e por que o sistema é construído e como trabalhar com ele. Entretanto nunca esqueça de que o objetivo principal do desenvolvimento de software é criar softwares, não documentos.”

3. Colaboração do cliente vale mais que a negociação de contrato

Embora seja declarada a importância de contratos bem elaborados, o movimento ágil enfatiza que o relacionamento e a cooperação entre clientes e desenvolvedores são preferenciais sobre os contratos restritivos (LEITÃO, 2010). Para Ambler (2002, p. 24) os desenvolvedores de sucesso trabalham próximos aos clientes, se esforçam para detectar suas necessidades e os educam durante este processo.

4. Mudanças valem mais que seguir um plano

Mudanças fazem parte do projeto de desenvolvimento de software. As pessoas mudam suas prioridades por vários motivos. À medida que o trabalho progride, os clientes mudam sua compreensão sobre o domínio do problema e sobre o que está sendo desenvolvido. Além disso, o próprio ambiente de negócio sofre alterações. Não há nada de errado em se ter um plano de projeto, porém, este plano deve ser flexível, e estar apto a receber mudanças que possam ocorrer durante o projeto (AMBLER, 2002).

As metodologias ágeis focam na honestidade ao código de trabalho, eficácia das pessoas que trabalham em conjunto e foco no trabalho em equipe. Isso mostra que a participação de representantes do cliente é essencial na formação do grupo de desenvolvimento, que possui autonomia e competência para eventuais ajustes das necessidades emergentes durante o processo do ciclo de vida do desenvolvimento. Isto significa que todos os participantes (desenvolvedores e representantes de clientes) estão preparados para fazer mudanças quando necessários. Além disso, os contratos existentes são formados com ferramentas que suportam e permitem que essas melhorias sejam realizadas (LEITÃO, 2010, p. 20).

3.2.2 Princípios do desenvolvimento ágil de software

Os membros da *Agile Alliance* criaram coleção de doze princípios, refinando as filosofias contidas no manifesto, a fim de que as pessoas compreendam melhor do que se trata o desenvolvimento ágil de software. Estes princípios são:

1. Nossa maior prioridade é satisfazer ao cliente mediante entregas de software de valor em tempo hábil e continuamente.
2. Receber bem mudanças de requisitos, mesmo em uma fase mais avançada no desenvolvimento. Os processos ágeis direcionam as mudanças para obter vantagens competitivas para o cliente.
3. Entregar softwares em funcionamento com frequência de algumas semanas a alguns meses, de preferência na menor escala de tempo.
4. As equipes de negócios e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.
5. Construa projetos ao redor de indivíduos motivados. Dê-lhes o ambiente e o apoio de que eles precisam e confie neles para realizar o trabalho.
6. O método mais eficiente de levar informações para uma equipe de desenvolvimento e fazê-las circular é a conversa cara a cara.

7. Ter o software funcionando é a principal medida de progresso.
8. Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários deveriam ser capazes de manter um ritmo constante indefinidamente.
9. Atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade.
10. Simplicidade - a arte de maximizar a quantidade de trabalho não-realizado - é essencial.
11. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.
12. Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz e então se ajustar e adaptar seu comportamento.

Segundo Ambler (2002, p. 25):

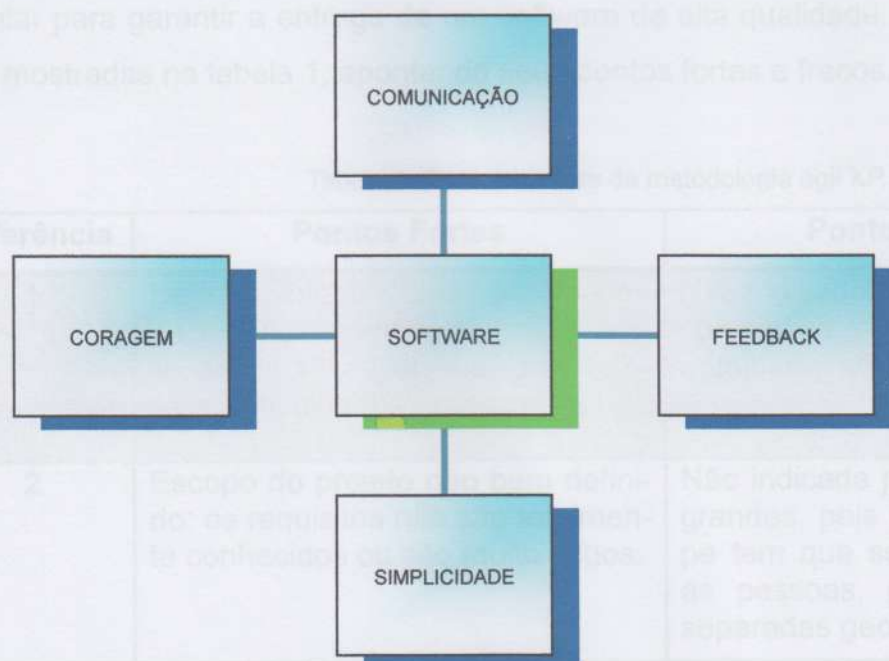
Creio que esses princípios formam os alicerçares do bom senso nos quais você pode basear seu trabalho de desenvolvimento de software bem-sucedido. Além disso, acredito que eles definem requisitos de alto nível para uma metodologia eficaz de software, usados para formular os valores, princípios e as práticas da modelagem ágil.

3.3 Abordagens da metodologia ágil

3.3.1 Extreme Programming

Proposto em 1996 por Kent Beck, o *Extreme Programming* (XP) foi baseado em suas experiências de desenvolver software. Pensando em fatores que tornassem o software fácil de ser criado e fatores que tornavam essa criação difícil. Sendo assim, através de um projeto na *DaimlerChrysler*, o *Extreme Programming* surgiu baseado em quatro dimensões que o software pode ser sempre melhorado (Figura 3). A comunicação é essencial e deve ser aprimorada constantemente; a simplicidade deve ser considerada primordial; é necessário obter *feedback* de como as atividades estão sendo executadas e se os resultados adquiridos estão sendo satisfatórios; e é necessário sempre ter coragem (PRIKLADNICKI; AUDY, 2007, p. 25).

Figura 3: As quatro dimensões do XP.



A comunicação ocorre de forma contínua entre o cliente e a equipe de desenvolvimento. Dessa forma o cliente tem uma maior aproximação e acompanha bem próximo o progresso do desenvolvimento. Através dessa interação o cliente tem a tarefa de decidir o que se deve ser construído e em que ordem deve ocorrer. Com relação à simplicidade, é utilizado o menor número de artefatos possíveis que não sejam códigos, e estes por sua vez devem estar sempre sendo revisados e corrigi-

dos. O software é liberado em pequenas versões que são testadas continuamente e através do *feedback* são avaliadas certificando a eficiência do código (PRIKLAD-NICKI; AUDY, 2007).

A coragem diz respeito a estilos e comportamentos dos integrantes da equipe, diante de situações que lhes tragam insegurança. Ter coragem implica em encarar os desafios, agir de forma correta e honesta distinguindo o que pode e o que não pode ser feito.

A XP concentra esforços da equipe de desenvolvimento em atividades que produzem resultados rapidamente na forma de software intensamente testado e totalmente alinhado às necessidades de seus usuários. Além disso, propõe a simplificação e organização do trabalho, combinando técnicas comprovadamente eficazes e eliminando atividades redundantes. Por fim, proporciona a redução de riscos nos projetos através da interatividade e constante avaliação das prioridades do cliente (TELES apud SENA, 2010).

A XP enfatiza o uso de 12 práticas, utilizadas de forma interativa e complementar para garantir a entrega de um software de alta qualidade. Essas 12 práticas são mostradas na tabela 1, apontando seus pontos fortes e fracos.

Tabela 1: Características da metodologia ágil XP.

Referência	Pontos Fortes	Pontos Fracos
1	Reuniões diárias: tiram-se dúvidas de histórias complexas, planejam-se as iterações do dia, discutem-se as dificuldades encontradas.	Não indicada para sistemas complexos, pois a análise não é detalhada.
2	Escopo do projeto não bem definido: os requisitos não são totalmente conhecidos ou são muito vagos.	Não indicada para equipes muito grandes, pois o escopo da equipe tem que ser bem definido, e as pessoas, geralmente, estão separadas geograficamente.
3	Foco na codificação, sendo que a documentação fica para segundo plano.	Não foca muito a gerência do projeto, o que interessa a equipe são os resultados, o código rodando.
4	Cliente presente é a chave para o sucesso.	Não é recomendada em ambientes onde os clientes não podem estar presentes.

