UNIVERSIDADE FEDERAL DE PELOTAS
Instituto de Física e Matemática
Departamento de Informática

Trabalho Acadêmico

SUGESTÃO E MODELAGEM DE PRÁTICAS DO DESENVOLVIMENTO ÁGIL PARA APLICAÇÃO EM DESENVOLVIMENTO DISTRIBUÍDO *ONSHORE INSOURCING*

Roberto Bonow

Pelotas, 2008

ROBERTO BONOW

SUGESTÃO E MODELAGEM DE PRÁTICAS DO DESENVOLVIMENTO ÁGIL PARA APLICAÇÃO EM DESENVOLVIMENTO DISTRIBUÍDO *ONSHORE INSOURCING*

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação do Instituto de Física e Matemática da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profª.Msc. Eliane Alconforado Diniz.
Co-Orientador: Prof. Juliano Lucas Gonçalves

PELOTAS
2008

Dados de catalogação na fonte:
Maria Beatriz Vaghetti Vieira – CRB-10/1032
Biblioteca de Ciência & Tecnologia - UFPel

B719s Bonow, Roberto

Sugestão e modelagem de práticas do desenvolvimento ágil para aplicação em desenvolvimento distribuído onshore insourcing / Roberto Bonow; orientador Eliane Alcoforado Diniz. – Pelotas, 2008. – 91f. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2008.

1.Informática. 2.Engenharia de software. 3.Desenvolvimento distribuído de software. 4.Desenvolvimento onshore insourcing. 5. Metodologias ágeis.6. Processo de desenvolvimento. I.Diniz, Eliane Alcoforado. II.Título.

CDD: 005.3

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus que tem me acompanhado e me sustentado durante toda a minha vida, ao longo do curso e projeto de conclusão. Obrigado Paizão!

Agradeço a minha família, especialmente ao meu pai Egon, e minha mãe Liane, que sempre me apoiaram nos estudos e nas decisões importantes da minha vida.

A minha namorada Janaina que, apesar da distância na maior parte deste trabalho, sempre esteve perto me ajudando e dando palavras de incentivo. Te amo pra sempre gatinha!

Agradeço aos meus orientadores Eliane e Juliano por todo apoio e orientação no desenvolvimento deste trabalho.

Agradeço a todos que direta ou indiretamente ajudaram a vencer esta importante etapa da minha vida.

“Assim, quer vocês comam, bebam ou façam qualquer coisa, façam tudo para a glória de Deus” 1 Coríntios 10:31

RESUMO

BONOW, Roberto. **Sugestão e modelagem de práticas do desenvolvimento ágil para aplicação em desenvolvimento distribuído *onshore insourcing*. .** 2008. 91. Trabalho acadêmico (Graduação) - Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

O processo de desenvolvimento de novos produtos de software tem gerado grandes e novos desafios, ampliando a complexidade de construção de sistemas para empresas, onde velhos problemas podem ser acentuados e novos problemas de desenvolvimento podem ser agregados. Com a globalização, surgiram novas oportunidades, onde muitas organizações têm demonstrado interesse pelo desenvolvimento distribuído de software. Nesse contexto, os projetistas vêm procurando encontrar um processo ou metodologia que melhore a produtividade e qualidade do software. Com o surgimento de novas metodologias conhecidas como ágeis, que procuram agregar a facilidade de desenvolvimento, a produtividade e a qualidade do produto final, se faz necessário que os desenvolvedores tenham conhecimento do potencial e domínio dessas novas metodologias em relação ao desenvolvimento com equipes distribuídas. Neste trabalho será feita uma sugestão de práticas para serem aplicadas por empresas onde suas equipes se encontram geograficamente dispersas, em um mesmo país. Será feita uma análise das práticas encontradas nas principais metodologias ágeis, além de uma modelagem das principais fases e papéis envolvidos durante o processo de desenvolvimento de software.

Palavras-chave:. Desenvolvimento distribuído de software. Desenvolvimento *onshore insourcing.* Metodologias ágeis. Processo de desenvolvimento

ABSTRACT

BONOW, Roberto. **Sugestão e modelagem de práticas do desenvolvimento ágil para aplicação em desenvolvimento distribuído *onshore insourcing*. .** 2008. 91. Trabalho acadêmico (Graduação) - Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

The process of developing new software products hás created great and new challenges, increasing the complexity of building systems for new companies, where old problems can be accentuated and new problems of development can be aggregated. With globalization, new opportunities have emerged, for many companies have shown interest in the development of distributed software. In this context, designers are trying to find a process or methodology that improves productivity and quality of the software. With new methods emerged, known as agile, which intend to aggregate facility of development to productivity and to quality of final products, it is necessary that designers know the potential and the field of these new methodologies regarding the development with distributed teams. In this paper a suggestion about practices to be applied by companies which teams are geographically dispersed, in the same country, will be made. Will be done an analysis of the practices used in the main agile methodologies and a model of the main stages and roles involved in the process of developing software.

Keywords: Distributed development of software. Onshore insourcing development. Agile methodologies. Development process

.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PDS	- Processo de Desenvolvimento de Software
XP	- *eXtreme Programming*
UP	- *Unified Process*
DSDM	- *Dynamic System Development Methodology*
RAD	- *Rapid Application Development*
FDD	- *Feature Drivem Development*
DDS	- Desenvolvimento Distribuído de Software
UML	- *Unified Modeling Language*
TI	Tecnologia da Informação
VoIP	Voz sobre *IP*

SUMÁRIO

1 INTRODUÇÂO

O desenvolvimento de software, ao longo dos anos, vem apresentando grandes avanços e transformações em termos teóricos, relacionados às técnicas, modelos e metodologias de engenharia de software (FARIAS, 2006). O processo de desenvolvimento de software é considerado o principal mecanismo para se obter software de qualidade e cumprir corretamente os contratos firmados entre os desenvolvedores e os seus clientes. Logo, é importante que sejam utilizados os conhecimentos adquiridos ao longo dos anos pela área da engenharia de software para se obter esse tipo de produto.

Atualmente, com a globalização e a busca por novas formas de desenvolvimento, o processo de desenvolvimento de software tem gerado grandes e novos desafios, ampliando a complexidade de construção de sistemas para empresas, onde velhos problemas podem ser acentuados e novos problemas de desenvolvimento podem ser agregados (PRIKLADNICKI; AUDY, 2008). O desenvolvimento distribuído de software está ganhando espaço nos últimos anos, onde existe uma busca por novos mercados, mão de obra especializada e redução de custos de desenvolvimento. Como exemplo do crescente uso do desenvolvimento distribuído, pode-se citar o modelo *onshore insourcing,* onde uma determinada empresa possui equipes de desenvolvimento situadas em locais dispersos, em um mesmo país, trabalhando em um mesmo projeto.

O desenvolvimento distribuído possui algumas dificuldades se comparado a ambientes de desenvolvimento com equipes co-localizadas. Dentre essas dificuldades pode-se citar a escolha do processo para desenvolvimento.

Há muito tempo, vem se tentando encontrar um processo ou metodologia que melhore a produtividade e qualidade do software, esta busca também ocorre no contexto distribuído de desenvolvimento. A cada dia a computação vem atuando

mais próxima às empresas e aos negócios, tornando-se uma área estratégica e fazendo com que as empresas se tornem cada vez mais dependentes da área da computação. Com isso o processo de desenvolvimento de software requer cada vez mais uma maior robustez e qualidade, razão pela qual surgiram novos métodos de desenvolvimento de software (COAD, 1996, BOOCH; RUMBAUGH; JACOBSON , 2005).

A partir da década de 90, novas abordagens surgiram para guiar a construção de processos de desenvolvimento de software, entre elas pode-se citar: *Extreme Programming* – XP; UP; Scrum; etc. Essas abordagens são conhecidas como metodologias ágeis para desenvolvimento de software e são uma alternativa em relação aos métodos pesados ou tradicionais, focados em etapas fixas como análise de requisitos, projeto, codificação, manutenção e testes.

1.1.Motivação

Hoje em dia é muito comum se encontrar software de baixa qualidade, projetos que excedem as previsões de custo e tempo de desenvolvimento, ou sistemas que não atendem a todas as exigências esperadas. A cada ano que passa a importância do software e, consequentemente, sua complexidade cresce.

O desenvolvimento de software com equipes dispersas tem ganhado força, e com isso os processos de desenvolvimento de software vem recebendo maior importância como forma de prover software de qualidade e cumprir corretamente os prazos de desenvolvimento. As metodologias ágeis são uma opção para o desenvolvimento de software, porém, não existe um modelo de processo de desenvolvimento ágil ideal para o desenvolvimento distribuído. Neste sentido, o presente trabalho busca fazer um estudo dentre as principais metodologias ágeis, onde serão avaliados os diferentes modelos existentes, realizando uma análise de quais práticas podem ser aplicadas no contexto distribuído de desenvolvimento.

As dificuldades encontradas pelos desenvolvedores quando da escolha do método de desenvolvimento que melhor atentam as suas necessidades nesse contexto foram agentes motivadores para a realização desse trabalho.

1.2 Objetivos

O objetivo deste trabalho é realizar uma análise das principais metodologias ágeis, verificando quais características podem ser aplicadas no contexto distribuído de desenvolvimento. O objetivo específico deste trabalho é aprofundar o conhecimento nessas duas áreas importantes da engenharia de software: desenvolvimento distribuído de software e metodologias ágeis de desenvolvimento.

A partir desta análise será feita uma sugestão de práticas que podem ser aplicadas no desenvolvimento distribuído *onshore insourcing*, que é caracterizado pelo desenvolvimento onde duas ou mais partições de uma empresa desenvolvem software em locais dispersos, em um mesmo país.

1.3 Organização do Trabalho

Este trabalho é composto por seis capítulos, os quais são apresentados a seguir:

Capítulo 2 - contém informações relativas a processos de desenvolvimento de software, com os principais conceitos relacionados a processos conhecidos como tradicionais, e ágeis; também é realizada uma comparação entre essas duas abordagens de desenvolvimento. Capítulo 3 - demonstra as características das principais metodologias ágeis. Capítulo 4 - trata sobre desenvolvimento distribuído de software, demonstrando os principais conceitos relacionados, com as características dos diferentes níveis de dispersão, além das vantagens e desvantagens deste modelo de desenvolvimento. Capítulo 5 - apresenta uma sugestão para aplicação no contexto distribuído *onshore insoursing,* baseada na análise das metodologias ágeis e das características do Desenvolvimento distribuído de Software. Capitulo 6 - tem-se as conclusões do presente trabalho, demonstrando as vantagens relacionadas ao uso deste modelo, dificuldades encontradas durante a realização deste projeto de conclusão, e possíveis trabalhos futuros que poderão dar continuidade ao presente estudo.

2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Um processo de desenvolvimento de software - PDS pode ser definido como um conjunto de atividades, as quais são necessárias para transformar requisitos em um produto de software, podendo gerar novos produtos conforme alterações futuras nos requisitos (JACOBSON, 1999). O PDS é um conjunto de passos parcialmente ordenados com a intenção de chegar-se a um objetivo. Na engenharia de software, o objetivo é entregar um produto de software capaz de atender os seus requisitos, de maneira eficiente e previsível e com um custo acessível (BOOCH; RUMBAUGH; JACOBSON, 2005). O alto custo de produção, geralmente, se dá pelo alto risco e complexidade de desenvolvimento do produto de software (HERZUM, 2000). Outros fatores, como equipe, ferramentas, gerenciamento de produção e vendas, e processos utilizados contribuem para o alto custo de desenvolvimento.

Segundo Waslawick (2004), um processo deve levar a produção de software de qualidade, de forma organizada, e com possibilidade de incluir ou modificar requisitos existentes, mesmo que o sistema já esteja implementado. Os processos surgiram visando a redução da complexidade no desenvolvimento. De acordo com Bezerra (2007 apud Tanaka e Banki, 2008), os principais objetivos de um processo são: definir as atividades a serem realizadas no decorrer do projeto, definir a ordem com que as atividades serão realizadas (e também quem irá executá-las), obter um maior controle do andamento, e padronizar a forma com que o software será desenvolvido. Cada uma destas atividades inerentes ao desenvolvimento de software despende certo tempo, e a soma do tempo de desenvolvimento de cada etapa resulta do tempo total do projeto. A Fig. 1 demonstra o tempo médio gasto em cada etapa, durante o ciclo de vida de um software.

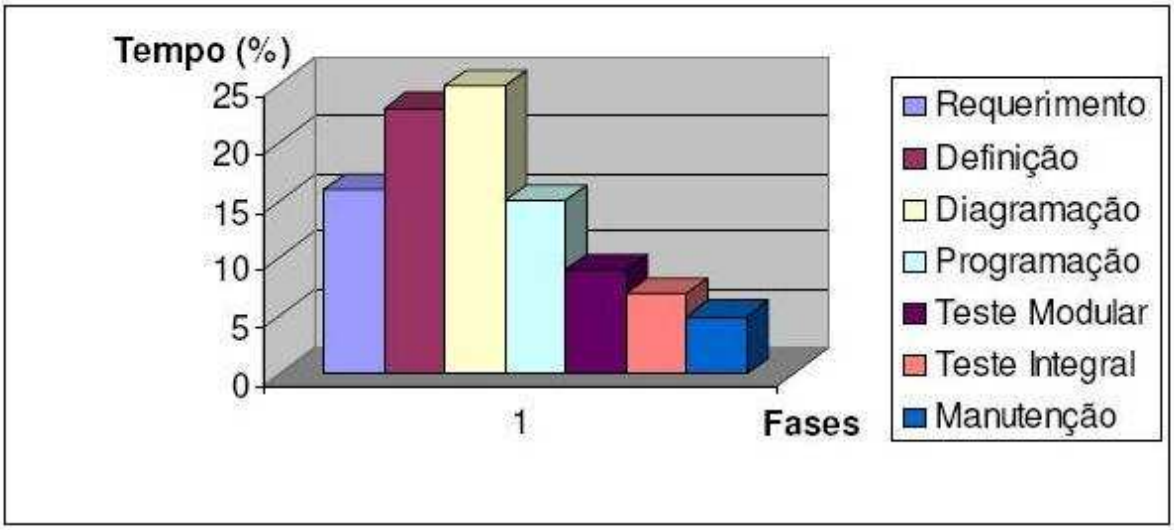

Figura 1: Ciclo de vida dos sistemas utilizando engenharia de software
Fonte: Higa, Neto e Furlan (1996).

Atualmente, com a evolução da engenharia de software, foram criados vários modelos de processos (ciclo de vida). Porém é difícil dizer qual modelo é melhor ou pior. Autores e empresas classificam seus processos de formas diferentes, dificultando uma uniformidade de avaliação (BONA, 2002), trazendo dificuldades durante a escolha de qual modelo utilizar para o desenvolvimento do produto. Assim como a engenharia de software evoluiu, a demanda por software também evoluiu. A complexidade dos sistemas de software está aumentando, o ambiente de negócios está crescendo, e o processo de globalização também tem gerado novos desafios no PDS (PRIKLADNICKI; AUDY, 2008). Cabe ainda ressaltar que, um produto de software não depende apenas de um bom processo de software, outros fatores como, a escolha de bons profissionais para operacionalizar é de vital importância para um projeto ser bem sucedido, porém, a escolha de um processo ruim para determinado projeto pode tornar a melhor das equipes ineficiente (MARTIN, 2002).

De acordo com Standish Group International (2001), até o ano 2000, poucos projetos atingiam êxito ao seu final. A pesquisa ilustrada na Fig. 2, demonstra que apenas 28% dos projetos conseguiam chegar aos seus objetivos com sucesso, os demais projetos ou eram cancelados ou extrapolavam suas previsões de custo ou tempo de desenvolvimento. A pesquisa demonstra a percentagem de projetos que falharam, foram cancelados, e obtiveram sucesso.