5	Flexível a mudanças encoraja as pessoas a enfrentarem com naturalidade.	Não é recomendado em projetos que o cliente exige uma extensa documentação do software.
6	Simplicidade: executar tarefas com simplicidade e qualidade, desenvolver funcionalidades realmente útil ao usuário.	Não recomendada em equipes que possuem resistências às práticas como <i>refactoring</i> , programação pareada, etc.
7	Iterações curtas, com isso o <i>feedback</i> do usuário é rápido e constante, realimentando o desenvolvimento das funcionalidades.	Não indicada em projetos em que o cliente não possa participar integralmente do desenvolvimento.
8	Os testes fazem parte do ciclo de desenvolvimento, aumentando assim a qualidade do software.	Se o <i>feedback</i> entre o cliente e a equipe não for constante, ou houver uma demora de respostas entre as partes, a XP não será produtiva.
9	Integração diária do código produzido.	Quando houver resistência em modificar o código já concluído, com o objetivo de torná-lo simples, estruturado.
10	Código coletivo: todos são responsáveis, e aptos a realizarem melhorias, caso necessário.	Não é recomendada em ambientes de trabalhos exaustivos, a XP enfatiza o trabalho de 8 horas diárias.
11	Padronização do código: evitar problemas na hora da refatoração e da manutenção.	Não indicada para projetos que necessitem de ciclos de iterações longos;
12	Refatorar: simplificar o código, sem alterar sua funcionalidade.	A XP não é uma metodologia flexível em relação às suas práticas, que devem ser seguidas rigorosamente.

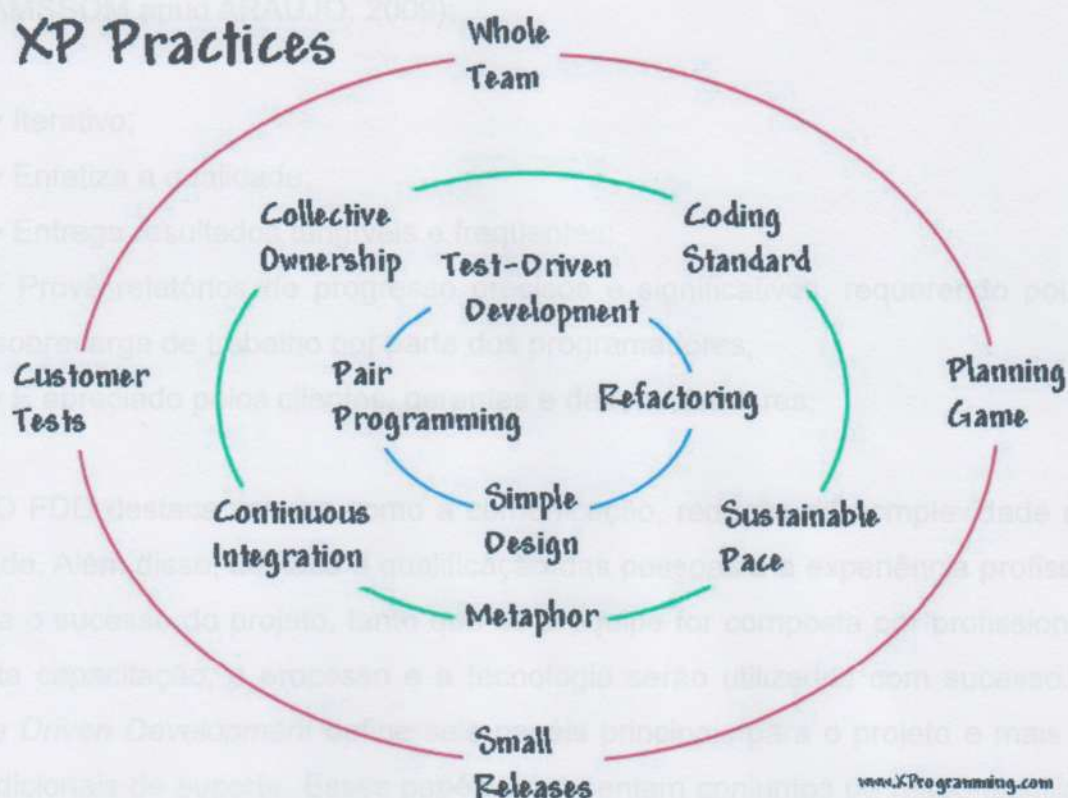
Fonte: adaptado de Savoine et al. (2009)

3.3.2 Feature Driven Development (FDD)

As práticas apóiam umas às outras. Pontos fortes compensam pontos fracos de outras. Cada item baseia-se em simplicidade. A riqueza está na interação entre os itens. (SAVOINE et al., 2009). A figura 4 abaixo também representa as práticas da metodologia XP.

Desenvolvedores. Peter Guad, definiu a ideia de Feature Driven Development e Feature List. Participaram da concepção das ideias do FDD renomados autores, entre eles: Tom De Marco, Tim Lister, Jerry Weinberg e Fredrick Brooks. Em sua es-

Figura 4: Práticas do XP.



Fonte: Xprogramming.com

No entanto, nem todos os limites da metodologia XP foram identificados, por isso a metodologia não é apropriada para ser utilizada em qualquer ambiente. Essas definições exigem mais pesquisas empíricas e experimentais a partir de diferentes perspectivas. (LEITÃO, 2010)

Um dos limites identificados na XP é o fato de ser voltado para pequenas e médias equipes apesar de se ter registros de sucesso em vários projetos de grande porte. Além disso, Beck (apud Leitão, 2010) também enfatiza a importância do ambiente em XP, visto que a comunicação e a coordenação entre os membros devem ser constante em um projeto. Também menciona que uma dispersão de programadores em dois andares ou até mesmo em um piso é intolerável para a XP.

3.3.2 Feature Driven Development (FDD)

O FDD é uma simples compilação de práticas estabelecidas nos últimos 30 anos. Criado em 1990 por um time liderado por Jeff De Luca em Cingapura. Um de seus maiores desenvolvedores, Peter Coad, definiu a ideia de *Feature Definition* e *Feature List*. Participaram da concepção das ideias do FDD renomados autores, entre eles: Tom De Marco, Tim Lister, Jerry Weinberg e Frederic Brooks. Em sua es-

sência o FDD é um processo ágil e adaptativo com as seguintes características (ABRAHAMSSON apud ARAUJO, 2009):

- Iterativo;
- Enfatiza a qualidade;
- Entrega resultados tangíveis e freqüentes;
- Provê relatórios de progresso precisos e significativos, requerendo pouca sobrecarga de trabalho por parte dos programadores;
- É apreciado pelos clientes, gerentes e desenvolvedores;

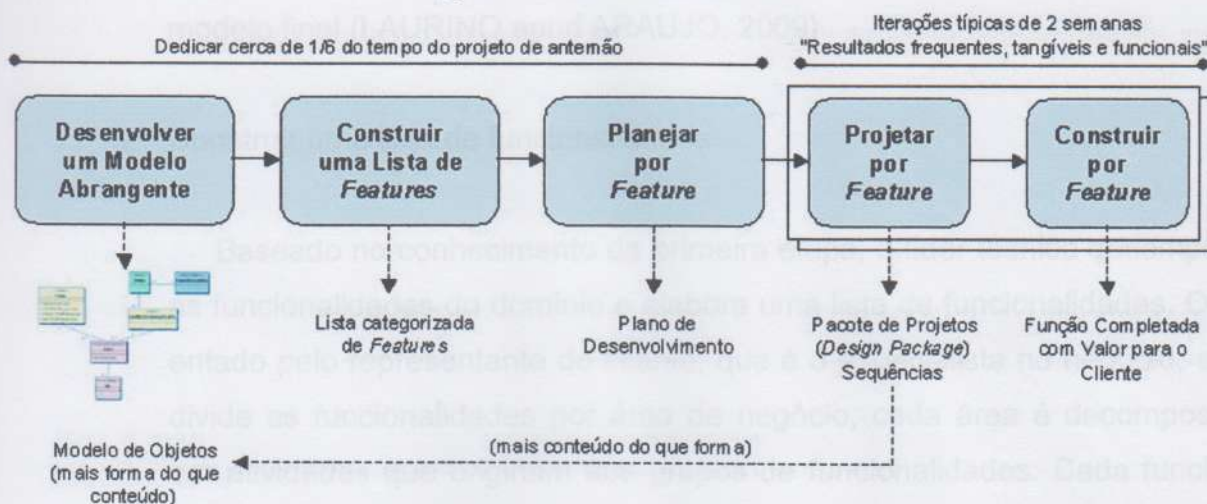
O FDD destaca valores como a comunicação, redução da complexidade e a qualidade. Além disso, destaca a qualificação das pessoas e a experiência profissional para o sucesso do projeto, tanto que se a equipe for composta por profissionais com alta capacitação, o processo e a tecnologia serão utilizados com sucesso. O *Feature Driven Development* define seis papéis principais para o projeto e mais alguns adicionais de suporte. Esses papéis representam conjuntos de responsabilidades que um profissional assume dentro de um projeto. De acordo com a complexidade de um projeto e da capacitação do profissional, é possível atribuir-lhe mais de um papel (PALMER; FELSING, apud ARAUJO, 2009).

Quando a equipe já possui informações suficientes quanto ao negócio ou domínio do problema e quanto aos processos que precisam ser automatizados, o FDD dá início a seu processo. Sua primeira etapa consiste na criação de um modelo de classes e objetos do negócio, desenvolvido junto a pessoas que possuem conhecimento especializado e pleno domínio acerca do problema. O segundo passo é desenvolver uma lista de funcionalidades ou *features* que o novo sistema deverá apresentar; este item é elaborado com base no conhecimento adquirido durante o primeiro processo de modelagem e captura de requisitos funcionais (PALMER; FELSING, apud ARAUJO, 2009).

Na sequência, um plano global do projeto é esboçado e suas responsabilidades são divididas entre os integrantes da equipe. O projeto é conduzido através de interações, que serão executadas em paralelo ou em série. Cada grupo de desenvolvedores trabalha em uma interação e aborda um pequeno conjunto de funcionalidades, trabalhando em sua construção e em seu projeto técnico. Cada uma deve ser construída em um prazo máximo de duas semanas; se a estimativa de tempo for superior a esta, ela deve ser dividida em menores funcionalidades. O projeto é con-

duzido dessa forma até que sejam desenvolvidas todas as funcionalidades. (MARTINS apud ARAUJO, 2009). Veja a **figura 5**.

Figura 5: Ciclo de vida do FDD.



Fonte: adailmr.sites.uol.com.br

Cinco processos definem o método FDD. O projeto começa com a modelagem das classes do domínio do problema, em seguida a tabulação das funcionalidades que o sistema deve disponibilizar e a seguir um planejamento de como serão implementadas tais funcionalidades. Os dois processos seguintes são onde a programação acontece efetivamente. O líder técnico seleciona um pequeno conjunto de funcionalidades para que seja desenvolvido nos próximos dias, com prazo máximo de duas semanas, identifica os donos das classes envolvidas e os convoca a integrar a equipe para a iteração. A equipe trabalha na modelagem dinâmica das funcionalidades e escrevem as estruturas das classes. Após este processo, a equipe faz uma inspeção no projeto técnico, e os desenvolvedores codificam as classes, fazem testes de unidade, integração e inspeção do código. A satisfação dos líderes com o trabalho desenvolvido inicia-se a liberação dos programas gerados para compor a versão candidata para o ambiente de produção. Os processos da metodologia FDD são descritos a seguir (MARTINS apud ARAUJO, 2009):

1. Desenvolver um modelo geral

Neste processo o especialista acerca do problema e a equipe de desenvolvimento trabalham juntos e executam uma análise de alto nível acerca do escopo do sistema e de seu contexto. O domínio do problema é

dividido pela equipe em áreas que realizam vários *walkthroughs* em cada uma dessas áreas, cada vez aumentando o nível de detalhe. À medida que o trabalho é desenvolvido a equipe constrói o modelo de classes e objetos do negócio. Depois todos os modelos são combinados para formar o modelo final (LAURINO apud ARAUJO, 2009).

2. Construir uma lista de funcionalidades

Baseado no conhecimento da primeira etapa, o líder técnico decompõe as funcionalidades do domínio e elabora uma lista de funcionalidades. Orientado pelo representante do cliente, que é o especialista no negócio, ele divide as funcionalidades por área de negócio, cada área é decomposta em atividades que originam aos grupos de funcionalidades. Cada funcionalidade deve pertencer apenas a um conjunto (atividade) e a somente uma área do negócio. As funcionalidades são organizadas numa *Feature Breakdown Structure* (FBS) onde são agrupadas em conjunto e separadas por área de negócio. (MARTINS apud ARAUJO, 2009).

3. Planejar por funcionalidade

No terceiro processo do método FDD o gerente de projeto, o gerente de desenvolvimento e o líder técnico, em conjunto, planejam a sequência de desenvolvimento dos grupos de funcionalidades (atividades de negócio) e dos conjuntos de funcionalidades (áreas de negócio) (FDD, 2009).

4. Construir por funcionalidade

Usando as especificações realizadas no projeto técnico, os proprietários das classes fazem a programação. Depois o código-fonte passa pelo teste de unidade e inspeção de código, mas fica evidenciado que nem todo código necessariamente passa pela inspeção. Há situações em que cenários simples são repetidos por várias partes do sistema, nestes casos a inspeção de apenas uma amostragem é suficiente para localizar grande parte dos problemas. Por fim, as funcionalidades desenvolvidas e testadas são integradas à versão final (MARTINS apud ARAUJO, 2009).

O FDD é definido através de um conjunto de práticas que embora não sejam inovadores, trazem uma forma inovadora de serem utilizadas: elas reforçam e complementam umas as outras. Essa metodologia pode ser implementada somente através de algumas práticas, mas para se ter todos os benefícios previstos pelo FDD, deve-se considerar todas elas. As práticas propostas pela metodologia são: modelagem de objetos do domínio, desenvolvimento por funcionalidade, código-fonte de cada classe pertence unicamente a um programador apenas. Equipes montadas com base nas funcionalidades, inspeções, *builds* constantes, gerenciamento de configuração do progresso (LAURINDO apud ARAUJO, 2009).

Segundo Araujo (2007, p. 35) “o FDD é um método simples, objetivo e fácil de seguir, possui técnicas objetivas para lidar com os problemas de desenvolvimento de software, técnicas claras para informar o progresso e permitir que decisões possam ser tomadas em tempo oportuno”. Entretanto, pode não se aplicar em ambientes que necessitem de constantes mudanças de escopo, ou com clientes que necessitem de soluções que exijam menos da sua participação.

3.3 Metodologia Scrum

O método Scrum surgiu de um artigo que sumarizam as 10 melhores práticas em empresas japonesas, escrito por Takeuchi e Nonaka, que tem por título “O jogo do desenvolvimento de novos produtos” que foi publicado na *Havard Business Review* em Janeiro de 1986. Neste artigo, foi introduzido o termo SCRUM, utilizado nos jogos de *Rugby*, quando os jogadores se organizam em círculo para planejar a próxima jogada. Uma forma de mostrar que o projeto deve ser conduzido por pequenos ciclos, mas com uma visão de longo prazo, que é vencer o jogo (ARAUJO, 2009).

O Scrum foi criado por Ken Schwaber e Jeff Sutherland, e surgiu da necessidade de encontrar uma metodologia capaz de abordar o problema de desenvolvimento de software de uma forma não tradicional. Segundo Zanatta (2004, p. 46) o Scrum tem como objetivo definir um processo para aplicá-lo a um projeto de desenvolvimento de software orientado a objeto, que tenha seu foco nas pessoas integrantes do projeto e que seja útil em ambientes que exigem rápidas respostas às mudanças de requisitos que surgirem durante todo o projeto.

Segundo Schwaber (2009):

Scrum não é um processo ou uma técnica para o desenvolvimento de produtos. Ao invés disso, é um *framework* dentro do qual você pode empregar diversos processos e técnicas. O papel do Scrum é fazer transparecer a eficácia relativa das suas práticas de desenvolvimento para que você possa melhorá-las, enquanto provê um *framework* dentro do qual produtos complexos podem ser desenvolvidos (Schwaber, 2009, p.3).

A aplicação do Scrum proporciona agilidade necessária em projetos adaptados às novas realidades organizacionais em ambientes caracterizados por mudanças constantes. Além disso, o Scrum possui o foco de proporcionar ampla comunicação da equipe e cooperação dos envolvidos no projeto. (LEITÃO, 2010).

A metodologia ágil Scrum é sustentada por três pilares: transparência, inspeção e adaptação. A **transparência** garante que os aspectos do processo que afetam diretamente o resultado devem ser visíveis para os que administram os resultados (SCHWABER; SUTHERLAND, 2011).

Os aspectos de um processo devem ser **inspecionados** freqüentemente, de forma a detectar e evitar variações inaceitáveis no processo. A freqüência de inspeção tem que considerar que todos os processos são alterados através da inspeção (SCHWABER; SUTHERLAND, 2011).

Dessa forma, caso o inspetor determinar que um ou mais aspectos do processo estejam além dos limites aceitáveis, e que o produto resultante será inaceitável, ele deve ajustar o processo ou o material a ser processado, e essa **adaptação** deve ser realizada com a maior rapidez disponível (SCHWABER; SUTHERLAND, 2010). O Scrum facilita a adaptação do projeto, pois não especifica ações quanto às práticas de desenvolvimento, antes conta com equipes auto-organizadas e com processos orientados a objetivos e resultados. A decisão de como utilizar e de quais estratégias criar para obter a produtividade e realizar a entrega de artefatos ficam por conta de quem está aplicando o processo (SCHAWBER, 2009).