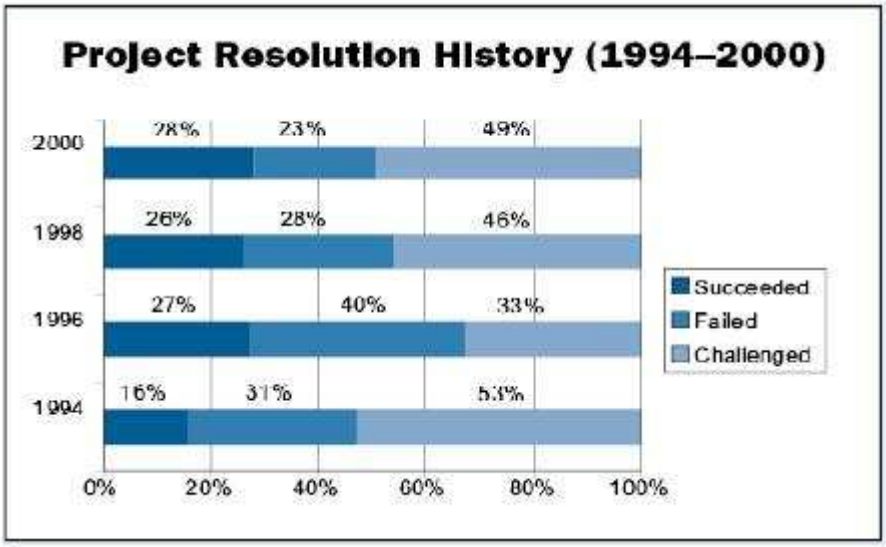

Figura 2: Taxa de sucesso nos projetos de software
Fonte: Standish Group International (2001).

Dos anos 90 até os tempos atuais, muitas mudanças ocorreram na forma de se enxergar a produção de software, novas metodologias e novas formas de gerenciar o desenvolvimento foram criadas. A quantidade de projetos que obtém sucesso no desenvolvimento está crescendo nos últimos anos, em 2002 a quantidade de projetos que obtiveram sucesso já estava em 34% (STANDISH GROUP INTERNATIONAL, 2003), porém estas taxas são preocupantes e ainda estão muito aquém do que é esperado para um projeto de software.

As principais formas de desenvolvimento utilizadas na década de 90 foram através de processos conhecidos como tradicionais, ou pesados.

2.1 Processos Tradicionais de Desenvolvimento de Software

As metodologias de desenvolvimento de software conhecidas como tradicionais são as metodologias mais antigas e consideradas como ultrapassadas por grande parte dos autores (SOARES, 2004). Essas metodologias têm seu foco em etapas fixas como: análise de requisitos, projeto, codificação e testes; com o objetivo de conduzir a produção do software. Devido a essa característica, dentre outras, freqüentemente podem ocorrer problemas como: atrasos na entrega de projetos; orçamento extrapolado; e insatisfação do cliente. Porém, elas surgiram em um contexto diferente do atual, onde a demanda por software não era tão grande e o acesso à tecnologia era excessivamente caro para a maioria dos usuários. Dentre os

modelos conhecidos como tradicionais pode-se citar o Cascata, Prototipagem, e Espiral, todavia, esses modelos ainda são muito utilizados,

Em contrapartida aos processos conhecidos como tradicionais ou pesados, é crescente o interesse pelo desenvolvimento ágil de software.

2.2 Desenvolvimento Ágil de Software

No final da década de 90 surgiu uma nova tendência para o desenvolvimento de software, a qual foi dada o nome ágil (leve). Segundo Fowler (2005) “é provável que a mudança mais relevante na forma de pensar o processo de desenvolvimento de software, nestes últimos anos, tenha sido o aparecimento do termo ‘ágil.".O aparecimento dessa abordagem ocorreu, principalmente, devido ao ritmo acelerado de inovações tecnológicas nas organizações e ambientes de negócios (BOEHN, 2006).

2.2.1 O Manifesto Ágil

No ano de 2001, um grupo de 17 importantes e renomados profissionais na área de software, reuniu-se com o objetivo de discutir valores e princípios que poderiam fazer com que equipes produzissem software com mais rapidez e também com facilidades em meio a freqüentes mudanças. Esse grupo de profissionais se auto denominou aliança ágil, e nesta ocasião formaram o manifesto da aliança ágil (MARTIN, 2002). O conteúdo deste manifesto, conhecido como *Manifesto for Agile Software Development*, contém os principais valores e princípios para nortear o PDS de forma ágil. Cabe salientar aqui, que os organizadores e os ideais contidos no manifesto ágil, não são contrários a adoção de processos no desenvolvimento de software, somente traçam diretrizes para a produção de software com mais agilidade. Os quatro valores básicos do manifesto estão descritos a seguir:

- **Indivíduos e interações** ao invés de processos e ferramentas
- **Software Funcionando** ao invés de documentação
- **Colaboração do cliente** ao invés de negociação de contrato
- **Respostas rápidas as mudanças** ao invés de seguir um plano

A explicação dos valores, e os doze princípios ágeis , que caracterizam as metodologias ágeis, estão contidos no apêndice A.

2.3 Comparação entre Metodologias Tradicionais e Metodologias Ágeis

As práticas abordadas no desenvolvimento ágil não são nenhuma novidade, somente foram reunidas práticas já existentes, e aplicadas ao PDS. A reunião e aplicação dessas práticas pode dar resultados diferentes se comparado a abordagens tradicionais de desenvolvimento. Dentre as principais diferenças entre esses dois paradigmas, segundo Fowler (2005), pode-se destacar (Fig. 3).

Metodologias Tradicionais	**Metodologias Ágeis**
Enfoque maior em processos ou algoritmos.	Enfoque maior nas pessoas envolvidas no projeto.
As pessoas são vistas como recursos.	As pessoas são vistas como indivíduos, onde cada um possui um ritmo e trabalho diferente
São preditivas, uma etapa só começa após o termino da etapa anterior.	São adaptativas e partem do principio de que é necessário saber somente o essencial no início do projeto.
Parte do princípio de que os requisitos do sistema não mudarão, são estáveis.	O conhecimento sobre o problema vai aumentando à medida que o sistema vai sendo desenvolvido.
São mais rígidas e burocráticas, é necessário saber de antemão a maior parte dos requisitos do sistema. Geram sobrecarga durante o desenvolvimento, o que pode comprometer o seu sucesso, prazos de entrega, e custos.	São iterativas e incrementais, com curtos ciclos de desenvolvimento, onde o cliente deve participar durante todo o ciclo de desenvolvimento.
A qualidade do produto é conseguida seguindo processos, desconsiderando o talento individual de cada pessoa.	É incentivado o trabalho em equipe, onde é considerado o talento individual de cada pessoa.

Figura 3 – Comparação entre metolodogias ágeis e tradicionais

3 METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Com surgimento do manifesto ágil, apareceram também algumas metodologias de desenvolvimento, são as metodologias ágeis de desenvolvimento de software. Algumas delas já existiam antes da publicação do manifesto ágil, mas não eram conhecidas por essa nomenclatura. Outras foram desenvolvidas ou aperfeiçoadas para serem caracterizadas como ágeis. A seguir serão demonstradas algumas das principais metodologias ágeis utilizadas no desenvolvimento de software atualmente.

3.1 eXtreme Programming

O *eXtreme Programming* - XP[1], é um dos métodos ágeis mais conhecidos, sendo uma proposta disciplinada para o desenvolvimento de software. É uma metodologia onde se dá prioridade a satisfação do cliente, criação de software com rapidez e alto valor, e também flexibilidade para mudanças no decorrer do projeto. Normalmente, é indicado para equipes de desenvolvimento relativamente pequenas e prazos de entrega inferiores a um ano (LARMAN, 2003). De acordo com Wells (2001), o XP pode ser visto como a construção de um quebra cabeças, onde cada peça pequena parece não ter significado nenhum, mas quando as peças são reunidas, um quadro maior pode ser visualizado.

De acordo com Prikladnicki e Audy (2008), “o XP é composto por quatro fases: PLANEJAMENTO, PROJETO, CODIFICAÇÂO e TESTE. Cada uma dessas fases possui uma série de regras e práticas que devem ser seguidas. Além disso, 12

[1] A metodologia XP foi criada em 1996, principalmente por Kent Beck e Ward Cunningham, que, baseado em experiências anteriores, sugeriram que para um projeto ser bem sucedido, deveria ser pensado, dando ênfase, em quatro dimensões, que são: comunicação, *feedback,* simplicidade e coragem (BECK, 2000).

práticas ao longo dessas fases suportam as quatro dimensões". Na seção 3.1.1 será apresentada as quatro fases. Na seção 3.1.2 e 3.1.3, as quatro dimensões e as doze práticas, respectivamente. A Fig. 4 demonstra as 12 práticas do XP.

3.1.1 As quatro Fases do XP

- **Planejamento -** fase onde o cliente escreve as *user stories* (cartões de estória) Cada cartão descreve uma funcionalidade para ser incluída na primeira iteração do desenvolvimento. Nesta etapa a equipe se familiariza com as práticas e ferramentas que serão utilizadas no projeto. Esta fase pode durar de uma semana a poucos meses.
- **Projeto -** nesta fase as estórias são colocadas em ordem, de acordo com a prioridade de desenvolvimento. Os programadores realizam uma estimativa de esforço para desenvolvimento de cada estória.
- **Codificação -** ocorrem diversas iterações até ser liberado o primeiro software funcional. É criado um primeiro programa, e nas iterações seguintes são adicionadas novas funcionalidades de acordo com a prioridade traçada. O cliente é quem decide quais estórias serão implementadas. São realizados testes, e ao final desta fase o sistema estará pronto para ser utilizado.
- **Teste -** são realizados testes antes de o sistema ser entregue ao cliente. Nesta fase podem ser encontradas novas funcionalidades que poderão, ou não, ser incluídas na versão atual. As idéias postergadas devem ser documentadas como sugestão para serem implementadas futuramente. Após a primeira liberação do sistema, podem ser incorporadas novas funcionalidades através de novas iterações. Esta fase pode requerer a alteração da estrutura da equipe. Ocorre quando não existam mais estórias para serem implementadas, e o cliente se encontra satisfeito com o sistema desenvolvido. É feita a documentação necessária. O encerramento do projeto também pode ocorrer quando o sistema não satisfaz as exigências do cliente, ou quando extrapola os custos de desenvolvimento.

3.1.2 As quatro Dimensões do XP

- **Comunicação -** após uma análise dos problemas ocorridos em alguns projetos de software, constatou-se que a maior causa das falhas foi a falta de comunicação entre os envolvidos. O fato de um programador não comunicar

a equipe sobre alterações no programa, pode gerar grandes problemas. Diversos autores (BECK, 2000; LARMAN, 2003; SOARES et al.,2007), constataram que a comunicação é o primeiro e o mais importante valor do XP e é promovida com um bom relacionamento entre a equipe, e também entre a equipe e o cliente. Deve-se tentar ao máximo realizar a comunicação pessoalmente, evitando-se o uso de meios eletrônicos, como e-mail e telefones.

- **Feedback -**.um bom *feedback* provê avaliações por parte da equipe, e também por parte do cliente, que poderá incluir ou redirecionar requisitos (LARMAN, 2003). A freqüente geração de artefatos operacionais gera avaliações por parte dos usuários, pois o cliente dá um rápido retorno sobre o sistema desenvolvido até o momento, e ajuda a guiar o desenvolvimento de software (NAPHTA, 2007). Quanto melhor for o *feedback* do andamento do projeto, mais tempo se terá para reagir. Podem-se avaliar necessidades com relação a artefatos já produzidos e também as necessidades futuras do projeto.
- **Simplicidade -** faça a coisa o mais simples possível. Deve-se fazer somente o que é necessário para o momento; as preocupações com outras soluções ou soluções mais complexas que dependem mais tempo e custos, devem ser deixadas somente para quando forem expressamente necessárias. Uma dificuldade para os desenvolvedores é conseguir se preocupar somente com o que deve ser feito hoje, sem se preocupar com todo o escopo do trabalho. Quando as preocupações com o futuro são evitadas, a equipe ganha tempo e permite que o cliente tenha acesso ao sistema o quanto antes, podendo assim dar um *feedback* para a equipe (MANHÃES. 2004).
- **Coragem -** mesmo quando há pressão para fazer o contrário, deve-se ter coragem para correr riscos, admitir erros, e eliminar o código que não é mais necessário (POLLICE, 2004). Coragem significa tomar decisões rápidas tornando o desenvolvimento o mais rápido possível. A necessidade de alterar um código que já estava funcionando é normal, e a equipe deve ter coragem para realizar estas alterações. Segundo Manhães (2004), o XP não possui uma fórmula mágica para lidar com a coragem e outros valores, o que muda é somente a forma com que se lida com esses valores.

3.1.3 As doze Práticas do XP

Jogo de Planejamento	Plano de desenvolvimento constantemente atualizado. Beck (2000) afirma que deve sempre haver um diálogo entre a equipe e o cliente.
Versões em Ciclos Pequenos	A cada iteração será agregado um maior valor ao produto de software. São entregues pequenas versões de software ao cliente.
Uso de Metáforas	Comunicação simples entre a equipe e o cliente. Deve haver compreensão entre os dois lados: desenvolvedores e cliente (BECK, 2000).
Desenho Simples	Mesmo sabendo de todo o escopo do projeto, a equipe deve se preocupar apenas com o que deve ser feito no momento, isso traz agilidade para o desenvolvimento.
Desenvolvimento Baseado em Testes	Os programadores escrevem testes para o código, e os clientes escrevem testes de aceitação (POLLICE, 2004).
Reestruturação Constante	Sempre que houver oportunidade, a equipe poderá fazer pequenas alterações com a intenção de melhorar o código desenvolvido, dando maior clareza e compreensão (BORBOREMA, 2007).
Programação aos Pares	a desenvolvimento é realizado em duplas que desenvolvem e testam o código em uma mesma máquina, promovendo revisão e avaliação imediata, diminuindo a probabilidade de ocorrerem defeitos (POLLICE, 2004).
Propriedade Coletiva	Qualquer um tem o direito e a responsabilidade de fazer alterações no código quando for necessário (POLLICE, 2004).
Integração Contínua	O sistema deve ser construído e integrado com a maior freqüência possível, o sistema deve ter sempre uma versão atualizada. Segundo Manhães (2004), cada funcionalidade produzida deve ser integrada a versão final.
40 Horas Semanais	Acredita-se que ultrapassando este número de horas trabalhadas, a equipe não terá a mesma qualidade. Pollice (2004) afirma que não se deve ultrapassar esta barreira por duas semanas consecutivas.
Cliente / Usuários Disponíveis	A equipe de desenvolvimento e o cliente trabalham junto no projeto e devem estar sempre disponíveis para responderem a possíveis dúvidas, e para definir ou alterar alguma exigência (POLLICE, 2004).
Padrão para Codificação	Assim como ocorre verbalmente, deve haver uma boa comunicação, também, através do código. Deve se adotar um padrão para toda a equipe: um conjunto de regras que enfatizam um bom entendimento através do código (POLLICE, 2004).

Figura 4 – As práticas do XP

O XP é uma das metodologias ágeis que tem se destacado pela produção de software de qualidade, o que tem trazido satisfação por parte do cliente, além de trazer um custo de produção competitivo, abalando o mercado de desenvolvimento de software atual. De acordo com Pollice (2004), as práticas foram projetadas para apoiar o desenvolvimento de software, e requerem disciplina para serem postas em prática. Em muitos casos requerem mais disciplina do que pode parecer à primeira impressão.

3.2 Scrum

Scrum é uma metodologia iterativa e incremental que pode ser utilizada tanto para desenvolver ou gerenciar a produção de software, como também de outros tipos de produtos (SOARES et al., 2007). As práticas do Scrum podem ser aplicadas onde há necessidade de pessoas trabalharem em conjunto, com o objetivo de atingir um resultado em comum. Ele não prescreve uma técnica específica para o desenvolvimento, e por este motivo, geralmente é utilizado junto com outras metodologias, como o XP, por exemplo (SLIGER; BRODERICK, 2008). Segundo Highsmith (2002), o ciclo de desenvolvimento é focado no controle e gerenciamento do projeto.