Detalharemos mais adiante que há três pontos de inspeção e adaptação em Scrum: a *Sprint planning meeting* e a *Sprint review* são utilizadas para inspecionar o progresso da execução das metas e do *release* e para realizar adaptações que aperfeiçoam o valor da próxima sprint. A *Daily Scrum meeting* é utilizada para inspecionar processos que compõem a meta de uma sprint e fazer adaptações a fim de otimizar o valor do dia de trabalho seguinte. Por fim, a *Sprint retrospective* analisa a sprint passada e determina adaptações para a próxima *sprint* ter mais produtividade, ser mais gratificante e agradável (LEITÃO, 2010).

3.3.1 Papéis e responsabilidades

O Scrum possui três papéis: o *Scrum Master*, o *Product Owner* e o *team*. Os membros do Scrum são chamados de “porcos” (*pigs*); todo o resto é “galinha” (*chicken*). *Chickens* não podem dizer aos *pigs* como fazer seu trabalho, ou seja, não é permitida interferência externa no Scrum. Essas nomenclaturas vêm da história mostrada na figura 6, que versa sobre envolvimento e comprometimento no negócio (LEITÃO, 2010).

Figura 6: História sobre o comprometimento do Scrum Team.



Fonte: <http://www.eliezer.com.br/post/envolvimento-x-comprometimento-dilbert/>

Cada integrante do Scrum assume responsabilidades no desenvolvimento do produto, como apresentado a seguir:

Scrum Master

O *Scrum Master* é responsável para garantir a aplicação do Scrum. É também sua responsabilidade garantir que esta metodologia seja entendida por toda a equipe. Os *Scrum Masters* fazem isso garantindo que as equipes obedeçam à teoria, práticas e regras da metodologia. Ele é um líder-facilitador para toda a equipe (Schwaber; Sutherland, 2011). Seu papel está diretamente relacionado a obter técnicas para a gestão do projeto e acompanhamento da equipe.

Product Owner

É o responsável por definir os requisitos do produto e a prioridade de execução das atividades do projeto, de acordo com o valor para o negócio. Seu principal objetivo é maximizar o valor do produto e do trabalho da equipe de desenvolvimento. Sua tarefa é basicamente gerenciar o *Product Backlog*. Isso inclui, entre outras atividades, a definição dos itens do *Product Backlog*, garantindo que este seja visível, transparente e claro para todos e a garantia de que a equipe de desenvolvimento entenda todos os itens do *Product Backlog* ao nível que se faz necessário (Schwaber; Sutherland, 2011).

O Product Owner é necessariamente uma pessoa, e não um comitê. Ele pode representar os desejos e as necessidades de um comitê no *Product Backlog*, mas caso haja interesse por parte do comitê de realizar mudanças em algum item, estes devem antes convencer ao Product Owner (Schwaber; Sutherland, 2011).

Para que seu papel seja desempenhado corretamente, é necessário que toda a organização respeite suas decisões. Essas decisões estão sempre visíveis no conteúdo e na organização do *Product Backlog*. Ninguém tem autorização para delegar à equipe de desenvolvimento trabalhos em um conjunto diferente de requisitos e a equipe de desenvolvimento não está autorizada a agir sobre o que outras pessoas determinarem (Schwaber; Sutherland, 2011).

Team

Consiste em profissionais que executam o trabalho de entregar uma versão usável que potencialmente incrementa um produto “pronto” ao fim de cada *Sprint*. Suas principais características são: auto organizadas, isto é, não necessitam de alguém que diga como transformar o *Product Backlog* em incremento de funcionalidades utilizáveis. Outra característica marcante, é que o Scrum não reconhece perfis dos membros da equipe além do papel de desenvolvedor, seja qual for o trabalho desempenhado pelo integrante, não há exceções para esta regra. Por fim, o Team, ou equipe do Scrum, não possui sub-equipes que se dedicam a conhecimentos específicos, como testes ou análise de negócios, embora dentro de cada equipe possam existir membros que possuam habilidades em áreas especializadas, mas pertencem e correspondem à equipe de desenvolvimento como um todo (Schwaber; Sutherland, 2011).

Para Schwaber e Sutherland (2011) “a equipe de desenvolvimento no Scrum deve ser suficientemente pequena para que se mantenha ágil, e grande o suficiente para completar uma parcela representativa do trabalho”. Equipes pequenas poderão encontrar dificuldades para lidar com as habilidades necessárias para um *Sprint*, e como resultado não conseguir entregar algo funcional. Por outro lado, ao ter uma equipe excessivamente grande, será necessário uma maior coordenação e gerenciamento, o que pode tornar o processo muito complexo.

3.3.2 Artefatos

Os artefatos permitem a utilização das práticas ágeis do Scrum em um projeto. A união dos artefatos permite a criação de um ciclo de atividades que determinam a qualidade da metodologia. Tais artefatos são definidos a seguir, com algumas características descritas.

Product Backlog

Trata-se de um conjunto de itens escolhidos que deve ser desenvolvido em uma *sprint*, associado a um valor de negócio, que pode ser composta de requisitos funcionais ou não. (LEITÃO, 2010). A responsabilidade pela criação do *Product Backlog* é do *Product Owner*, incluindo o seu conteúdo, ordenação e disponibilidade.

É interessante destacar também, que em um projeto Scrum, um *Product Backlog* nunca está completo. Sua versão inicial apenas descreve o conhecimento adquirido no início das reuniões e os requisitos mais bem conhecidos. Sua evolução se dá ao mesmo tempo em que o produto e o ambiente em que está sendo utilizado também evoluem (Schwaber; Sutherland, 2011).

Sendo assim, o *Product Backlog* existe para reunir todas as funcionalidades, funções, requisitos, melhorias e ajustes que significam mudanças a serem feitas no produto durante as próximas versões. Cada item tem atributos de descrição, ordem e estimativa (Schwaber; Sutherland, 2011).

A fase do *Product Backlog* é aonde são coordenadas reuniões com os envolvidos com o projeto, com a finalidade de definir requisitos técnicos e funcionais, como padrões, estratégias de negócios e tecnologia a serem utilizados no projeto (SENA, 2010). A tabela 2 apresenta um exemplo de um *Product Backlog* de um sistema de reservas on-line.

Tabela 2: Exemplo de *Product Backlog*.

Nível de Prioridade	Categoria	Descrição do Item
A	Reserva	Os clientes poderão fazer reserva de apartamento
A	Reserva	Os clientes poderão cancelar a reserva
A	Reserva	Os clientes poderão fazer alterações de data da reserva
A	Reserva	Os clientes poderão fazer consulta de reservas
M	Reserva	Criação do <i>Book</i> de Reserva
A	Pagamento	O meio de pagamento da reserva será cartão de crédito
B	Apartamento	Os apartamentos deverão ser cadastros
B	Apartamento	Os apartamentos são classificados por categoria
A	Cliente	Precisamos registrar os dados dos clientes

Níveis de Prioridade: A-alto, M-médio e B-baixo.

Fonte: Scrum Experience, versão 16.

Sprint Backlog

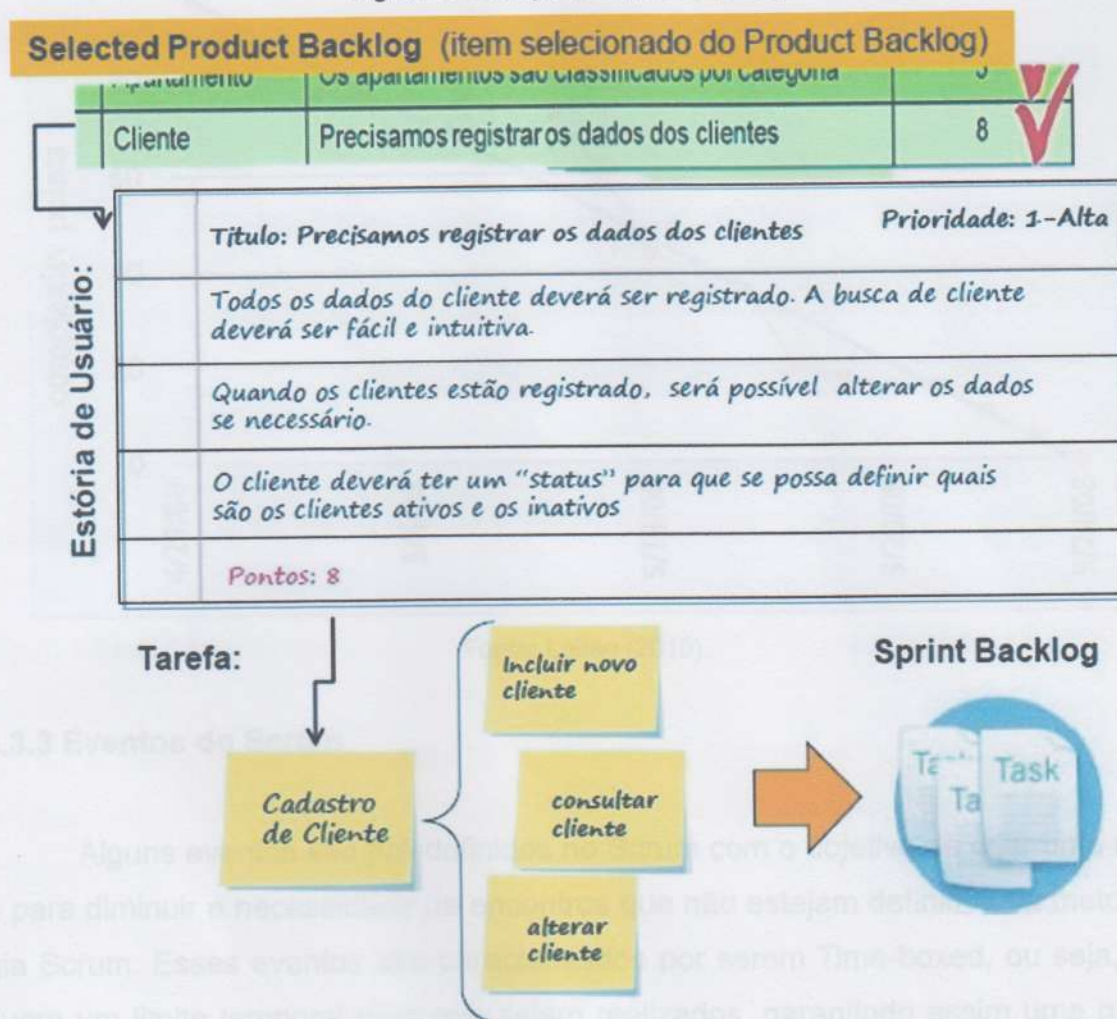
Assim como o *Product Backlog*, o *Sprint Backlog* é uma lista de atividades a serem desenvolvidas dentro de uma *Sprint*, de forma que ao final se tenha um produto a apresentar ao Cliente. Esse artefato é criado com base no *Product Backlog*. A partir deste, é extraído um conjunto de funcionalidades a serem desenvolvidas. (SENA, 2010).