O fato de não prescrever técnicas específicas para desenvolvimento, o destaca dentre os outros métodos, pois, segundo Schwaber (2004), o Scrum dá uma ênfase maior para o gerenciamento do projeto possuindo atividades como monitoramento, feedback, e reuniões com a equipe, com a intenção de identificar problemas, falhas e impedimentos no processo de desenvolvimento.

As técnicas do Scrum, informalmente, podem ser comparadas a um jogo de rúgbi[2], o Scrum é dividido em uma série de *sprints*, Em um jogo, *sprint* significa uma corrida de curta distância, no Scrum significa uma das etapas pelas qual um projeto progride para completar uma iteração. Esses *sprints* ocorrem em períodos de duas a quatro semanas e são similares as iterações do XP.

Como principais papéis envolvidos no desenvolvimento do Scrum, tem-se o *Product Owner*, que é o responsável por definir e alterar os requisitos, avaliar os

[2] No Rúgbi, um grupo de jogadores trabalha em equipe com a intenção de chegar a um resultado comum. Através deste trabalho em equipe, jogadores podem colocar uma bola quase perdida, novamente em jogo.

resultados e prioridades de cada *sprint*, e pelo retorno financeiro do produto. O *Scrum Team* é a equipe que irá se auto organizar durante o ciclo, podendo decidir em qual ordem serão realizadas as tarefas dentro do *sprint*, para concluir a iteração, e assume a responsabilidade de planejar o seu próprio trabalho. O *Scrum Master* é o gerente da equipe, auxilia a equipe a não desviar o foco do trabalho, ou de interferências externas, participa das reuniões diárias de planejamento, garantindo que o *sprint* planejado será cumprido corretamente (SCHWABER, 2004). A Fig. 5 demonstra o funcionamento do clico de vida do Scrum.

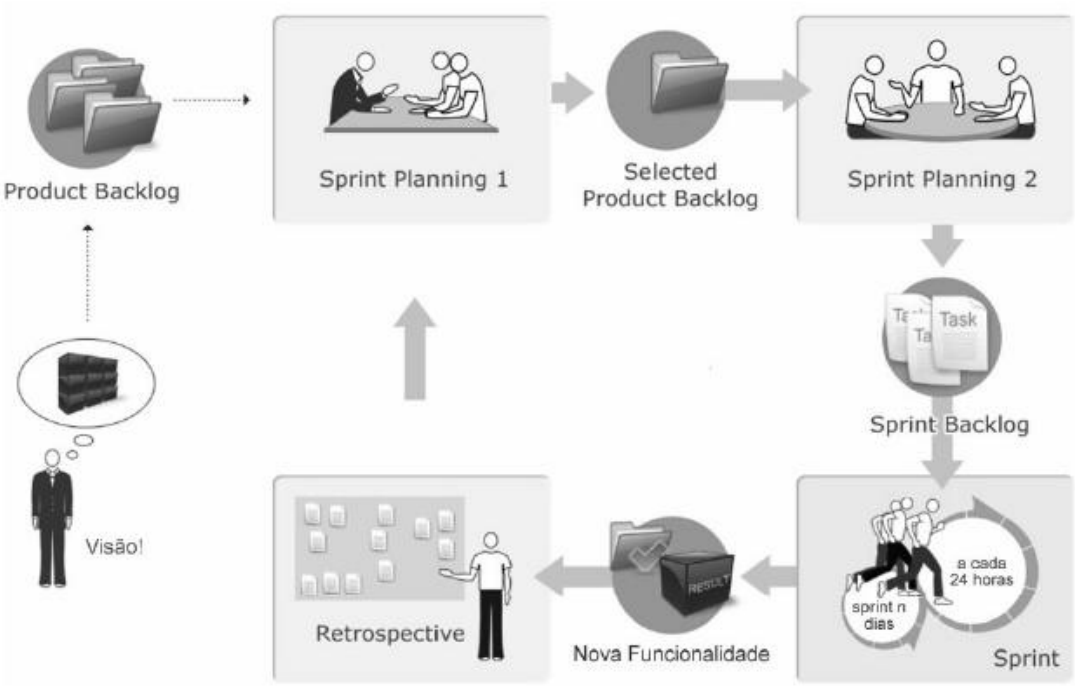

Figura 5 - Ciclo de vida do Scrum
Fonte: Soares et al. (2007).

O primeiro sprint é o *Product Backlog,* onde são definidas as funcionalidades e características esperadas para o produto a ser desenvolvido. O *Product Backlog* é definido pelo *Product Owner* e, por se tratar de um processo iterativo, a cada novo ciclo poderá haver a inclusão, exclusão ou alteração destas funcionalidades (requisitos do produto). Na etapa *Sprint Planning* são realizadas reuniões de planejamento entre o *Scrum Owner* (cliente) e o *Scrum Team* (equipe). Essas reuniões são realizadas antes de cada *sprint,* e são verificados os trabalhos que precisam ser feitos e estimativas para o cumprimento das tarefas. *Sprint backlog* é o trabalho que será desenvolvido no *sprint*, e que, ao final será demonstrado ao cliente.

No *sprint* é onde o produto será desenvolvido com base nos requisitos definidos com *Product Owner*, esta etapa ocorre em intervalos de duas a quatro semanas, onde a cada dia são realizadas reuniões com o objetivo de tratar sobre o andamento do projeto, o que foi feito até o momento e o que ainda necessita ser implementado. Em seguida vem o *Sprint Retrospective*, onde todos os envolvidos no desenvolvimento se reúnem e tratam sobre como está o desenvolvimento até o momento. Perguntas do tipo, "o que foi bem durante o ultimo *sprint*?", ou "o que pode ser melhorado para o próximo *sprint*?" são comumente utilizadas durante esta etapa (SCHWABER, 2004). Como exemplos de empresas que utilizam o Scrum, pode-se citar a Microsoft, Google, Philips, Nokia, John Deere, entre outras.

3.3 Processo Unificado

O Processo Unificado - *Unified Process* - UP[3] é um processo iterativo e incremental, sugere que ciclos de desenvolvimento sejam utilizados para se chegar a soluções adequadas (POLLICE, 2004). O fato de ser incremental possibilita a inclusão, exclusão ou alteração de requisitos, mesmo após o início do desenvolvimento. Segundo Abrahamsson (2002), o ciclo de vida do UP é dividido em quatro fases: concepção, elaboração, construção e transição. Cada uma dessas fases é dividida em diferentes fluxos de desenvolvimento, e ao final de cada iteração é produzido um artefato de software. A duração de uma iteração pode durar de duas semanas até seis meses.

Segundo Wazlawick :

> "O Processo Unificado propõe um processo ágil, com poucos artefatos e pouca burocracia, o que permite o desenvolvimento de software rapidamente, o que interessa ao cliente é o software pronto, e não uma pilha de documentos justificando por que não ficou pronto"(WAZLAWICK, 2004).

Conforme Kroll e Kruchten (2003), O UP contém várias visões de processos prontas para a sua utilização, além de suportar a customização, onde diferentes configurações de processos podem ser montadas através dele, pode suportar tanto grandes equipes como pequenas equipes.

---

[3] O UP foi proposto no ano de 1998 por Grady Booch, James Rumbaugh, e Ivar Jacobson, da empresa Racional, através da verificação das melhores práticas aplicadas ao desenvolvimento de software (LARMAN, 2003). A empresa Racional foi adquirida pela IBM, hoje o Processo Unificado também é reconhecido como Racional Unified Process.

3.3.1 As quatro Fases do Processo Unificado

Concepção - durante esta fase, a equipe (analista) entra em contato com o cliente, com a intenção de buscar as primeiras informações relativas ao sistema. De acordo com Wazlawick (2004), esta fase deve ser rápida e deve produzir um relatório sucinto para avaliar a viabilidade para desenvolver o projeto em questão. Nesta fase é definido o escopo do projeto.

Elaboração - segundo Booch, Rumbaugh e Jacobson (2005), é nesta fase onde é realizado o projeto, implementação e teste da arquitetura, onde incluirá o plano de negócio, requisitos de alto nível e plano de projeto inicial. Nesta fase serão definidas as funcionalidades e a arquitetura a ser desenvolvida. Geralmente esta fase envolve poucas pessoas. São analisados os objetivos gerais do processo, e é decidida a viabilidade do prosseguimento do desenvolvimento.

Construção - é durante esta fase que será criada uma primeira versão do sistema (BOOCH; RUMBAUGH; JACOBSON, 2005). Um produto é desenvolvido de maneira iterativa e incremental. Dependendo do tamanho do projeto, esta fase poderá ser dividida em várias iterações.

Transição - nesta fase o produto é entregue ao cliente. Através de respostas do usuário, são corrigidos possíveis defeitos, e são realizados testes e treinamento dos usuários. São feitas várias repetições de versões beta do produto, e é produzida a documentação para o usuário.

Ao longo das quatro fases do UP, nove fluxos de desenvolvimento vão acontecendo paralelamente. A cada iteração eles são executados de acordo com as características de cada fase. Os diferentes fluxos são: Modelagem de negócio, Requisitos, Análise e Projeto, Implementação, Teste, Implantação, Gerenciamento de Configuração e Alteração, Gerenciamento de Projetos, e Ambiente (ABRAHAMSSON, 2002). A Fig. 6 demonstra o funcionamento do *workflow* do UP com as suas fases e fluxos de desenvolvimento.

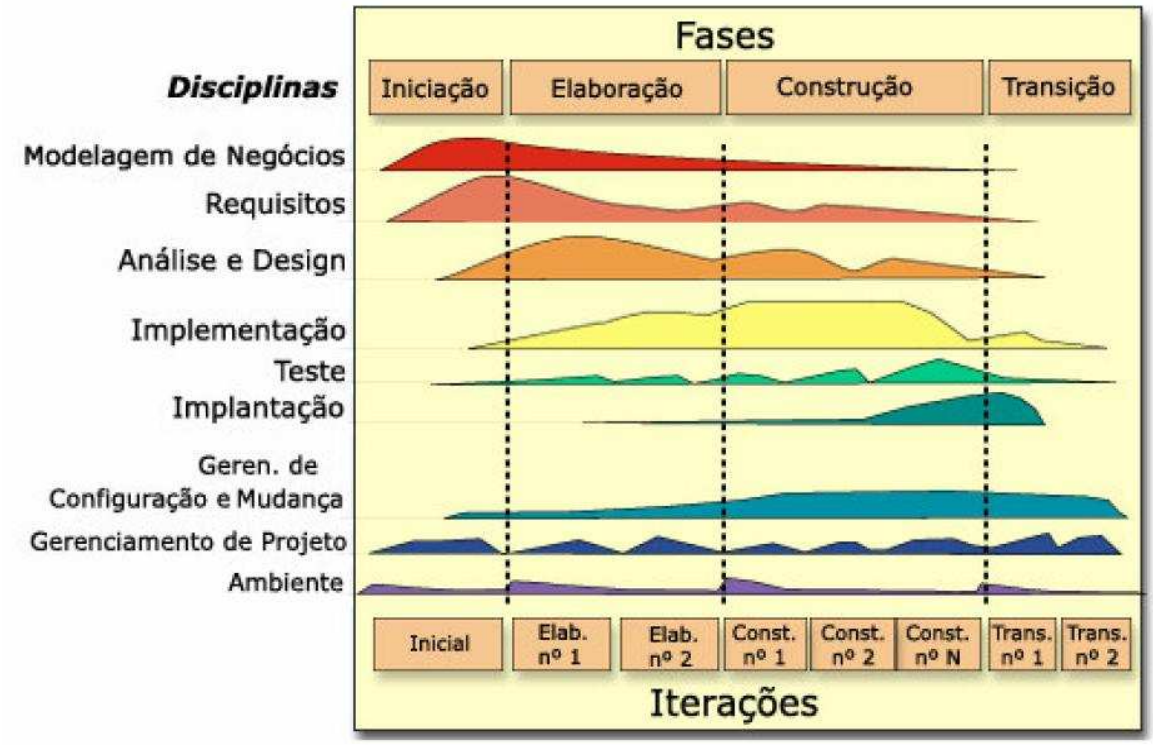

Figura 6 - Workflow do Processo Unificado
Fonte: Souza (2007).

3.4 Metodologia para Desenvolvimento Dinâmico de Sistemas

A Metodologia para Desenvolvimento Dinâmico de Sistemas - *Dynamic Systems Development Methodology* – DSDM, também é uma metodologia que se insere entre as metodologias ágeis, sua proposta é disponibilizar um método para desenvolver sistemas de forma dinâmica. É uma metodologia iterativa e incremental, muito popular entre os países da Europa, que combina ciclo de vida de desenvolvimento de software com ciclo de vida de gerenciamento do projeto (SLIGER: BRODERICK, 2008). A DSDM procura abordar os principais problemas encontrados no desenvolvimento de software, como o pouco envolvimento entre cliente, equipe e encarregados do projeto, dificuldade no cumprimento de prazos de entrega, custo do projeto, ou algum outro problema.

De acordo com Abrahamsson et al. (2002), o princípio fundamental por trás desta metodologia está no fato de, ao invés de fixar as funcionalidades do produto, e somente depois ajustar o tempo e os recursos, é melhor fixar o tempo e os recursos, para depois ajustar o número de funcionalidades adequadamente. A Fig. 7 ilustra o princípio fundamental da DSDM, fazendo um comparativo com as abordagens tradicionais.

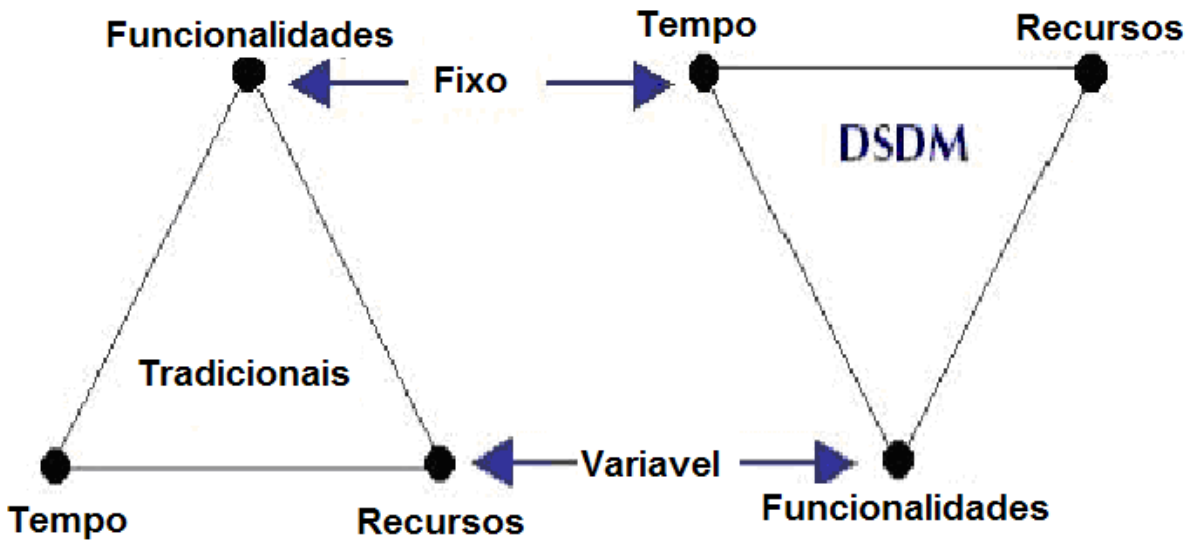

Figura 7 - Principio fundamental da DSDM
Fonte: Adaptado deTeixeira et al. (2005).

A utilização da DSDM é indicada para equipes de dois a seis integrantes, o número mínimo de duas pessoas é porque deve-ser ter um usuário e um fomentador, pelo menos. Já o número de seis pessoas, é um valor que foi atribuído na prática da metodologia. DSDM pode ser aplicada tanto em pequenos projetos, como também em grandes projetos. Quando é utilizada em grandes projetos, o sistema pode ser subdividido em componentes, que podem ser desenvolvidos por equipes pequenas (ABRAHAMSSON, 2002).

Para obter sucesso através desta metodologia, é importante que a equipe de gerentes tenha uma boa receptividade na adoção da metodologia, garantindo a motivação de todos os envolvidos no projeto. Os desenvolvedores devem ter um bom "espírito de equipe", e ser dotados de poder para tomar certas decisões no decorrer do projeto, além de ter acesso a condições de ambiente e tecnologia adequados para o desenvolvimento.

3.4.1 As práticas da DSDM

A DSDM possui nove práticas que formam a sua base de desenvolvimento., sendo similares as práticas do XP. Dentre elas pode-se destacar: a autonomia das equipes para tomar decisões; a freqüente entrega de produtos; e a boa aceitação com relação a mudanças nos requisitos no decorrer do projeto (ABRAHAMSSON, et al., 2002). Estas práticas se adequam aos valores e princípios descritos pelo manifesto ágil.

3.4.2 As cinco fases da DSDM

O ciclo de vida da DSDM é composto por um conjunto de cinco fases. A Fig. 8 demonstra o relacionamento entre cada fase do ciclo de vida.

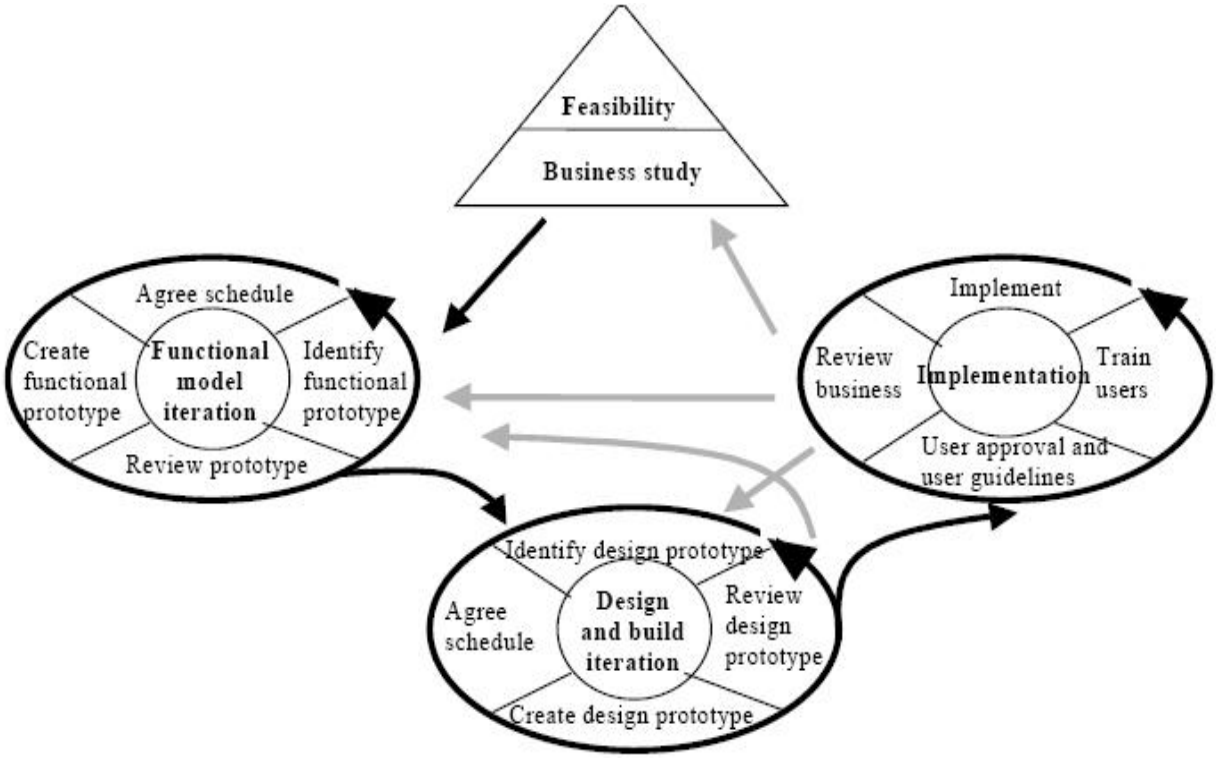

Figura 8 - Ciclo de Vida da DSDM
Fonte: Abrahamsson et al. (2002).

As fases da DSDM são bastante parecidas com as fases encontradas na metodologia UP. O estudo da viabilidade é marcado principalmente pela análise sobre a conveniência de utilização da DSDM, onde pode ser desenvolvido um protótipo do projeto. No estudo do negócio é feita uma análise de alto nível das características do projeto. Cabe ressaltar que nesta fase são realizados *workshops* com todas as áreas que estarão envolvidas no decorrer do desenvolvimento, incentivando a comunicação e o conhecimento do projeto, por toda a equipe. A primeira iteração é feita durante a iteração do modelo funcional, nesta fase os requisitos são refinados, e é feita a listagem das funcionalidades que serão entregues. Na iteração de projeto e construção o sistema é desenvolvido, testado e é entregue uma versão funcional ao cliente. Na fase de implementação é entregue a versão final ao cliente, onde também é realizado o treinamento com os usuários finais.

Cabe destacar ainda que, durante a avaliação dos requisitos, DSDM utiliza a técnica MoSCow[4], para realizar a classificação da prioridade de desenvolvimento.

3.5 Desenvolvimento Guiado por Funcionalidades

O Desenvolvimento Guiado por Funcionalidades - *Feature Driven Development* – FDD, é uma metodologia utilizada para o desenvolvimento e gerenciamento de software. FDD é uma junção do gerenciamento de projetos (*lightweight development approach),* desenvolvido por Jeff De Luca, com a experiência de análise e modelagem orientadas a objetos (*feature-oriented object modeling), de* Peter Coad (SLIGER; BRODERICK, 2008). Segundo Palmer e Felsing (2002), FDD consiste em um conjunto de cinco processos seqüenciais, e provê os métodos, técnicas e diretrizes necessárias para a construção e entrega de um sistema. Ao contrário de algumas metodologias ágeis, o FDD indica ser satisfatório, também, para a construção de sistemas críticos.

Como características desta metodologia, ainda pode-se citar a importância da qualidade nas funcionalidades entregues ao usuário, e a freqüente entrega de resultados através do desenvolvimento dividido em iterações. FDD propicia uma boa precisão na rastreabilidade de relatórios, monitoramento detalhado dentro do projeto, com resumos de alto nível para clientes e gerentes. Com a utilização do FDD, é possível saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa de desenvolvimento são sólidos (HIGHSMITH, 2002).

O desenvolvimento guiado por funcionalidades, indica que se deve desenvolver partes do sistema, que atribuam valor para o cliente Estas funcionalidades devem ser produzidas em intervalos menores do que duas semanas, e podem ser comparadas a requisitos funcionais.

3.5.1 Os cinco Processos do FDD

O ciclo de vida do FDD é composto por processos, os quais são executados sequencialmente. A duração de cada ciclo é de, no máximo, duas semanas e ao final deve retornar uma funcionalidade ao cliente. Os cinco processos seqüenciais

[4] A sigla MoSCow significa: ***M****UST have this* (TEM que ter isto), ***S****HOULD have this if at all possible* (DEVE ter isto, se for possível), ***C****OULD have this if it does not affect anything else* (PODE ter isto, se não afetar o resto), e ***W****ON'T have this time but* ***W****OULD like in the future* (NÃO VAI ter isto agora, mas SERIA bom ter no futuro).

do FDD estão ilustrados pela Fig. 9, e explicados logo após, segundo Highsmith (2002), Coad, Lefebvre e Luca(1999).

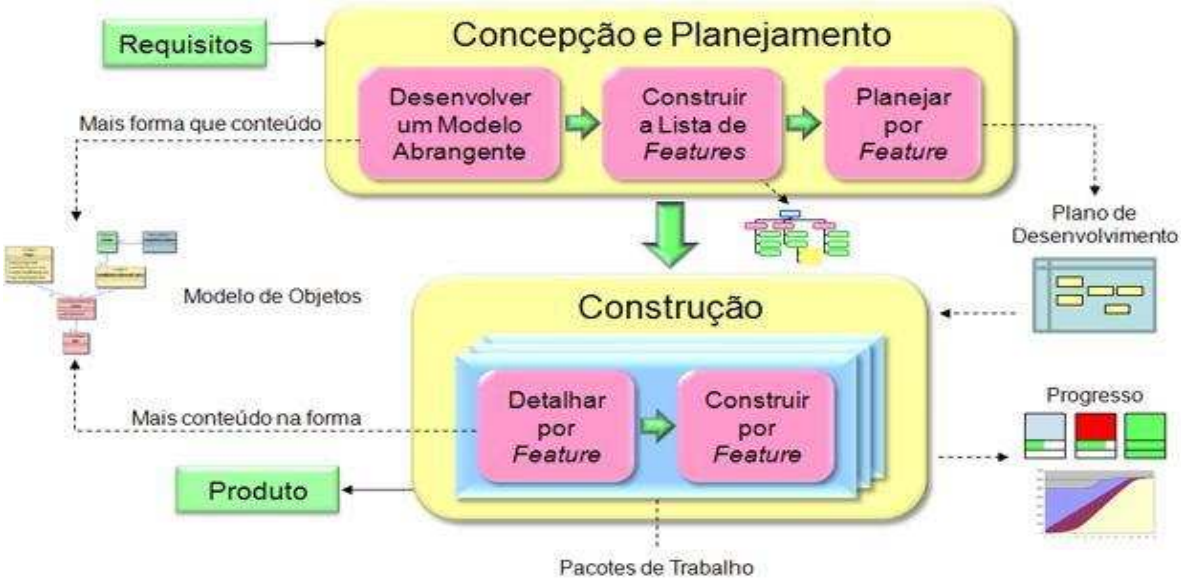

Figura 9 - Os cinco processos do FDD
Fonte: Heptagon (2008).

- **Desenvolver um Modelo Abrangente -** esta fase é focada na forma geral com que o sistema será desenvolvido (seu esqueleto). Especialistas apresentam uma visão geral da área que será modelada, com informações sobre a área de domínio. São desenvolvidos os modelos específicos de áreas sobre o domínio do negócio. Forma-se pequenos grupos (equipes de modelagem), relacionados com o domínio da atividade, que estudam os documentos disponíveis (modelos de objetos, requisitos funcionais, modelos de dados, guias do usuário) esses grupos apresentam seus modelos e é selecionado, por consenso, um dos modelos ou uma combinação dos modelos propostos. Nas equipes de modelagem são feitos rodízios entre os integrantes do projeto, onde todos tem a chance de acompanhar o processo em funcionamento. Esta fase comporta as atividades de formação da equipe de modelagem, estudo dirigido sobre o domínio, estudo da documentação, desenvolvimento do modelo, e refinamento de modelo abrangente de objetos.
- **Construir a Lista de Funcionalidades-** Abrange todo o projeto, são identificadas as *features* (funcionalidades) para todos os requisitos. Uma equipe formada pelos programadores líderes da primeira fase, decompõe-se funcionalmente o domínio em áreas e atividades de negócio. É então formada

uma lista de todas as *features,* que são organizadas de forma hierárquica, com prioridade de tamanho. Esta fase abrange as atividades de formação da equipe da lista de funcionalidades, construção da lista, e avaliação interna e externa com o auxílio dos membros da equipe.

- **Planejar por Funcionalidade -** Nesta etapa é planejada a ordem com que as *features* serão desenvolvidas. É gerado um plano de desenvolvimento baseado na dependência e complexidade das *features*, carga horária e disponibilidade da equipe. Encarregados desta etapa são o gerente de projeto, gerente de desenvolvimento, e programadores-chefe. Dentre as atividades relacionadas é possível citar a formação da equipe de planejamento, determinar a seqüência de desenvolvimento, atribuição das atividades de negócio aos programadores-líderes, e atribuição de classes aos desenvolvedores.
- **Detalhar por Funcionalidade -** As funcionalidades são agendadas para serem desenvolvidas, as quais são atribuídas aos respectivos programadores chefes. Após serem selecionadas as funcionalidades que serão desenvolvidas, é formada uma equipe de *features*, identificando os desenvolvedores que estarão envolvidos nesta etapa. São produzidos diagramas para as *features* selecionadas. Os desenvolvedores devem descrever os prefácios das classes e métodos desenvolvidos. Nesta fase estão envolvidas as atividades de formação da equipe de funcionalidades, estudo dirigido do domínio, estudo da documentação, desenvolvimento dos diagramas, refinamento do modelo de objetos e escrita dos prefácios das classes e métodos.
- **Construir por Funcionalidades -** É produzida uma função com valor para o cliente. Atividades como implementação de classes e métodos, inspeção de código, testes de unidade e atualização da versão, são atribuídas a esta fase.

4 DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

A área da engenharia de software tem evoluído e acompanhado as inovações tecnológicas nos últimos anos. Um grande exemplo desta evolução é a crescente procura pelo desenvolvimento de software, onde as equipes encontram-se geograficamente distantes, e as partes do sistema são desenvolvidas em locais distintos.

Segundo Prikladnicki:

> "A evolução dos ambientes fisicamente distribuídos mostra uma nova tendência em desenvolvimento de software no âmbito mundial. Até pouco tempo atrás não se desenvolviam projetos com equipes dispersas globalmente da forma que tem ocorrido hoje." (PRIKLADNICKI, 2003).

Como exemplo dessa nova tendência, pode-se citar o Brasil, que até o ano de 2003, seu modelo de exportação de software era voltado para a exportação de produtos, devido a uma boa intenção do governo de criar uma marca no mercado internacional. Porém não havia percepção de valor adicional agregado a produtos tecnológicos produzidos no Brasil. Como a mão de obra especializada nos países desenvolvidos é cara, a terceirização de serviços de software (*outsourcing*) começou a ganhar mais valor. Hoje em dia países desenvolvidos têm buscado alternativas nesta área em países como Brasil e Índia (SAUR, 2004).

A globalização e a crescente busca por maior competitividade e redução de custos, tem levado as empresas a adotarem este modelo, ao qual se chama modelo distribuído de desenvolvimento de software - DDS, fazendo empresas atravessarem fronteiras, o que causa um impacto tanto no marketing e distribuição, quanto na forma de concepção, produção, projeto, teste e entrega dos sistemas de software para os clientes (HERBSLEB; MOITRA, 2001). A Fig. 10 demonstra o ranking com os países mais atrativos para *outsourcing* de tecnologia da informação - TI. Fatores como atrativo financeiro, disponibilidade de mão de obra e ambiente empresarial são

levados em conta. Como pode ser observado, o Brasil pulou da 10ª posição, no ano de 2005, para a 5ª posição, no ano de 2007, sendo o país mais atrativo na área, entre todos os países da América do Sul, América Central, e América do Norte.

Figura 10 - *Global services location index*
Fonte: Kearney (2007).

De acordo com Prikladnicki e Audy (2008), uma equipe de desenvolvimento de software pode estar reunida em uma mesma localização física, ou seja, sem uma distância relevante entre os envolvidos, como também pode estar à distâncias nacionais, continentais ou globais, ou seja, em diferentes locais dentro de um mesmo país, entre continentes diferentes, ou em diferentes partes do globo. Em uma equipe global de desenvolvimento pessoas de diferentes nacionalidades trabalham unidas, diante de diferentes culturas e fusos horários, e por um extenso período de tempo (MARQUARDT; HORVATH 2001). A dispersão entre as equipes é classificada de acordo com a maior distância entre as unidades envolvidas no projeto. Por exemplo, em determinado projeto, três equipes na América do Sul, e duas equipes na Europa, trabalham distribuidamente. A distância será classificada como Global, pois se trata de países diferentes localizados em continentes diferentes. Equipes dispersas, a uma distância continental, podem ter até seis fusos horários de diferença. Quando a distância é continental a dispersão física é consideravelmente maior entre as equipes. A Fig. 11 ilustra as diferentes distâncias físicas entre os atores envolvidos no desenvolvimento.