Para Schwaber e Sutherland (2011) o *Sprint Backlog* é um plano de atividades com informações suficientes para que as mudanças em progresso sejam entendidas no Scrum diário. Caso um novo trabalho seja necessário, a equipe adiciona tal atividade ao *Sprint Backlog*. As estimativas do trabalho restante são atualizadas na medida em que o trabalho é desempenhado e concluído. Durante uma *Sprint*, alguns itens podem ser identificados como desnecessários e são removidos. As alterações no *Sprint Backlog* são efetuadas exclusivamente pela equipe de desenvolvimento.

Assim, Schwaber e Sutherland (2011) definem o *Sprint Backlog* como “uma figura em tempo real, altamente visível, do trabalho que a equipe de desenvolvimento planeja atingir durante o *Sprint*, e pertence somente à equipe de desenvolvimento”. A

figura 8 mostra um exemplo de um *Sprint Backlog* criado a partir de um item selecionado do *Product Backlog*.

Figura 7: Exemplo de Sprint Backlog.

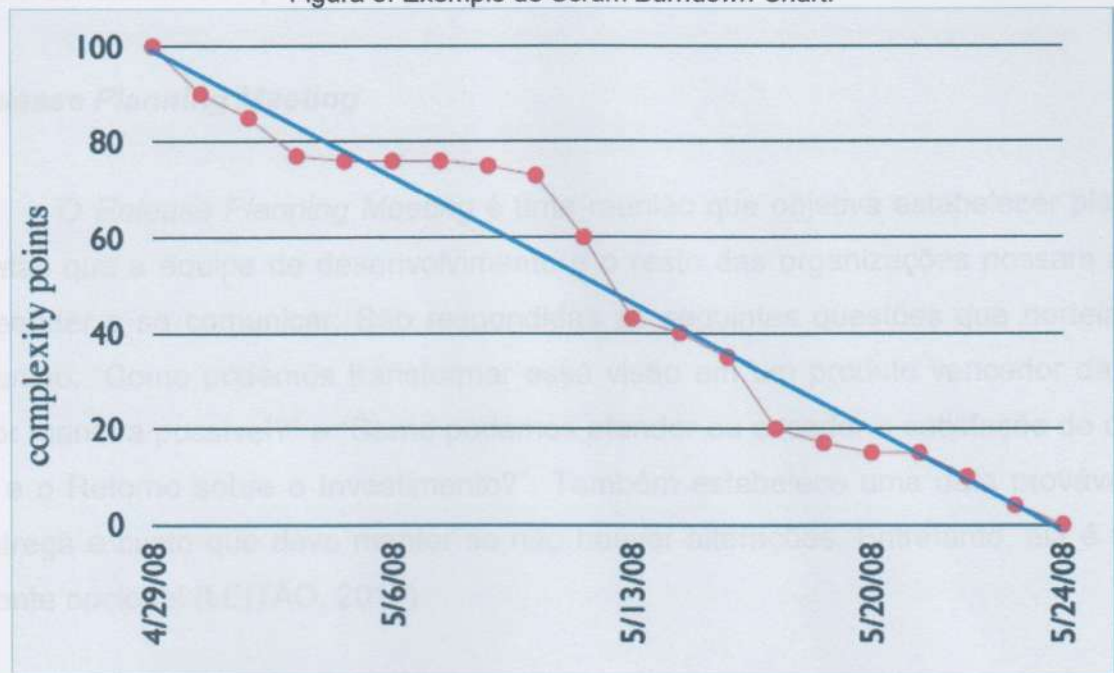


Fonte: Scrum Experience, versão 16

Gráfico *BurnDown*

O gráfico *Burndown* (*Burndown Chart*) serve para mapear o progresso da equipe de desenvolvimento e do trabalho restante ao longo do tempo, exibindo a correlação entre a quantidade de trabalho restante em qualquer ponto no tempo e o progresso das equipes na redução deste trabalho. É constituído por duas retas: uma de tendência, que representa uma iteração ideal onde o trabalho é concluído de forma estável e uniforme, e a reta real, que representa o trabalho efetivamente realizado pela equipe ao longo do tempo estipulado. A reta real decresce conforme as tarefas vão sendo finalizadas ao longo da *sprint* (SCHWABER apud LEITÃO, 2010), como mostrado na Figura 9.

Figura 8: Exemplo de Scrum Burndown Chart.



Fonte: Leitão (2010).

3.3.3 Eventos do Scrum

Alguns eventos são pré-definidos no Scrum com o objetivo de criar uma rotina e para diminuir a necessidade de encontros que não estejam definidos na metodologia Scrum. Esses eventos são caracterizados por serem Time-boxed, ou seja, possuem um limite temporal para que sejam realizados, garantindo assim uma quantidade adequada de tempo gasto com planejamento sem que haja perdas no processo de planejamento do projeto (Schwaber; Sutherland, 2011).

O Sprint

Sprint é um intervalo de duas a quatro semanas durante o qual uma versão potencialmente funcional de um produto é criada. Segundo Schwaber e Sutherland (2011, p. 7) "*Sprints* consistem em uma reunião de planejamento do *Sprint*, Scrum diários, o trabalho de desenvolvimento, a reunião de revisão do *Sprint* e a Retrospectiva do *Sprint*".

Durante uma *Sprint*, não é possível efetuar nenhuma mudança que afete ao objetivo principal da *Sprint*. O escopo pode ser detalhado e negociado várias vezes entre o *Product Owner* e o *Scrum Team* à medida que os requisitos do sistema se

tornar mais conhecidos. Por fim, a composição do *Scrum Team*, ou equipe de desenvolvimento, bem como a qualidade dos objetivos da *Sprint* se mantêm constantes até a conclusão da *Sprint* (Schwaber; Sutherland, 2011).

Release Planning Meeting

O *Release Planning Meeting* é uma reunião que objetiva estabelecer plano e metas que a equipe de desenvolvimento e o resto das organizações possam compreender e se comunicar. São respondidas as seguintes questões que norteiam a reunião: “Como podemos transformar essa visão em um produto vencedor da melhor maneira possível?” e “Como podemos atender ou exceder a satisfação do cliente e o Retorno sobre o Investimento?”. Também estabelece uma data provável de entrega e custo que deve manter se não houver alterações. Entretanto, ela é totalmente opcional (LEITÃO, 2010).

Sprint Planning Meeting

É nesta etapa que a iteração é planejada, selecionando as funcionalidades a serem implementadas durante a *sprint* com base nos itens do *Product Backlog* pré-definido e priorizado. Essa reunião consiste basicamente em duas partes, cada uma com duração média de 4 horas. Na primeira parte o *Product Owner* define as prioridades, seleção de itens do *Product Backlog* e as metas da *Sprint*.

Na segunda parte da *Sprint Planning Meeting* quando a equipe já possui definidas as funcionalidades a serem implementadas no *Sprint*. A equipe tem como objetivo definir o *Sprint Backlog*. Detectando quais as tarefas serão necessárias para cumprir a meta da *Sprint*.