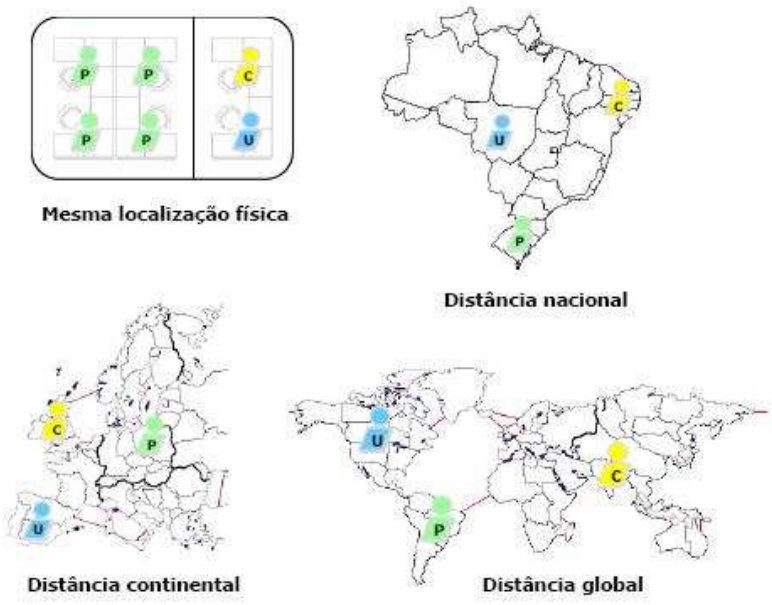

Figura 11 - Distância física entre atores
Fonte: Prikladnicki (2003).

Um fator de grande importância, que é considerado um fator motivador para o rápido crescimento desta forma de desenvolvimento de software, é o fato de que, segundo Herbsleb et al. (2001 apud PRIKLADNICKI; AUDY, 2008), quando a distância entre desenvolvedores, ou equipes de desenvolvedores é maior do que 30 metros, a comunicação entre os mesmos se torna idêntica à comunicação entre desenvolvedores situados a milhares de metros de distância. Ou seja, como se as equipes estivessem trabalhando de forma distribuída, ainda que em uma mesma localização física.

A globalização tem causado grandes impactos nos diversos setores da economia global, não obstante, algo parecido está acontecendo na área de desenvolvimento de software. Estudos demonstram que o DDS, ao mesmo tempo que é um grande desafio, é uma grande promessa (PAULISH; BASS; HERBSLEB, 2005). Fatores como redução de custos, facilidade de encontrar mão de obra especializada, motivam a busca por esta forma de desenvolvimento. Outro fator que tem estreitado fronteiras entre os diversos locais é a facilidade de comunicação que tem evoluído de forma muito rápida nos últimos anos. Como exemplos dessa evolução pode-se citar tecnologias como Voz sobre IP - VoIP, que reduz custos e facilita a comunicação entre pessoas e equipes geograficamente distantes. Hoje em dia conta-se com tecnologias capazes, inclusive, de eliminar custos de

comunicação. A necessidade das empresas manterem seus custos competitivos no mercado global, o considerável crescimento de grandes empresas e equipes virtuais, e a pressão do mercado pela exploração de mão de obra especializada aliada a diferentes fusos-horários, são enumerados por Herbsleb e Moitra (2001) como os principais fatores motivadores para o DDS. Conforme Karolak (1998 apud Souza, 2007) uma organização virtual é caracterizada por entidades diferentes, trabalhando em locais distintos, como se estivesse no mesmo ambiente. Dois termos muito utilizados nesta área são: *time-to-market,* que são as pressões para reduzir o tempo de entrega de um produto, e *follow-the-sun, que é a* possibilidade de se ter 24 horas contínuas trabalhando com equipes distantes, devido a diferentes fusos-horários.

De acordo com Lima (2007), o mercado mundial de software tende para a descentralização crescente das atividades de desenvolvimento. Com isto há necessidade de ferramentas e técnicas que apóiem os problemas inerentes à distribuição.

4.1 Desafios do DDS

O DDS, também tem encontrado alguns desafios. Segundo Rockenbach (2003), Herbsleb et al. (2001), Damian e Zowghi (2002) os principais desafios estão relacionados com diferenças culturais, distância geográfica, coordenação, comunicação e espírito de equipe. A Fig. 12 demonstra as forças que prejudicam e favorecem o DDS.

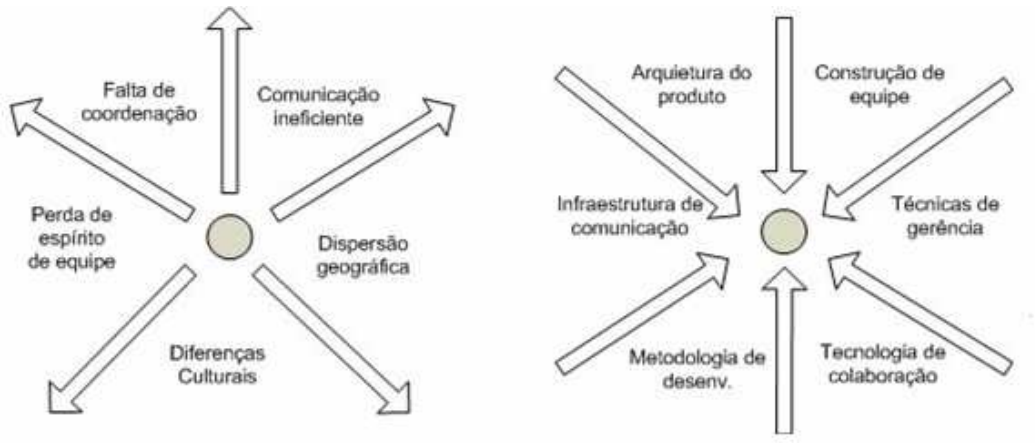

Figura 12 - Forças que prejudicam e favorecem o DDS
Fonte: Adaptado de Carmel (1999).

4.1.1 Forças que Prejudicam o DDS

- **Diferenças Culturais -** de acordo com Prikladnicki e Audy (2008), o gerenciamento das diferentes culturas é fundamental para um bom funcionamento de uma equipe distribuída. É importante estar atento às mais variadas expectativas existentes nas diversas culturas, pois as mesmas diferem na forma de comunicação, reações e atitudes frente a diversos problemas, ou a formas com que lidam com diferentes hierarquias. Ainda segundo Prikladnicki e Audy (2008), a produtividade de uma equipe distribuída, aumenta quando as mesmas desenvolvem uma cultura comum. Equipes bem sucedidas criam suas próprias culturas. O aumento da distância geográfica pode aumentar essas diferenças culturais, porém, pode haver diferenças culturais onde a distribuição geográfica é pequena (Holmstrom et al. 2006). Problemas provenientes de diferenças culturais podem ser encontrados até mesmo em equipes não distribuídas.
- **Dispersão Geográfica -** segundo Carmel (1999), no desenvolvimento distribuído se lida com dois tipos de dispersão: a dispersão geográfica, que é relativa a distância física entre os envolvidos; e também dispersão temporal, que é relativa aos diferentes fusos horários, pelo qual cada equipe estará inserida. A dispersão temporal pode vir diminuir a comunicação síncrona entre os componentes envolvidos. Já a dispersão geográfica dificulta a comunicação, essa dificuldade pode reduzir a comunicação entre as partes envolvidas.
- **Coordenação -** no desenvolvimento distribuído há uma dificuldade de coordenação por diversos motivos, dentre eles, a dificuldade de comunicação, diferenças entre culturas, ou variações entre as tecnologias empregadas. A gerência de configuração é um dos grandes desafios encontrados em ambientes distribuídos. Coordenar o desenvolvimento de cada localidade com desenvolvimento do produto como um todo, pode vir a ser complexo (PRIKLADNICKI; AUDY, 2008).
- **Comunicação -** a comunicação é um fator, ao qual se deve dar muita atenção no DDS. É recomendado que todas as equipes envolvidas se comuniquem umas com as outras (PAULISH; BASS; HERBSLEB, 2005).

Segundo Prikladnick e Audy (2008), nem sempre é possível realizar reuniões presenciais com todos os envolvidos no processo. E, conforme Damian et al. (2006) os desafios apontados no DDS apontam para a utilização de processos dependentes de comunicação informal.

- **Espírito de Equipe -** de acordo com Carmel (1999), deve-se ter um grande cuidado com o trabalho em equipe, pois problemas de falta de confiança podem levar a sua desestruturação.

4.1.2 Forças que Favorecem o DDS

- **Infra-estrutura de Comunicação -** para realizar o desenvolvimento de software distribuidamente, deve-se ter bons meios de comunicação, garantindo a confiança entre as equipes envolvidas. É importante que os meios de comunicação sejam de alta velocidade. Segundo Prikladnicki e Audy (2008), as tecnologias de comunicação devem possibilitar novas formas de comunicação formal e ampliar a comunicação informal entre os componentes envolvidos no desenvolvimento. Atualmente a maioria dos países disponibiliza meios de comunicação com bom desempenho.
- **Arquitetura do Produto -** a arquitetura de software é um fator determinante na efetividade e redução das dificuldades do DDS e deve se basear no princípio da modularidade, considerada a principal forma de resolver e alocar tarefas complexas distribuidamente;
- **Tecnologia de colaboração -** são usadas para ampliar a comunicação entre os membros. Podem ser divididas em tecnologias genéricas, ou tecnologias de colaboração para suporte as atividades e engenharia de software.
- **Técnicas de gerência de projetos -** aspectos importantes como gerência de conflitos, coleta de métricas, formas de reconhecimento e bonificação e escolha de um gerente apropriado;
- **Processo de desenvolvimento -** a existência de um processo de desenvolvimento comum é fundamental para a sincronização da equipe. Um dos principais objetivos de se ter um processo comum às equipes, é a sincronização de atividades (PILATTI; PRIKLADNICKI; AUDY, 2005).
- **Formação de equipes -** conhecimento do papel de cada um e da equipe como um todo. Formação de equipe, com papéis bem definidos é importante para o DDS.

4.2 Dimensões de um Projeto de Desenvolvimento Distribuído

O DDS demonstra alguns fatores que devem ser considerados durante a realização de um projeto. Este modelo foi proposto por Evaristo e Scudder (2000) e pode ser aplicado em qualquer tipo de desenvolvimento distribuído, não somente para o desenvolvimento de software. As Diferentes dimensões estão detalhadas na Fig. 13 e são explicadas logo após, segundo Evaristo e Scudder (2000).

Figura 13 - As diferentes dimensões do DDS
Fonte: Evaristo e Scudder (2000).

- **Confiança -** é importante que todos os *stakeholders* envolvidos no projeto distribuído tenham confiança uns nos outros, assim poderá haver espírito de equipe. Tanto a confiança quanto o espírito de equipe são dificuldades encontradas em DDS. Devem ser buscadas formas para incentivar este valor entre a equipe.
- **Nível de Dispersão -** dispersão está relacionada com distância física entre os envolvidos. De acordo com Prikladnick (2003), a dispersão entre os *stakeholders* pode ser relacionada com a equipe de projeto, clientes e usuários. Quanto maior a dispersão entre os envolvidos, maiores são as dificuldades encontradas para realizar um monitoramento das atividades.
- **Sincronização -** é a capacidade de pessoas distantes trabalharem paralelamente, em um mesmo projeto. Quando se lida com fusos-horários idênticos essa dificuldade é reduzida, pois possibilita que as equipes trabalhem no mesmo horário com maior facilidade.
- **Complexidade -** a complexidade do projeto a ser desenvolvido está diretamente ligada com a dificuldade em realizar o desenvolvimento

distribuído. Fatores como tecnologia empregada, tamanho do projeto e objetivos esperados, influenciam na complexidade do projeto.

- **Tipos de Atores (*stakeholders*) -** em um projeto distribuído, se lida com vários grupos de *stakeholders*, que podem ser desenvolvedores, pessoas ligadas a gerência, clientes, ou qualquer outra pessoa que tenha ligação com o projeto em qualquer localização da equipe. Quanto maior o número de *stakeholders* envolvidos, maiores podem ser as dificuldades encontradas no projeto distribuído.
- **Processo de Desenvolvimento -** é importante que exista um processo comum a todas as equipes, assim um desenvolvedor pode saber a qualquer momento em que ponto está o projeto (PRIKLADNICKI; AUDY, 2008). Segundo Pillati, Prikladnicki e Audy (2005), um dos maiores objetivos para se ter um processo comum para todas as equipes, é a sincronia das atividades. Não é conveniente para um gerente, coordenar um projeto onde cada equipe utiliza um processo diferente. Existe diversas metodologias disponíveis para serem utilizadas ou adaptadas para projetos distribuídos. Processos podem auxiliar em questões referentes à comunicação, qualidade e planos do projeto.
- **Tipo de Projeto -** o tipo de projeto a ser desenvolvido influi diretamente na forma como o mesmo será gerenciado. Diferentes técnicas de gerência podem ser aplicadas, não somente ao projeto, mas também ao diferente número de stakeholders de cada projeto.
- **Procedimentos e Padrões -** a padronização dos procedimentos utilizados em um projeto distribuído deve ser parte da cultura da empresa. Diferentes padrões, como técnicas de comunicação, de gerência, ou de desenvolvimento podem ser adotados pela organização.
- **Distância Percebida -** é a possibilidade, ou não, de se encontrar qualquer pessoa envolvida no projeto. Os integrantes podem estar fisicamente distantes, ou fisicamente próximos. A distância percebida afeta as atividades gerenciais do projeto.
- **Cultura -** diferenças culturais podem ser relacionadas tanto para aspectos específicos de cada país, quanto diferentes culturas organizacionais. O gerenciamento das diferentes culturas é essencial para o sucesso do

desenvolvimento distribuído. Em projetos distribuídos realizados no mesmo país, os fatores culturais podem não afetar tanto o andamento do projeto. Segundo Prikladnicki e Audy (2008), equipes bem sucedidas costumam criar uma cultura comum para a equipe, aumentando a produtividade, efetividade e satisfação.

4.3 Classificações do DDS

A principal característica do DDS é a distância entre as diversas partes envolvidas no processo de desenvolvimento. A distância pode ser: física, onde os atores (*stakeholders*) estão em locais distintos, ou temporal, ou estão em fusos horários diferentes. O desenvolvimento distribuído ainda pode ser classificado em diversos níveis de dispersão, como ilustrado na Fig. 14.

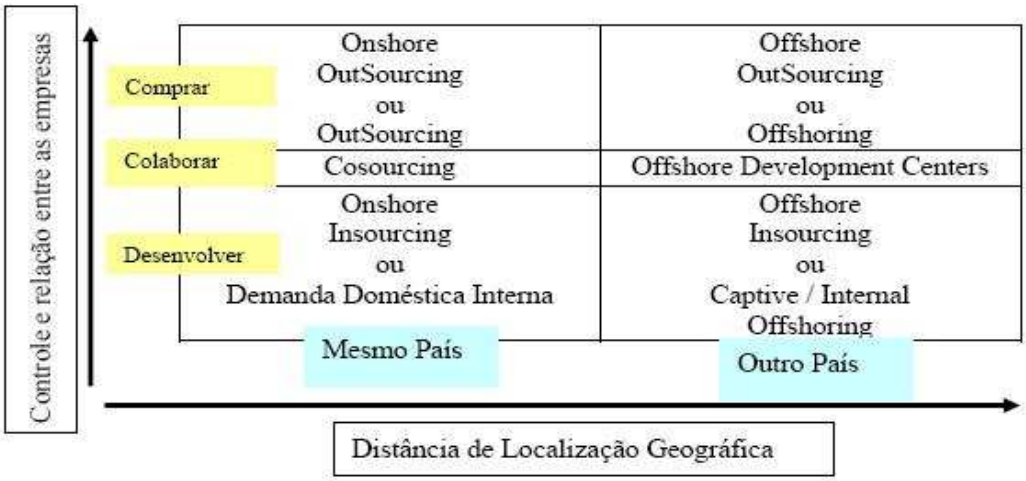

Figura 14: Classificação da relação de sub-contratação
Fonte: Evaristo e Scudder (2000)

O termo terceirização vem sendo muito utilizado nas mais diversas áreas, desde o comércio até a indústria, onde empresas buscam por soluções com melhores custos através da transferência parcial ou total de determinada atividade para terceiros. Segundo Khan (2003), esta terceirização na área tecnológica, também conhecida como *outsourcing*, é definida como uma significativa contribuição de empresas externas aos recursos humanos ou físicos ligados a componentes de TI. E, de acordo como Pilatti (2006), é a prática de contratar uma organização externa para o desenvolvimento de determinado produto, ao invés do mesmo ser

desenvolvido na própria sede da empresa, tendo como objetivo principal a redução de custos.