Daily Meetings

Consistem em reuniões diárias que avaliam os progressos do projeto e as dificuldades encontradas durante as atividades de desenvolvimento. Nesta reunião também é realizada a discussão e determinada a prioridade da lista de *backlog* do produto. Sendo assim, Rising (apud SENA, 2010, p. 71) afirma que as *Daily Meetings* não contribuem diretamente para a conclusão das tarefas, mas satisfaz vários objetivos, principalmente a concentração dos esforços no backlog.

Sprint Review Meeting

A *Sprint Review Meeting* é um ponto de inspeção ao fim de cada iteração, em que é realizada uma reunião de revisão com duração máxima de 4 horas, acompanhada e regida pelo *Scrum Master*. É uma reunião tipicamente informal, onde a equipe exhibe ao *Product Owner* e aos *stakeholders* os itens de *Backlog* considerados prontos e inicia-se uma nova *sprint* (LEITÃO, 2010).

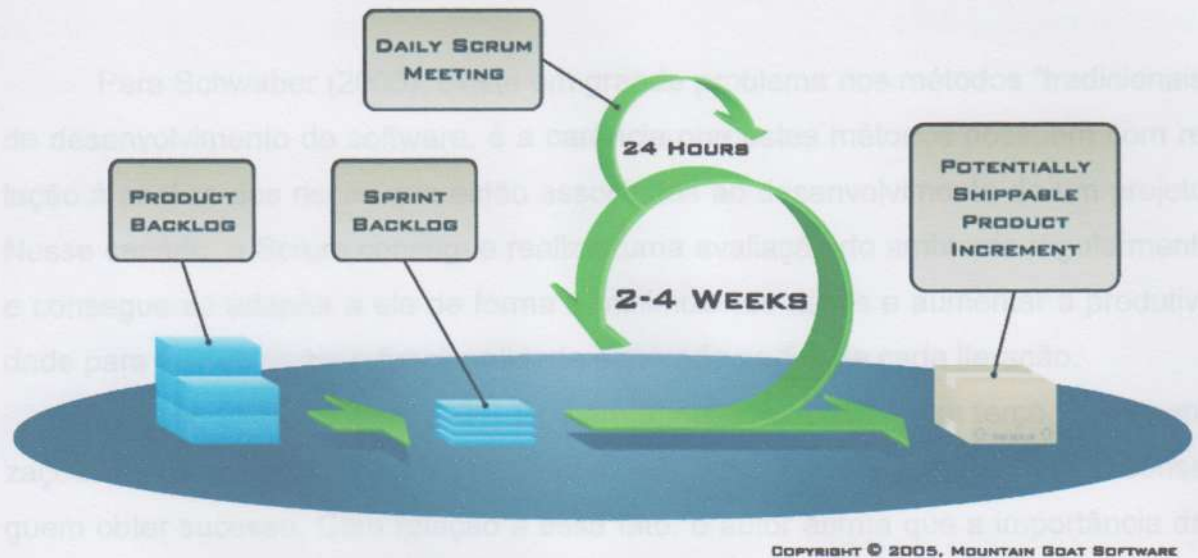
Sprint Retrospective

Realizada pelo *Scrum Team* é uma reunião com time-box de cerca de 3 horas, liderada pelo *Scrum Master* com o objetivo de rever, conforme a metodologia Scrum e suas práticas, o processo de desenvolvimento com a finalidade de torná-lo mais eficaz para a próxima *Sprint*. Isso é possível pelo fato do Scrum possuir conjunto de práticas que visam a melhoria adaptativa do processo, efetuando buscas por melhorias na produtividade ou qualidade do produto final (LEITÃO, 2010).

3.3.4 Ciclo de desenvolvimento

Em Scrum, projetos são desenvolvidos numa série de iterações com prazos de 2 a 4 semanas chamadas de *sprints*, que por sua vez tem como objetivo entregar um produto em funcionalidade e com valor agregado para o negócio. O ciclo de desenvolvimento do Scrum inicia com uma visão simples do produto que será desenvolvido, chamado de *Product Backlog*, que contém suas características definidas pelo *Product Owner*. A equipe de desenvolvimento se reúne com *Product Owner* antes do início de cada *sprint* para priorizar o trabalho a ser feito e selecionar as tarefas dentre as que existem no *Product Backlog* que a equipe pode realizar durante o tempo determinado da *sprint*; as tarefas selecionadas compõem o *Sprint Backlog*. Nessa fase também são definidas as ferramentas que serão utilizadas e detectados os possíveis impedimentos. A Figura 9 ilustra, de forma resumida, o ciclo de desenvolvimento do Scrum (LEITÃO, 2010).

Figura 9: Ciclo de desenvolvimento com Scrum



Fonte: www.mountaingoatsoftware.com/scrum (2011)

Para cada Sprint executado, a equipe realiza *Daily Scrum Meeting* com time-box de 15 minutos, são pequenas reuniões diárias onde é realizado o acompanhamento das atividades desempenhadas diariamente no projeto. A cada fim do Sprint é efetuada uma reunião de revisão, chamada de *Sprint Review Meeting*, para que o *Scrum Team* possa apresentar uma versão potencialmente entregável do produto para o *Product Owner*. Por fim, a *Sprint Retrospective Meeting* tem o objetivo de analisar o desempenho, o progresso do projeto e a versão do produto gerada, para que haja melhorias na equipe, no processo ou no produto para a *sprint* seguinte (LEITÃO, 2010).

3.3.5 Ferramenta de gerenciamento Redmine

Redmine trata-se de um software livre executado em ambiente web, capaz de gerenciar projetos e suas falhas (bugs). Além de trabalhar com múltiplos projetos, ele exibe uma série de ferramentas que auxiliam o acompanhamento das atividades distribuídas para os participantes de uma equipe. O *Redmine* foi baseado no *Trac*, outro software bem conhecido que se propõe a fazer o mesmo controle. Além de gerenciar as tarefas, também possui integração com diversos repositórios de dados (SVN, Subversion, Git, etc.), ferramenta para documentação e alguns itens de análise de desempenho da equipe.

3.3.6 Aplicações práticas

Para Schwaber (2006), existe um grande problema nos métodos “tradicionais” de desenvolvimento de software, é a carência que estes métodos possuem com relação à análise dos riscos que estão associados ao desenvolvimento de um projeto. Nesse cenário, o Scrum consegue realizar uma avaliação do ambiente regularmente e consegue se adaptar a ele de forma a minimizar os riscos e aumentar a produtividade para que se tenha a funcionalidade esperada no fim de cada iteração.

Entretanto Schwaber (2006) também afirma que apenas um terço das organizações que implementam o Scrum como um modelo de desenvolvimento conseguem obter sucesso. Com relação a esse fato, o autor afirma que a importância dada aos paradigmas convencionais de desenvolvimento é uma das principais causas para impedir a implantação dos métodos ágeis.

Segundo o autor, tanto as metodologias tradicionais quanto as metodologias ágeis não são fáceis de serem readaptadas. Tomando como exemplo o papel do gestor de projetos, que em uma metodologia tradicional gerencia a equipe e controla toda a produção da equipe de desenvolvimento, ao migrar para a metodologia Scrum, ele se depara com um alto nível de autogestão dessas equipes, e vê sua função modificada para *Scrum Master*, além de não exercer mais autoridade sobre a equipe de desenvolvimento. Certamente uma resistência ao modelo será gerada (SENA, 2010).

Entretanto, tais adaptações ao Scrum são plenamente possíveis, como exemplo temos algumas empresas como Philips, Cisco Systems, Siemens, Nokia, Microsoft e Sun Microsystems que implementaram este modelo (SENA, 2010).

Um dos principais motivos no aumento da produtividade dentro de um projeto é que sua organização exige que o Scrum funcional produza resultados de alto valor para o negócio desde a primeira *Sprint*, evitando que se façam componentes desnecessários e sem usabilidade para o cliente. Esse problema atinge muitas organizações que não implementam a metodologia Scrum (SCHWABER, 2006). Para Paula (2010, p. 11) “Um processo Scrum bem implementado poderá alcançar o efeito Toyota: quatro vezes mais produtividade e doze vezes mais qualidade.” Ela ainda afirma que:

Outra vantagem de se utilizar o Scrum é a produtividade aliada ao conforto. Ele permite aos colaboradores desenvolverem software na medida em que reduzem a pressão de administração, pois oferece liberdade para que cada membro da equipe selecione seus próprios trabalhos e organizem entre si (PAULA, 2010, p. 11).

Finalizando, Schwaber (2006) afirma que o Scrum possui uma reengenharia contínua que permite repentinas mudanças aos projetos. Tal característica é útil em alguns casos, porém há uma dificuldade para satisfazer casos em que não existe uma disponibilidade maior para a execução das iterações necessárias do Scrum, seja no quesito técnico ou pessoal. Da mesma maneira, o Scrum não é indicado para ambientes complexos de desenvolvimento. Isso exige um comprometimento pessoal excessivo por parte dos componentes da equipe, pois os gestores terão que delegar o processo de tomadas de decisões exclusivamente aos membros da equipe, o que lhes dá a possibilidade de erros, o que pode ser desastroso para o projeto. (SENA, 2010).

Abaixo, a tabela 3 apresenta uma análise dos itens do Scrum, comparando seus pontos fortes e fracos.

Tabela 3: Práticas e características da metodologia ágil Scrum.

Pontos Fortes	Pontos Fracos
Indicada quando não se tem todos os requisitos do sistema ou estes não estão totalmente definidos.	Não indicada para sistemas complexos, pois a análise não é detalhada.
Flexibilidade para mudanças constante durante as fases do projeto.	Não indicada para projetos que exigem vasta documentação.
O cliente faz parte da equipe em tempo integral.	Não indicada quando os clientes não têm disponibilidade de tempo ou são resistentes a troca de informações.
Reuniões diárias que guiam o andamento do projeto.	Não indicada para equipes resistentes a reuniões frequentes.
Ciclos de desenvolvimento curtos e constantes.	Não indicada para projetos que necessitam de ciclos de iterações longos.
Comunicação entre os membros da equipe é freqüente.	Não indicada em ambientes onde as tarefas são impostas às pessoas.
Revisões das funcionalidades realizadas acontecem no final de cada ciclo.	Não indicada em projetos que o líder não divide responsabilidades com a equipe.

Fonte: Leitão (2010).

4. ELABORAÇÃO DE PROPOSTA DE IMPLANTAÇÃO DO MÉTODO SCRUM

Aqui descreveremos o ambiente proposto como objeto de análise, identificando suas necessidades quanto à adesão de uma metodologia ágil para o gerenciamento de seus projetos. Em seguida apresentamos, com base nas necessidades identificadas na análise inicial, uma proposta de adequação da metodologia Scrum.

4.1 Caracterizações do ambiente

O estudo de caso deste trabalho trata-se do núcleo de desenvolvimento de softwares da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte – FARN. Esse núcleo é constituído por professores e alunos, que se dispõe a formar uma equipe para realizar projetos de desenvolvimentos de softwares, tanto para a própria instituição, como também para outras organizações.

Atualmente, o núcleo desenvolve um projeto de desenvolvimento de um sistema para as clínicas integradas da FARN. As Clínicas Integradas da FARN oferecem atendimento humanizado à população em geral nas áreas de Fisioterapia, Nutrição, Enfermagem e Psicologia.

A proposta é desenvolver um sistema que integre as informações dos pacientes, de forma que os prontuários sejam armazenados em banco de dados, e o gerenciamento de toda a clínica seja feita via sistema.

4.1.2 Equipe de desenvolvimento

4.1.1 Processo de desenvolvimento

A parte de análise de requisitos desenvolvida pelo grupo utiliza procedimentos bem semelhantes aos processos da metodologia “tradicional”. Utilizando-se de artefatos da UML, criam casos de uso, diagramas de classes e documentos onde são realizadas as especificações dos requisitos funcionais e não-funcionais, bem como definições das regras de negócio.

Toda essa informação é colhida junto ao cliente, todavia, não existe um acompanhamento por parte deste junto ao grupo de desenvolvimento. Não há um representante do cliente que entenda todo o processo necessário para o desenvolvimento do software, nem que saiba definir quais as funcionalidades são prioritárias para que se tenha de forma mais rápida um software potencialmente funcional.

As reuniões do núcleo são feitas quinzenalmente. Nessas reuniões são definidas atividades a serem desenvolvidas, e seus responsáveis. Há também discussões sobre os requisitos do sistema. Em alguns casos as informações acerca do negócio não são suficientes, gerando dúvidas quanto ao desenvolvimento de algumas funcionalidades. Nessas reuniões, professores também auxiliam os alunos integrantes da equipe a executar tarefas relacionadas ao desenvolvimento do software, pois como se trata de um grupo que também visa o aprendizado, se faz necessário até que todos os integrantes da equipe estejam com um nível de conhecimento suficiente para realizar as suas atividades de forma independente. O tempo dessas reuniões não é definido, bem como a participação da equipe nem sempre é com todos os seus integrantes.

4.2.1 Ao fim de cada reunião, novas tarefas estarão definidas e atribuídas à equipe e com prazos definidos. As tarefas antigas são reorganizadas e distribuídas entre os integrantes presentes na reunião e seus prazos são prorrogados. Todo o gerenciamento dessas atividades, acompanhamento e detalhamento das tarefas são realizados através do *Redmine*, um software com licença *Open Source*, capaz de gerenciar projetos de desenvolvimento de software.

Prod Não existe nenhuma rotina de testes dentro do núcleo, para garantir a qualidade do software codificado. Todavia os membros da equipe podem desenvolver seus próprios testes, identificando erros e garantindo um software limpo e sem erros.

O Product Owner é responsável por criar, juntamente com a equipe, um conjunto de requisitos para o negócio do cliente. O Product Backlog são as tarefas a serem realizadas, e é incrementado com novos requisitos sempre que se

4.1.2 Equipe de desenvolvimento

No núcleo de desenvolvimento de softwares da FARN participam professores e alunos da instituição. Os professores atuam como orientadores e gerenciam as etapas de análise e desenvolvimento dos softwares, enquanto os alunos integram a equipe de desenvolvimento, com funções de analista de requisitos, DBAs e desenvolvedores, sendo que uma pessoa pode assumir várias funções dentro da equipe.

Entretanto a equipe de desenvolvimento enfrenta diversos problemas na sua formação, distribuição e execução de tarefas. Manter a equipe de forma contínua, comparecendo a reuniões e executando tarefas dentro do prazo tem se tornado uma tarefa cada vez mais complexa dentro do núcleo. A rotatividade de pessoas que passam pelo núcleo é relativamente alta, com isso, há um aumento no número de tarefas não executadas e atraso na entrega do produto, prejudicando o cliente gerando insatisfação para todos os envolvidos no processo.

4.2.2 Percebe-se ainda que, a equipe não é auto-organizada, antes, necessita da intervenção dos professores para que esses gerenciem as atividades das equipes, delegando atividades e distribuindo responsabilidades entre os componentes do núcleo.

4.2 Proposta de implantação do Scrum

Após efetuar a análise da situação atual do núcleo, foi elaborada uma proposta de utilização do Scrum, através da criação de artefatos, definição de papéis e realização de eventos definidos pela metodologia ágil.

4.2.1 Artefatos

Os artefatos definidos no Scrum ajudam a elaborar o *Sprint*, espaço de tempo necessário para que uma determinada funcionalidade seja desenvolvida. Para isso, os seguintes artefatos serão utilizados:

Product backlog

Com o objetivo de realizar um levantamento da necessidade inicial do cliente, o *Product Owner* é responsável por criar, juntamente com a equipe, um conjunto de funcionalidades capaz de atender ao negócio do cliente. O *Product Backlog* só acaba quando o projeto encerra, e é incrementado com novos requisitos sempre que se tiver um aprofundamento no conhecimento das necessidades do cliente.

Sprint backlog

Após se obter uma primeira versão do Product Backlog, é criada uma série de funcionalidades a serem desenvolvidas durante um sprint. O time-box para um *sprint* deve ser de 2-4 semanas, dependendo da quantidade de itens que a compõe. Ao executar todos os itens de um Sprint Backlog a equipe tem que apresentar o produto de forma funcional, pronto para uso, para que seja avaliado e validado pelo cliente.

4.2.2 Redefinição de papéis e responsabilidades

Product Owner

Definir dentro da equipe, uma pessoa capaz de representar o cliente, tendo o conhecimento de toda a sua necessidade e que se responsabilize pela formulação do *Product Backlog*. O *Product Backlog* não necessariamente é desenvolvido totalmente pelo Product Owner, antes deve haver a participação da equipe também em sua criação. Porém, somente ao *Product Owner* é atribuída a responsabilidade pelo gerenciamento deste artefato.

Scrum Master

Definir também dentro da equipe o *Scrum Master*, capaz de garantir que o processo do Scrum seja seguido por todos os membros da equipe. Ele também se responsabiliza por liderar as reuniões e acompanhar as atividades realizadas pela equipe. O papel do Scrum Master deve ser adaptado para que este possa gerenciar a equipe, definindo também funções e papéis. Isto se faz necessário porque embora o Scrum sugira uma equipe auto-organizada, a falta de experiência ou a necessidade de tomar decisões durante o projeto exigem o papel de um líder gerenciador para o projeto.