O termo *Insourcing* surgiu a partir do termo *outsourcing*, logo, pode ser considerado como uma variação do mesmo. Utiliza-se o termo *Insourcing* quando se tem unidades semi-independentes, prestando serviços para outras unidades dentro da própria empresa, com preços e condições pré-acordados ou, também pode significar a retenção de um determinado serviço na própria empresa, no setor de demanda de serviço (GEFEN, 2003). O serviço pode ser requisitado em outra parte da empresa. Mesmo sendo a mesma empresa, as unidades de negócio podem estar localizadas em outro local, sendo até mesmo em outro país, neste caso são conhecidos como centros globais de desenvolvimento de produto, ou prestação de serviço (REPONEN, 2002). Os termos *offshore* e *onshore* indicam, respectivamente, a contratação de serviços em um país diferente, ou no mesmo país.

Segundo Prikladnicki e Audy (2008), os modelos apresentados na Fig. 13 podem ser definidos como:

- ***Onshore insourcing -*** quando o processo de distribuição ocorre no mesmo país. É considerado uma das mais simples formas de DDS, a forma mais conhecida pelos gerentes de projetos. Uma empresa mantém um departamento dentro da própria empresa ou em uma subsidiária localizada no mesmo país *(insourcing),* e este provê os projetos de desenvolvimento de software através de projetos internos.
- ***Onshore outsourcing -*** o termo *outsourcing* indica a contratação de uma empresa terceirizada, onde esta realiza o desenvolvimento dos produtos de software ou serviços para a empresa contratante. A empresa contratada se encontra no mesmo país da empresa contratante *(onshore).*
- ***Offshore outsourcing -*** quando uma determinada empresa realiza a terceirização do desenvolvimento dos produtos de software ou serviços, mas a empresa contratada não está situada no mesmo país da empresa contratante.
- ***Offshore insourcing -*** indica a criação de uma empresa subsidiária, para prover os serviços de desenvolvimento de software e (ou) serviços. A empresa subsidiária deve se localizar obrigatoriamente no exterior, ou seja, em um país diferente da empresa contratante *(offshore).*

Dentro dos conceitos apresentados anteriormente ainda encontra-se termos como *co-sourcing* que representa dois provedores de serviços trabalhando em conjunto para um mesmo cliente, e *offshore development centers*, quando equipes geograficamente distribuídas trabalham colaborativamente de forma remota. O termo *nearshore* também é utilizado, e caracteriza uma forma de desenvolvimento em países pouco distantes, como Brasil e Argentina, ou Estados Unidos e Canadá.

4.4 Desenvolvimento Open Source

Esta seção apenas demonstra um exemplo de aplicação distribuída, fazendo um contraste com as metodologias ágeis vistas no capítulo 3. De acordo com Fowler (2005), *open source* é um estilo de software, e não um processo. O que chama a atenção é que na comunidade open source não existe um estilo específico de se realizar o desenvolvimento, e as metodologias ágeis têm se tornado bastante popular neste meio.

O cenário atual de comercialização de produtos de software apresenta duas particularidades ainda pouco discutidas: a atuação em mercados de software livre e DDS. Não são poucos os que enxergam no software livre uma oportunidade única para concretamente modificar o cenário atual da indústria mundial de software, dominado por algumas poucas multinacionais. Mas toda grande mudança é igualmente acompanhada de desinformação, inseguranças e principalmente de riscos.

O software livre, ao contrário de seu mercado, hoje em dia é um fato, e vem sendo alvo de grande discussão no meio acadêmico e comercial. Nesse modelo, a fábrica de software é composta por uma equipe fixa, geograficamente distribuída, que será responsável pela captação de negócios e desenvolvimento inicial de um produto de software livre, que poderá ser posteriormente disponibilizado para que uma comunidade de desenvolvimento participe no processo de melhoria do mesmo.

As metodologias ágeis têm se tornado muito populares entre os desenvolvedores de software livre. Segundo Raymond (1998), software livre, normalmente, é desenvolvido por times auto-organizados, que raramente se reúnem presencialmente, coordenando suas atividades através de comunicações baseadas em computadores.

5 DESENVOLVIMENTO ONSHORE INSOURCING A PARTIR DE METODOLOGIAS ÁGEIS

Atualmente, com a globalização existe uma grande necessidade de agilidade durante o desenvolvimento de produtos de software. Nesse novo cenário, muitas empresas estão adotando o modelo DDS. Para uma empresa que irá aderir à forma de DDS, é recomendável que comece da forma mais simples (*onshore insourcing*), com menos desafios críticos. A partir de uma primeira experiência com um modelo simples, a empresa poderá optar por modelos mais complexos como *offshore* e (ou) *outsourcing*. Também poderá começar com modelos simples e projetos simples, e depois migrar para projetos mais complexos. O foco deve ser na aprendizagem. À medida que a empresa vai adquirindo experiência no desenvolvimento distribuído poderá surgir oportunidades para outros cenários de desenvolvimento (PRIKLADNICKI; AUDY, 2008).

O modelo de desenvolvimento *onshore insourcing* possui facilidades como a pouca diferença temporal e geográfica, e menores problemas relacionados à comunicação. As equipes estando localizadas no mesmo país, geralmente, não possuem grandes fusos-horários de diferença (não havendo a prática de trabalho *follow-the-sun*), a comunicação não é tão prejudicada quanto no *offshore*, pois em desenvolvimento *onshore,* na maioria dos casos, as equipes se comunicam pelo mesmo idioma e não possuem diferenças culturais relevantes. Essas facilidades são de extrema importância para o desenvolvimento ágil de software que, de acordo com as características do manifesto ágil, dão muita ênfase para a comunicação e trabalho em equipe.

O foco deste trabalho é direcionado para o DDS do tipo *onshore insourcing* (modelo de desenvolvimento distribuído demonstrado na seção 4.3). A análise das outras formas de desenvolvimento distribuído está fora do escopo do trabalho. Cabe

ressaltar que a prática *onshore insourcing,* embora seja uma prática em muitas empresas, ainda carece de estudos relacionados a metodologias que lhe dêem suporte ao processo de desenvolvimento. Segundo Rocha, Oliveira e Vasconcelos (2004), o *outsourcing* requer maior formalismo por parte dos desenvolvedores, isso dificulta a participação do cliente no acompanhamento do andamento do processo. Em muitos casos, certificações em modelos de qualidade são exigidas da empresa desenvolvedora, tornando necessária a utilização de processos bem definidos durante o desenvolvimento.

Nesse trabalho será abordado formas para se desenvolver software distribuidamente, utilizando as melhores práticas encontradas nas principais metodologias ágeis de desenvolvimento. Será feita uma análise das metodologias estudadas no capitulo 3, onde serão destacadas as características destas metodologias, aplicáveis a *onshoring Insourcing*, modelo caracterizado, principalmente, pela colaboração de esforços entre empresas (ou repartições de empresas) distantes fisicamente. Após, serão destacadas as melhores práticas, realizando uma sugestão de metodologia ágil para ser aplicada nesta forma de desenvolvimento distribuído.

A análise dos dados será feita a partir de técnicas específicas. Uma delas é a análise qualitativa, onde os textos são analisados de forma sistemática, a partir do material, e guiados por teoria com explicações adicionais para aumentar a compreensão ou esclarecer determinado segmento. A outra técnica é a *grounded theory*, onde parte-se da suposição, que durante a coleta de dados, o pesquisador reflete e analisa os conceitos implícitos, desenvolvendo, aprimorando e interligando os conceitos teóricos (MAYRING, 2002). A análise quantitativa também será utilizada em casos onde não exista um critério válido para escolha de uma determinada prática, e se tenha exemplos de sucesso na aplicação, ou em casos onde mais de uma metodologia utiliza uma determinada técnica. Como exemplo pode-se citar as metodologias UP e FDD, ambas se baseiam nas melhores práticas, e ambas sugerem a utilização de *Unified Modeling Language*- UML[5] para realizar a modelagem. A análise textual será realizada sobre os assuntos expostos nos capítulos 2, 3 e 4. A seguir será tratado a melhoria de processos de software.

5 UML é uma linguagem gráfica utilizada para visualização, especificação, construção e documentação de artefatos de sistema.

5.1 Melhoria dos Processos de Desenvolvimento de Software

A realização do desenvolvimento de software consiste em vários fatores necessários para que se chegue ao final do desenvolvimento com sucesso. Uma boa equipe de desenvolvimento, ambiente de trabalho adequado, um bom relacionamento entre os integrantes, a escolha de um bom processo de desenvolvimento, além de outros fatores, são essenciais para se chegar a um produto de software de qualidade. Como foi observado no capítulo 2 e 3, existem vários processos, ou metodologias, disponíveis para guiar o desenvolvimento de software. Cada uma dessas metodologias foi criada baseada nas melhores práticas, por seus autores. Porém, não é possível dizer que exista uma metodologia ideal para todas as atividades, ou para todos os tipos de projetos. Uma determinada metodologia pode ser ideal para uma determinada atividade, ao mesmo tempo em que para uma outra atividade não traga bons resultados. Existem projetos dos mais variados tamanhos, com diferentes disponibilidades de recursos técnicos e humanos, além de diferentes objetivos, prazos e custos.

O DDS, (capitulo 4), trouxe muitas facilidades, e também muitos desafios para o ciclo de desenvolvimento de software. Muitas das dificuldades encontradas em um projeto co-localizado são tratadas de formas particulares em cada metodologia. Há uma grande dificuldade de se encontrar uma metodologia ágil que possua características para tratar dos principais desafios encontrados no DDS, surgindo a necessidade de uma análise mais aprofundada das práticas encontradas nas principais metodologias ágeis de software, com a intenção de sugerir uma boa forma de se desenvolver, com agilidade, software em ambiente distribuído. A partir do estudo bibliográfico, serão incorporadas as práticas e técnicas das metodologias ágeis, e meios para suprir possíveis deficiências dessas metodologias, que podem ser aplicadas ao DDS.

Cada processo ou metodologia para desenvolvimento de software, possui características próprias. Processos de software são formados por estruturas complexas e necessitam de julgamento humano para a sua utilização. Como foi demonstrado nos capítulos 2 e 3, existem vários processos de desenvolvimento, o que dificulta a sua automatização, fazendo com que muitas empresas desenvolvam processos diferentes para as suas necessidades. O fato de não existir um processo

ideal, possibilita que sejam feitas melhorias nos processos existentes (SOMMERVILLE, 2003). Muitas empresas que se dedicam ao desenvolvimento de software investem na melhoria dos seus processos de desenvolvimento (TAMAKI; HIRAMA, 2007). Segundo Sommerville (2003), a melhoria dos processos de software, significa compreender os processos existentes e modificá-los para que possam atender as necessidades relativas à qualidade do produto, custo e tempo de desenvolvimento. A Fig. 15 demonstra um modelo iterativo para melhoria de processos. Cada etapa será explicada, baseado em Sommerville (2003). Através deste modelo será feita a análise comparativa das metodologias ágeis, de forma a se criar uma sugestão aplicável ao desenvolvimento distribuído *onshore insourcing.*

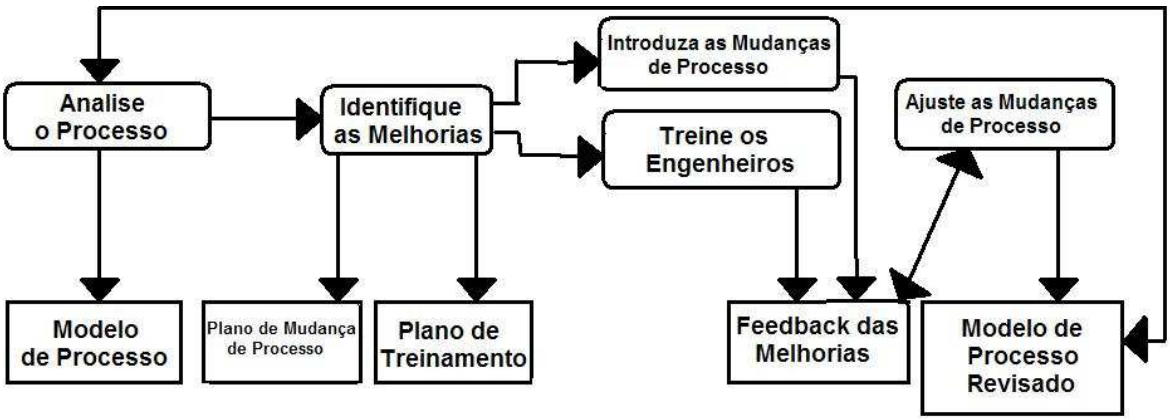

Figura 15 - Modelo de melhoria de processo
Fonte: Sommerville (2003)

Modelo de processo - envolve o estudo das metodologias ágeis que servirão como base para a posterior sugestão para o desenvolvimento distribuído *onhore insourcing.*

Análise de processo - envolve a análise das metodologias ágeis estudadas no capítulo 3. Serão analisadas as características de cada uma destas metodologias. A análise quantitativa dessas, permite fazer uma avaliação objetiva dos benefícios (ou problemas) de cada uma delas pode trazer ao desenvolvimento *onshore insourcing.*

Identificação de melhorias - se ocupa na utilização dos resultados obtidos na etapa de análise. Neste caso são identificadas as características das metodologias ágeis relacionadas principalmente com as forças que favorecem e prejudicam a prática do DDS. O enfoque está na eliminação dos principais problemas, propondo novos procedimentos, métodos ou ferramentas para corrigi-los.

Introdução de mudança de processo - se preocupa com a introdução das mudanças vistas na etapa anterior, integrando com outras atividades. As mudanças devem ser compatíveis com as outras atividades inerentes ao processo, e com os padrões organizacionais do tipo de projeto.

Treinamento em mudanças de processo - é necessário haver um treinamento, para se obter os benefícios das mudanças e melhoramentos dos processos. Se as mudanças nos processos forem impostas sem um treinamento adequado, os efeitos das mudanças podem não resultar na melhoria da qualidade do produto desenvolvido.

Ajuste de mudanças - assim que as mudanças forem introduzidas, pode ser necessário que sejam feitos alguns ajustes, para que os problemas menores possam ser descobertos e novas modificações sejam propostas.

Modelo de processo revisado - ao final do ciclo de análise de melhorias nas metodologias ágeis, tem-se um modelo de sugestão ágil para ser aplicado ao desenvolvimento distribuído *onshore insourcing.*

É necessário que sejam feitas melhorias nos processos de desenvolvimento de software para que os mesmo supram todas as necessidades de um determinado projeto. Um processo ágil, para ser aplicado ao desenvolvimento distribuído, necessita manter certas formalidades e ao mesmo tempo deve ser o mais leve possível. O processo deve atender as exigências do desenvolvimento distribuído, mantendo as características desenvolvimento ágil.

Além de favorecer melhorias, um processo deve ser de fácil utilização. Atualmente o mercado não se preocupa somente com a qualidade do produto, atributos como usabilidade, portabilidade, manutenibilidade e reusabilidade também devem ser levados em consideração. Em muitos casos consideram-se processos como sendo ideais, onde se segue cegamente suas recomendações, o que pode levar a dificuldades no decorrer da implementação (MENDONÇA, 2006). Porém, cada projeto tem suas peculiaridades, fazendo com que não exista um processo ideal para todos os projetos.

A seguir será iniciada uma análise comparativa entre as metodologias ágeis com relação as principais características do DDS *onshoring insourcing.*

5.2 Análise das Metodologias Ágeis com relação ao DDS *onshore insourcing*

Baseado nas características das metodologias ágeis demonstradas pelo estudo bibliográfico, serão sugeridas as melhores características destas metodologias, formando assim a base para o DDS *onshore insourcing.* Muitas destas práticas ágeis foram validadas em outros trabalhos relacionados com desenvolvimento distribuído, onde algumas das metodologias citadas foram aplicadas e analisadas de forma integral nos respectivos estudos.

Através da análise das metodologias ágeis, nota-se que a fomentação do espírito de equipe é muito importante. Esta característica deve ser suprida até mesmo em equipes dispersas fisicamente. Uma boa interface de comunicação pode ajudar a diminuir os problemas de comunicação entre as equipes durante o desenvolvimento, ajudando a unir as equipes, porém é necessário que a comunicação entre as mesmas seja incentivada continuamente. Segundo Kruchten (2004), metade do PDS é a comunicação entre as pessoas. Não se deve deixar formar gargalos de comunicação ente os desenvolvedores. É necessário ter disponibilidade de acesso a qualquer integrante do processo de desenvolvimento no momento em que for necessário. Todos os processos de desenvolvimento para equipes distribuídas apontam para o uso de comunicação informal (KRUCHTEN, 2004; URDANGARIN, 2008). Por outro lado, acesso a diferentes meios de comunicação não significa que as equipes estão utilizando-os de forma adequada. Fowler (2006) aponta bons resultados com a utilização de embaixadores[6] no seu projeto distribuído, onde sempre havia um representante da matriz da empresa presente nas unidades de desenvolvimento, este fato melhorou consideravelmente os níveis de comunicação entre as equipes distantes.

Segundo Prikladnicki e Audy (2008), o uso de grupos de *e-mail*, *chat* eletrônico e o uso de fórum de discussões, também têm mostrado grande valor no incentivo à comunicação. De acordo com Urdangarin (2008), estas formas de comunicação foram aprovadas por equipes distribuídas utilizando a metodologia XP.

Quando se fala em desenvolvimento global de software, fica praticamente inviável a comunicação face-a-face entre todos os *stakeholders* envolvidos no

6 Por embaixadores, entendem-se representantes da matriz da empresa, presentes nas diferentes unidades de desenvolvimento.

processo. Como a distância entre as equipes é grande, essas reuniões se tornam demasiadamente custosas para serem realizadas com equipes inteiras. Para solucionar este problema, um integrante de cada equipe pode ser selecionado para ser intermediário entre os times de desenvolvimento. Este integrante (líder de equipe) auxilia a abrir canais de comunicação entre todos os *stakeholders.* (DAMIAN, 2002; PRIKLADNICKI, 2003). Em projetos *onshore insourcing*, quando a distância não é consideravelmente grande, estas reuniões podem ser feitas também com integrantes da equipe de desenvolvimento, deve ser realizada uma reunião com os líderes de equipe sempre ao final de cada iteração. Estas reuniões podem ser realizadas através de *Workshops* no início do projeto, característica defendida pela metodologia DSDM, que se mostra válida para a integração da equipe, além de fazer com que todos os envolvidos fiquem por dentro de todas as funcionalidades do projeto a ser desenvolvido. Uma série de *workshops,* sendo aplicada no estudo de negócio é importante para entender toda à área de negócio sobre a qual o desenvolvimento ocorrerá (FOWLER, 2005), *e* torna possível que todos os *stakeholders* se conheçam pessoalmente já no início do projeto, quebrando barreiras de comunicação que poderiam ocorrer durante o desenvolvimento. É mais fácil ocorrer uma melhor comunicação através de *e-mail*, por exemplo, onde se conhece o destinatário pessoalmente. O *workshop* também possibilita que dúvidas relacionadas aos requisitos do sistema que poderiam vir a surgir no transcorrer do projeto, sejam sanadas logo no início.

Uma prática do XP que demonstra aceitação, também, no desenvolvimento distribuído, é a programação em duplas (*Pair Programming)*, realizada no contexto de cada equipe. Dentre outras características, a programação em duplas estimula o trabalho em equipe fazendo com que se tenha uma boa comunicação entre as duplas co-localizadas, aumentando assim comunicação entre todos os envolvidos no projeto. *A* programação em duplas já foi aplicada em DDS, onde se obteve bons resultados (URDANGARIN, 2008).

De acordo com Herbsleb (2001 apud PRIKLADNICKI; AUDY, 2008), um dos problemas dos ambientes tradicionais de desenvolvimento, é que os mecanismos para coordenação de trabalho em ambiente distribuído não existem, ou são falhos, afetando a capacidade de colaboração entre os integrantes.

Conforme as características de cada uma das metodologias vistas no capítulo 3, notou-se que o Scrum é a mais voltada para o gerenciamento de atividades como um todo, sendo a metodologia XP voltada para a fase de desenvolvimento, dando pouca atenção para práticas de gerenciamento. Não é aconselhável o uso do XP para projetos longos, pois são tratados apenas aspectos relativos a cada iteração no seu ciclo de desenvolvimento. Apenas as funcionalidades previstas são tratadas. É possível fazer a integração destas duas metodologias para que a deficiência de uma seja suprida com as características da outra (SOUZA, 2007). Também podem-se utilizar práticas de outras metodologias (ou criar novas práticas) para suprir possíveis deficiências em determinada metodologia. Como o Scrum é focado em aspectos organizacionais, por exemplo, ele não especifica a prática da programação em duplas ou propriedade coletiva de código no seu ciclo de desenvolvimento.

Ao contrário do XP, tem-se no UP um processo um pouco mais rígido, onde conta-se com um *framework* com 4 fases bem definidas de desenvolvimento, e disciplinas que guiam o desenvolvimento do software. O XP também possui 4 atividades básicas como o UP, porém estas atividades apenas se utilizam de práticas para obter melhores resultados na fase de desenvolvimento. A partir desta constatação, chega-se a conclusão de que as práticas do XP podem ser utilizadas durante a fase de construção da metodologia UP. Para isto devem ser analisadas quais destas práticas, além da programação em duplas, vista anteriormente, podem ser aplicadas ao DDS. A Fig. 16 ilustra a utilização das práticas do XP no Processo Unificado.

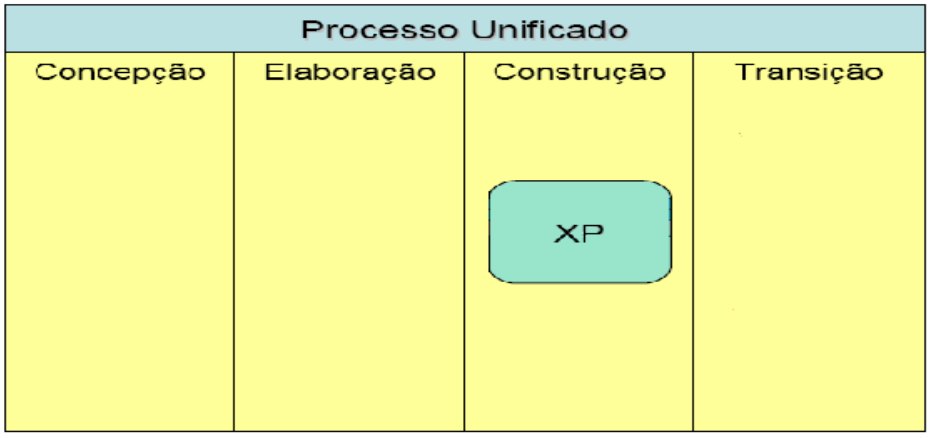

Figura 16 - Utilização de práticas do XP no Processo Unificado
Fonte: adaptado de Heptagon (2008).

Fowler (2006), destaca que a integração contínua obteve bons resultados durante um projeto *offshore* entre Índia e outros países. Projetos *offshore* são consideravelmente mais complexos do que projetos *onshore*, logo esta prática também pode ser aplicada no contexto *onshore insourcing.* A integração se favorece da característica iterativa e incremental inerentes a todas as metodologias ágeis estudadas. A utilização de ferramentas de controle de integração e testes, assim como o uso de *wikis* ou outro tipo de recurso, são importantes para o desenvolvimento distribuído (FOWLER, 2006; URDANGARIN, 2008). A integração permite avaliar com maior clareza o andamento do projeto, encontrar possíveis falhas com maior rapidez e possibilita que o cliente se mantenha satisfeito durante todo o ciclo de desenvolvimento do sistema.

Constata-se que outras práticas utilizadas na metodologia XP são compatíveis com o DDS. Dentre elas pode-se destacar a prática do cliente sempre disponível, que deve sofrer algumas adequações, por exemplo, o cliente poderá estar à disposição em uma das unidades, sendo o gerente responsável por deixar o cliente a par do andamento do projeto, e por repassar informações para as outras unidades.

A prática da propriedade coletiva do código é interessante, pois traz muita agilidade para o DDS, onde se tem grandes barreiras de comunicação. A propriedade individual (ou a propriedade da dupla que codificou), poderia trazer grandes problemas na hora de fazer alterações, ou para localizar a pessoa que desenvolveu um determinado artefato de software. No *outsourcing* se lida com diferentes empresas, onde a propriedade coletiva do código deve ser analisada de forma mais cuidadosa.

A padronização do código vai totalmente ao encontro das necessidades do desenvolvimento distribuído *onshore insourcing*, pois o código acaba se transformando em uma forma de comunicação. Todas as unidades terão facilidade em entender o que foi desenvolvido, o código fica claro e reduz a necessidade do uso de meios de comunicação para esclarecer possíveis dúvidas.

De acordo com Urdangarin (2008), uma das deficiências encontradas durante aplicações do XP no DDS, se deu pela falta de uma pessoa sempre presente, responsável por esclarecer duvidas e questões das equipes de

desenvolvimento. Conforme Marchesi et al. (2002) e Ebert et al. (2001), a atribuição de uma pessoa responsável por esta atividade, pode suprir estas dificuldades. É importante que alguém com experiência e bom conhecimento da metodologia, um líder de equipe, esteja sempre à disposição para auxiliar os desenvolvedores em possíveis dúvidas. De acordo com as metodologias estudadas no capítulo 3, esta deficiência apontada pela metodologia XP, pode ser perfeitamente suprida pela adoção de características da metodologia Scrum, que dentre todas elas, é a mais direcionada para o gerenciamento de atividades, além de fomentar do espírito de equipe. A figura do *Scrum Master*, aliado as práticas do XP, pode trazer bons resultados, pois ele auxilia a equipe a manter o foco no trabalho, participando das reuniões diárias com a equipe, e ajudando a manter os envolvidos com a visão do projeto como um todo. O Scrum, assim como as outras metodologias estudadas, não cobre todas as atividades relacionadas ao desenvolvimento distribuído, porém, alguns aspectos podem ser utilizados.

A metodologia UP é composta por um *framework* customizável, logo não é possível dizer que ele é totalmente incompatível com as outras metodologias (LARMAN, 2003). O UP possui diferenças com relação ao XP na etapa de modelagem, que é realizada através de diagramas de seqüência UML, já o XP utiliza as *user stories*, que são cartões respondidos pelo cliente, formando o conjunto de requisitos do sistema. Assim como o UP o FDD também utiliza UML, que de acordo com Highsmith (2002), após serem reconhecidos os requisitos, eles são listados, definidos e documentados através de *user stories*, onde é sugerido a construção de diagramas de classes em UML e diagramas de seqüência UML, para auxiliar a compreensão do projeto.

O uso de UML é uma boa opção para o desenvolvimento distribuído *insourcing*, pois se trata de uma padronização onde todas as unidades poderão compreender as diversas partes do sistema com maior facilidade.

No UP, antes de uma iteração de duas semanas, é comum utilizar a metade de um dia para discutir sobre idéias do projeto ou utilizar algum tipo de modelagem visual para a equipe, já no XP um *feedback* de 15 ou 20 minutos é o suficiente antes da equipe começar a codificar (LARMAN, 2003). Em projetos distribuídos é necessário gastar um pouco mais de tempo para as equipes ficarem por dentro da fase que irá começar, pois o desenvolvimento envolverá várias equipes e poderão

surgir duvidas no decorrer da iteração. Utilizar um pouco mais de tempo no início da iteração, pode fazer com que a equipe produza mais, e conclua o projeto em menos tempo.

Durante o ciclo de vida da metodologia Scrum, ocorrem dois tipos de reuniões, uma é a de planejamento, que ocorre antes da cada iteração e é realizada junto com o cliente, e a outra é de retrospectiva, que ocorre a cada 24 horas. Durante cada sprint ocorrem reuniões de 15 a 30 minutos para manter todos os envolvidos informados sobre o progresso e as dificuldades encontradas (SCHWABER, 2004). Essas reuniões são essenciais para um bom gerenciamento do projeto.

No desenvolvimento distribuído deve ser usado ferramentas para suprir a dispersão geográfica. É necessário ter bons meios de comunicação, que possam viabilizar a utilização, como exemplo, o uso ferramentas de vídeo-conferência para reuniões entre os lideres de cada equipe e possíveis reuniões entre os membros das equipes de desenvolvimento. Além de ocorrerem reuniões entre as equipes dispersas, as reuniões das equipes co-localizadas devem ocorrer diariamente (a cada *sprint* de 24 horas, segundo a metodologia Scrum) com a presença do líder de equipe. Cabe lembrar que a presença do cliente é uma característica básica das metodologias ágeis, e este deverá estar disponível em uma das unidades do projeto. Adaptando para o desenvolvimento *onshore insourcing*, o cliente deverá estar presente nas reuniões entre os lideres de equipe, e nas reuniões entre a equipe de desenvolvimento da unidade onde ele estará presente, estando sempre à disposição para sanar dúvidas de qualquer *stakeholder* envolvido. Prikladnicki (2003) relata que "todos os membros do time tinham a liberdade de entrar em contato com o cliente para resolver problemas de interpretação gerados por requisitos mal escritos ou ambíguos” em sua análise sobre projetos distribuídos. O constante *feedback* com o cliente, em times distribuídos, traz auto-estima para a equipe. A disponibilidade dos programadores entrarem em contado com o cliente para tirar duvidas, gera um sentimento de comprometimento com o projeto (PRIKLADNICKI, 2003). Neste projeto, apontado por Prikladnicki (2003), envolvendo países diferentes, o cliente foi representado pela equipe central de desenvolvimento.

Todas as metodologias ágeis estudadas possuem características em comum, pode-se notar que existem semelhanças entre elas em cada fase de

desenvolvimento. A DSDM possui muitas características em comum com o UP, é uma metodologia dinâmica onde o problema vai sendo resolvido de forma iterativa, convergindo para um resultado. A DSDM fornece um framework independente de ferramentas, onde o utilizador pode adaptá-lo conforme as suas necessidades. O XP também é baseado em 4 fases, sendo um pouco diferenciado principalmente por suas práticas, onde o sucesso de sua utilização depende de seus utilizadores.

A metodologia FDD também pode ser comparada com as outras metodologias, tendo muitas características em comum. As 3 fases iniciais são compatíveis com as fases de concepção e elaboração do UP, assim como são equiparáveis com a etapa de construção do *Product Blacklog* da metodologia Scrum. As duas fases subseqüentes se equiparam principalmente a fase de construção da metodologia UP e aos *sprints* de desenvolvimento do Scrum.

Características do Scrum podem ser aplicadas também em contexto distribuído de desenvolvimento. Fowler (2006), afirma que as metodologias ágeis primam pelos interesses do cliente, e podem ser aplicadas ao desenvolvimento com equipes dispersas. A Fig. 17 demonstra uma comparação entre as fases das metodologias ágeis estudadas.

UP	***Scrum***	***FDD***	***DSDM***	***XP***
- Concepção	- *Product Blacklog*	- Desenvolver um Modelo Abrangente - Construir Lista de *Funcionalidades*	- Estudo da Viabilidade - Estudo do Negócio	- Planejamento
- Elaboração	- *Sprint Planning* - *Sprint Blacklog*	- Construir Lista de *Funcionalidades* - Planejar por *Funcionalidade* - Detalhar por *Funcionalidade*	- Iteração do Modelo Funcional	- Projeto
- Construção	- *Sprint* de 2-4 semanas - *Sprint* de 24 horas	- Detalhar por *Funcionalidade* - Construir por *Funcionalidade*	- Iteração de Projeto e Construção	- Codificação
- Transição	- Novas Funcionalidades - Retrospectiva	- Produto	- Implementação	- Testes

Figura 17 - Comparação entre as fases das metodologias

Outro fator importante quanto a escolha de uma metodologia de desenvolvimento, é a análise com relação a sua agilidade. No desenvolvimento distribuído, também é necessário a busca por agilidade, porém, devem ser levadas em consideração as necessidades e problemas encontrados neste contexto de desenvolvimento. Normalmente, a avaliação das metodologias com relação a sua agilidade, está imbuída de subjetividade e preconceitos pessoais, tornando difícil uma avaliação de forma precisa das mesmas. A Tab. 1 demonstra um comparativo desenvolvido por Highsmith (2002) que avalia as metodologias de acordo com uma série de características. São dadas notas de 1 a 5, quanto maior a nota mais ágil é a metodologia.