Sprint Retrospective

Equipe de desenvolvimento

A equipe de desenvolvimento do núcleo deve trabalhar em constante comunicação, participando de todos os eventos, expondo suas dificuldades encontradas durante o *Sprint*. Ela será gerenciada pelo *Scrum Master*. Com os itens do sprint backlog disponível, cada integrante será capaz de assumir a responsabilidade pelo desenvolvimento dos itens, e garantir o cumprimento do prazo para a execução das tarefas. É importante que a equipe que inicie um *Sprint* seja a mesma até o seu término.

4.2.3 Definição de Eventos

Os Eventos são reuniões com tempo definidos, que devem resultar em artefatos para serem utilizados diariamente durante a execução do projeto.

Sprint Planning Meeting

O *Sprint Planning Meeting* deverá ocorrer sempre que for necessária a criação de um novo *Sprint Backlog*. É uma reunião que envolve todos os integrantes do projeto.

Sprint Review Meeting

Este evento deve ocorrer sempre que for necessário realizar revisões na *Sprint*. Como a reunião diária não será realizada, por se tratar de um grupo de estudantes e professores, que não fazem as atividades de forma secundária, a *Sprint Review Meeting* servirá como as reuniões necessárias para realizar o acompanhamento das tarefas realizadas durante um *Sprint*. Nesse evento, poderão ser incluídos ou retirados itens do *Sprint Backlog*, desde que tais itens não comprometam o prazo de entrega do produto da *Sprint*.

Sprint Retrospective

É um evento que deverá ser realizado sempre que se concluir um *Sprint Backlog*. Nessa reunião, devem-se avaliar todas as formas utilizadas durante o *Sprint* a fim de que se possa ter sempre melhoria nos processos. Também é avaliado o produto final da *Sprint*, onde a equipe apresenta o produto totalmente funcional.

Ferramenta de gerenciamento Redmine

O software *redmine* deve continuar sendo utilizado como ferramenta de gerenciamento de projetos Scrum, para isso é necessário adequações, como por exemplo, definindo os papéis de Scrum Master, Product Owner e equipe e criando *Backlogs* através das configurações que são disponibilizadas pelo software.

4.4 Resultados esperados

Com a proposta apresentada em execução, espera-se obter os seguintes resultados:

- Partes do software prontas a cada Sprint executado.
- Comprometimento da equipe com as tarefas do Sprint, a fim de obter um resultado real e funcional.
- Levantamento de requisitos de forma Incremental, para cada parte do software criada.
- Total comprometimento do cliente em relação ao projeto, podendo inclusive atribuir-lhe a responsabilidade de priorização das partes do software a serem desenvolvidas.

5. CONSIDERAÇÕES FINAIS

Observando os resultados da pesquisa realizada, podemos verificar que as metodologias ágeis surgiram como uma solução para os problemas encontrados nos mais variados projetos de desenvolvimento de software, e que o método tradicional possui algumas falhas na sua própria composição, mesmo assim, ainda é útil em varios projetos de desenvolvimento de software nas mais variadas organizações pelo mundo.

O manifesto ágil descreve claramente que a metodologia ágil enfoca áreas que realmente se fazem necessárias em um projeto de software. Pessoas ao invés de processos, software funcional ao invés de documentação, colaboração do cliente ao invés de negociação de contrato e mudanças ao invés de um escopo bem definido são os itens do manifesto que se propõem a resolver grande parte dos problemas que surgem na utilização das metodologias tradicionais.

Dentre as metodologias apresentadas, o Scrum se destaca por ser uma metodologia simples, que se adéqua a pequenas equipes e a projetos que estejam abertos a constantes mudanças. Suas práticas não determinam o que fazer, elas dão uma série de possibilidades para que a metodologia se adéque às mais diversas empresas garantindo a mesma qualidade.

Conclui-se, portanto, através do estudo de caso efetuado neste trabalho, que quando não há metodologias, o projeto tende a perder o foco, não garantindo qualidade ao software desenvolvido, além de perder o controle sobre as estimativas de prazo para a conclusão do projeto, que por consequência geram excesso de gastos e acabam aumentando o valor do software.

Identificamos também que no núcleo de desenvolvimento de software da FARN alguns problemas relacionados à formação da equipe, rotatividade dos membros, alto índice no número de tarefas em aberto, bem como um alto índice de prorrogação de prazos da conclusão das tarefas. Constatou-se ainda que há uma falta de participação do cliente na priorização de itens a serem desenvolvidos.

Para estes e outros problemas apresentados, o Scrum possui recursos suficientes para sanar todas as deficiências atuais do núcleo, além de trazer uma nova forma de desenvolvimento para a equipe, fazendo com que as pessoas sejam mais valorizadas se tornando trabalhadores do conhecimento.

REFERÊNCIAS

- AMBLER, Scott W. **MODELAGEM ÁGIL**, ARTMED EDITORA, PORTO ALEGRE, 2002.
- ARAUJO, Leonardo. **Estudo comparativo da compatibilidade entre as melhores práticas do pmi e scrum**. 2009 Dissertação (Pós-graduação) – Gestão de projetos. Fiap, São Paulo, 2009.
- BARIFOUSE, Rafael. **Mercado de software corporativo atingirá US\$ 232 bi em 2010**. Disponível em <<http://colunas.epocanegocios.globo.com/tecneira/2010/09/20/mercado-de-software-corporativo-atingira-us-232-bi-em-2010-diz-gartner/>>. Acesso em 06 ago 2011.
- BECK, Ken. **Programação Extrema (XP) explicada: acolha as mudanças**. Bookman, Porto Alegre, 2004.
- CARVALHO, Ariadne; M. B. Rizzoni e CHIOSSI, Thelma C. dos Santos. **Introdução à engenharia de software**. ed.Unicamp, São Paulo, 2001.
- CISNEIROS, Hugo. **Modelo de Desenvolvimento Ágil**. Disponível em: <<http://www.devin.com.br/modelo-scrum/>> Acesso em: 23 nov 2011.
- FEATURE DRIVEN DEVELOPMENT. Disponível em: <<http://www.featuredrivendevelopment.com/>>. Acesso em: 30 nov 2011.
- HEPTAGON FDD – FEATURE DRIVEN DEVELOPMENT Disponível em <<http://adailmr.sites.uol.com.br>>. Acesso em 30 nov 2011.
- LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. 2010, 72 f. Trabalho de conclusão de curso (graduação) – Engenharia de Computação. Poli, Recife, 2010.
- LEITE, Alessandro. **Metodologia de desenvolvimento de software**. Disponível em <<http://www.devmedia.com.br/post-1903-Metodologia-de-desenvolvimento-de-Software.html>> Acesso em 01 ago 2011.

MAZZOLA, Vitorio Bruno. **Engenharia de software e sistemas de informação**. disponível em <<http://www.multibrasil.net/2010/11/engenharia-de-software-vitorio-bruno.html>>. Acesso em 10 nov 2011.

PAULA, Renata de Souza Alves. **Scrum na Melhoria do Gerenciamento de Projetos de Software**. Revista Engenharia de software magazine ed. 23 p. 8-17, DevMedia, Rio de Janeiro, 2010.

PRESSMAN, Roger S. **Engenharia de software**. Pearson Makron Books, São Paulo, 2007.

PRIKLADNICKI Rafael; AUDY, Jorge Luis Nicolas. **Desenvolvimento distribuído de software**. Elsevier, São Paulo, 2007.

SAVOINE, M.; et al. **Análise de Gerenciamento de Projeto de Software Utilizando Metodologia Ágil XP e Scrum: Um Estudo de Caso Prático**. In: XI Encontro de Estudantes de Informática do Tocantins, 2009, Palmas. Anais do XI Encontro de Estudantes de Informática do Tocantins. Palmas: Centro Universitário Luterano de Palmas, 2009. p. 93-102. Disponível em: <http://www.portaict.com.br/blogs/tecnologia/administracao/files/files/Analise_de_Gerenciamento_de_Projeto_de_Software_Utilizando_Metodologia_Agil_XP_e%20_Scrum_Um_Estudo_de_Caso_Pratico.pdf> Acesso em 06 ago 2011.

SCHWABER, Ken. **Agile Project Management with Scrum**. Microsoft Press, 2006.

SCHWABER, Ken. **Guia do Scrum**. Scrum Alliance. 2009.

SCHWABER, Ken; SUTHERLAND, Jeff. **Scrum Guide: Developed and sustained**. Scrum.org. 2011.

SENA, Emerson Souza de. **Estudo comparativo entre as metodologias ágeis Extreme programming e Scrum**. 2010. Dissertação (Pós-graduação) – Especialização em desenvolvimento de sistemas corporativos, Farn, Natal, 2010.

SOUZA, Luciano Malaquias de. **Método Ágil XP (Extreme Programming)** Revista eletrônica da FIA Vol.III ano 2007. Disponível em <http://intranet.fia.edu.br/acesso_site/fia/academos/revista3/6.pdf>. Acesso em 25 nov 2011.

ZANATTA, Alexandre Lazaretti. **Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI**, 2004 – Disponível em <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf>. Acesso em 28 nov 2011.