Tabela 1 - Comparação da agilidade das metodologias
Fonte: adaptado de Highsmith (2002)

Agilidade	XP	Scrum	UP	DSDM	FDD
Simplicidade	5	5	2	3	5
Resultados	5	5	3	3	5
Colaboração	5	5	3	4	4
Adaptabilidade	3	4	3	3	3
Técnica	4	3	4	4	4
Prática	5	4	3	4	3
Média	**4.5**	**4.5**	**3**	**3.5**	**4**

Através desta comparação verifica-se que XP e Scrum são as metodologias mais ágeis, logo após vem à metodologia FDD. Três destas metodologias primam pela simplicidade, obtendo nota máxima neste quesito. UP e DSDM são metodologias ágeis bastante parecidas, ficando com notas 3 e 3.5. A metodologia UP é a menos ágil dentre as 5 analisadas.

Para a utilização de metodologias ágeis no desenvolvimento distribuído, também é necessário avaliar suas principais características. Sabe-se que para o sucesso do desenvolvimento distribuído é importante que exista um processo comum a todas as equipes e, neste contexto, as metodologias ágeis demonstram vantagens com relação às metodologias ditas tradicionais. No desenvolvimento distribuído, há uma maior necessidade de coordenação do trabalho, e busca de formas para controlar o andamento do projeto. A metodologia Scrum possui

características de gerência de projeto, dentre as estudadas, é a mais indicada para este tipo de projeto. Scrum já foi aplicada com sucesso em desenvolvimento de software livre, com características de desenvolvimento distribuído (SOARES et al., 2007), necessitando apenas de pequenos ajustes com relação à comunicação e presença do cliente no local.

No desenvolvimento *onshore* não há problemas relevantes com diferenças culturais, diminuindo o maior problema do desenvolvimento distribuído, que é a comunicação. Com poucos problemas relativos a idioma ou diferentes culturas, basta ter meios de comunicação que viabilizem a comunicação entre os *stakeholders*, incentivando os envolvidos a utilizarem estes meios, fomentando o espírito de equipe, que são características marcantes das metodologias ágeis.

Com relação a ferramentas de apoio, Fowler (2006) e Urdangarin (2008), destacam o uso de *wikis* em seus projetos distribuídos de software, que foram de fundamental importância para o sucesso. *Wiki* é uma ferramenta utilizada para produzir informação de forma colaborativa. Através de *wikis* se obtém agilidade e a participação de várias pessoas colaborativamente, gera conteúdo com a ajuda de vários usuários utilizando o mesmo documento (BRAGAGLIA, 2006). Urdangarin (2008) descreve que o uso desta ferramenta foi o principal meio de colaboração entre as equipes dispersas, com ela foi possível trocar informações entre todas as equipes, possibilitando que os requisitos, informações de arquitetura e *design*, desenvolvimento e testes, ficassem todos em um mesmo local, com fácil acesso para todos os *stakeholders*. Urdangarin (2008), ainda relata que através de *wiki*, é possível associar a unidade de desenvolvimento às funcionalidades desenvolvidas (ou em desenvolvimento), é possível rastrear os requisitos.

Ferramentas deste tipo funcionam bem em projetos distribuídos, pois são simples de usar, e podem ser acessadas através de browsers de internet em qualquer local com acesso a internet. Segundo Fowler (2006), qualquer tipo de informação comum do projeto deve ser disponibilizado na *wiki*: funcionalidades, diagramas UML, instruções para integração, ou qualquer informação que se julgue importante para o sucesso do desenvolvimento. Equipes locais também podem utilizar wiki para publicar seus diagramas UML, que podem servir de documentação para a arquitetura do software, e também para publicar os papéis de cada *stakeholder* envolvido no processo (URDANGARIN, 2008).

6 SUGESTÃO DE MELHORES PRÁTICAS ÁGEIS PARA APLICAÇÃO EM ONSHORE INSOURCING

A partir da análise das melhores práticas das metodologias ágeis estudadas no capítulo 3, será apontada uma sugestão de práticas para serem aplicadas no desenvolvimento distribuído *onshore insourcing.* Para começar, serão destacados os principais papéis que devem estar presentes no desenvolvimento.

Os principais papéis envolvidos no projeto são:

- Cliente -: pode estar representado em uma das unidades onde será desenvolvido o projeto.
- Equipes de desenvolvedores - em cada unidade que irá participar do projeto deverá ser formada, ou selecionada uma equipe para o desenvolvimento.
- Gerente de projeto - pessoa responsável por todo o projeto.
- Líder de equipe - pessoa que irá liderar uma equipe, cada unidade deverá ter a figura do líder de equipe.

Em cada projeto poderão ser criados outros papéis, nesta sugestão somente os funções básicas serão detalhadas. Os diferentes papéis envolvidos estão diretamente ligados com as figuras do *Scrum Owner* (cliente), *Scrum Team* (equipe desenvolvedores), e *Scrum Master* (gerente de projeto) utilizados na metodologia Scrum. Para figura do Líder de equipe, será utilizado o nome *Scrum Coach,* (assim como no jogo de rúgbi, no projeto ágil distribuído, deve haver um líder, que auxilie a equipe e possa suprir possíveis dúvidas). Cada unidade envolvida no desenvolvimento distribuído terá que ter, obrigatoriamente, as figuras do *Scrum Team* e do *Scrum Coach*, podendo o *Scrum Master* realizar o papel do *Scrum Coach* em uma das unidades.

De acordo com o estudo realizado, o *workflow* do UP apresentou-se como uma boa opção para servir de base no desenvolvimento ágil. O principal motivo para a escolha do UP foi pelo fato de ser instanciavel para vários tipos de projetos e empresas, conforme as necessidades do projeto. Souza (2007) constatou que o UP é a metodologia mais indicada para a aplicação em grandes projetos distribuídos, porém sua aplicação de forma integral pode ser demasiadamente burocrática. Tem-se então a necessidade de fazer algumas modificações no seu *workflow* com a intenção de torná-lo mais ágil. Como foi observado anteriormente, as metodologias ágeis estudadas possuem muitas similaridades entre elas, e suas etapas de desenvolvimento se assemelham com as fases de concepção, elaboração, construção e transição definidas na metodologia UP. As iterações entre as etapas de desenvolvimento ocorrem conforme a Fig. 18

Figura 18 - Ciclo de vida de software baseado em ciclos iterativos
Fonte: Wazlawick (2004)

O processo de desenvolvimento inicia com a etapa de concepção, onde o cliente toma a decisão de desenvolver o projeto, entrando em contato com a empresa de desenvolvimento. Nesta etapa é realizada uma análise do problema e é verificado se o sistema será desenvolvido a partir do zero, ou se é um sistema já existente que precisa de novas funcionalidades.

Para a fase de concepção são analisadas as características gerais do sistema e uma das equipes é selecionada para realizar a modelagem do sistema. Esta equipe é dividida em pequenos grupos que analisam os dados iniciais, como requisitos, modelos de negócio, modelos de objetos e modelos de dados. Torna-se necessário analisar a viabilidade de desenvolvimento *onshore insourcing* do sistema. Também devem ser realizados pequenos *workshops* entre todos os envolvidos na

fase, demonstrando e refinando as funcionalidades do sistema. O gerente de projeto e o cliente também precisam estar presentes nesta etapa.

Após a etapa de concepção, surge a etapa de elaboração que é caracterizada, pela definição das equipes que estarão envolvidas no projeto, com os seus respectivos líderes, que também devem participar da mesma. Todas as funcionalidades são separadas de acordo com a prioridade, devendo ser considerado a quantidade e as características das equipes que estarão envolvidas no projeto distribuído. Deve ser criada a estrutura de comunicação para as equipes dispersas tais como, wikis, grupos de e-mail e mecanismos para vídeo-conferência.

A lista de funcionalidades deve ser avaliada pelo cliente. Nesta etapa devem estar presentes as figuras do cliente, gerente de projeto, o líder de cada equipe envolvida, além da equipe de desenvolvedores que foi selecionada inicialmente, na fase de concepção.

Assim como é praticado na metodologia Scrum, em todas as iterações e a cada 24 horas, devem ocorrer reuniões rápidas com os integrantes da equipe. As reuniões realizadas a cada 24 horas devem contar com a presença da equipe local (equipe da unidade de desenvolvimento). As reuniões que ocorrem a cada iteração , devem contar com a presença do gerente de projeto, cliente e líderes de cada equipe, onde lhes são apresentados *workshops* sobre o andamento do projeto (que já foi feito e o que ainda falta fazer). A cada sprint podem ser feitas alterações nas funcionalidades do projeto.

Após serem selecionados todos os integrantes das equipes, deve ser fomentada a integração entre todos os *stakeholders* envolvidos, para que haja uma boa comunicação durante a etapa de construção, bem como nas iterações restantes.

Segundo Damian et al. (2006), é importante manter a visibilidade do processo entre todo o time de desenvolvimento distribuído. Podem ocorrer problemas quando os líderes não se encontram presentes para acompanhar todo o processo de desenvolvimento. A integração de todos os integrantes no desenvolvimento é de fundamental importância, pois ajuda a aumentar os canais de comunicação durante o desenvolvimento. Reuniões face-a-face entre os líderes de equipes, gerente e cliente devem ser feitas no início do projeto na forma de

workshop. Se a distância entre as unidades de desenvolvimento não for muito significativa, as equipes de desenvolvimento também devem se fazer presentes neste *workshop* inicial de desenvolvimento. Caso o custo de deslocamento entre os integrantes das equipes seja muito alto para as proporções do projeto, esta integração pode realizada por vídeo-conferencia ou outro meio que se considerar adequado para tal. O principal motivo para a integração de todos os *stakeholders* é que todos se conheçam pessoalmente antes de iniciado o projeto distribuído, além de conhecerem o sistema que será desenvolvido. O fato de se conhecerem pessoalmente no início, facilita e incentiva a comunicação através de outros meios durante o desenvolvimento. Todos os envolvidos no projeto (inclusive o cliente) devem estar sempre à disposição para eliminar qualquer tipo de dúvida relativa ao projeto.

Prosseguindo na fase de elaboração, as funcionalidades devem ser colocadas em ordem para desenvolvimento, e é feito o planejamento das atividades das equipes. São atribuídas atividades aos líderes das equipes, e estes atribuem às atividades as suas equipes de desenvolvimento. Nesta fase, cada equipe dispersa já tem conhecimento de quais atividades irá desenvolver. São criados os diagramas das funcionalidades, e feito o prefácio das classes que serão desenvolvidas. Os líderes de equipes, devem sempre fazer o *feedback* ao gerente de projeto, através da realização das reuniões diárias.

Na fase da construção cada equipe realiza a implementação das funcionalidades distribuídas para cada líder de equipe. Nesta etapa é gerado artefatos de software com valor para o cliente. Deve ocorrer constante integração de software entre as equipes distribuídas e conseqüente atualização de versões do produto.

Na etapa de transição, o produto é entregue ao cliente, onde os usuários finais são treinados e podem sugerir novas funcionalidades a serem implementadas, e onde defeitos são concertados. Nesta etapa também é feita a documentação e o manual de utilização do produto. Ao final a equipe deve se reunir e realizar o *feedback* do projeto, analisado e aperfeiçoando o seu PDS.

Cada empresa possui características próprias, com diferentes necessidades e diferentes número de equipes e desenvolvedores. O número mínimo de

integrantes sugerido para cada equipe deve ser de duas pessoas, sendo um deles o líder. Não é sugerido um número máximo de integrantes, depende do tamanho e da complexidade de cada projeto. Um número muito grande de integrantes por equipe pode deixar o processo mais pesado. Cada empresa deve realizar um *feedback* do desenvolvimento, realizando uma avaliação também com relação ao sucesso referente ao tamanho das equipes de desenvolvimento.

A Fig. 19 realiza a demonstração das etapas a serem seguidas, com a utilização de práticas de metodologias ágeis em DDS *onshore insourcing*. Como características importantes têm-se:

- O cliente deve estar sempre à disposição de qualquer *stakeholder* do projeto.
- O gerente do projeto pode realizar o papel de um líder de equipe.
- A equipe local é a matriz, ou o local onde o cliente é representado.

fases	***papéis envolvidos***	***principais características***
- concepção	- cliente - gerente - equipe local	- análise do problema - análise das funcionalidades iniciais - realização de *workshop* - estudo de viabilidade - analisar necessidade de DDS - escolha de líderes de equipes
- elaboração	- cliente - gerente - equipe local - líderes de equipes	- formação das equipes - criar interface de comunicação entre unidades - separação das funcionalidades - avaliação das funcionalidades - geração de protótipo
- construção	- cliente - gerente - líderes de equipes - todas as equipes	- realização de *workshop* - criar modelos de objetos - implementação de classe e métodos - integração e atualização da versão do software - inspeção de código e testes de unidade - primeira versão do sistema - reuniões com equipes locais e com os líderes a cada 24 horas
- transição	- cliente - gerente - líderes de equipes - todas as equipes	- entrega do produto ao cliente - correção de defeitos - inclusão de novas funcionalidades - treinamento - produção de documentação para o usuário - *feedback*

Figura 19 - Ciclo de desenvolvimento ágil para *Onshore Insourcing*

6.1 Modelagem das Fases de Desenvolvimento

Após serem definidos os artefatos pertencentes a cada fase de desenvolvimento, juntamente com e os papéis básicos para o desenvolvimento *onshore insourcing,* será feito um detalhamento de cada uma das fases. Como foi observado no capítulo 4, é importante ter um modelo de processo com fases bem definidas, adaptando-se as necessidades do ambiente onde será aplicado. Baseado na análise dos processos, os papéis básicos são: **o gerente**, que é responsável pela liderança e coordenação geral do projeto; **os líderes de equipes**, que lideram cada unidade de desenvolvimento e, juntamente com o gerente e com o cliente, realizam a identificação dos requisitos e a modelagem dos casos de uso, definindo as funcionalidades do sistema e as equipes de desenvolvedores. Estas equipes realizam do desenvolvimento, atuando principalmente na fase de construção do sistema, cada unidade terá um grupo de desenvolvedores; **o cliente** é o destinatário do produto, que também estará presente, à disposição das equipes, em todas as fases de desenvolvimento para suprir possíveis dúvidas.

As etapas abrangidas neste trabalho são as etapas de concepção, elaboração, construção e transição, será feita a modelagem destas etapas, juntamente com a modelo de seleção de equipe, e o modelo de avaliação e *feedback.*

A ferramenta JUDE UML *Community*[7], versão 5.4, foi utilizada para realizar a modelagem das fases a papéis envolvidos no desenvolvimento. A Fig. 20 demonstra os elementos utilizados na modelagem.

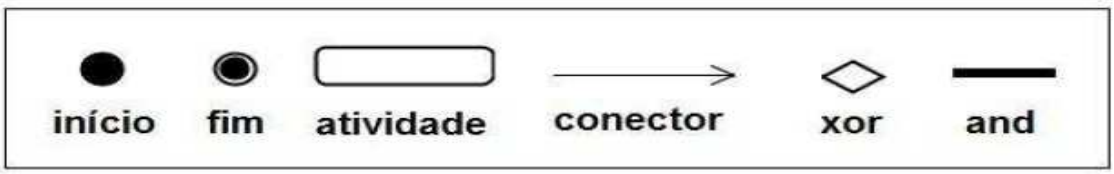

Figura 20 - Elementos utilizados na modelagem do sistema

7 JUDE é um software para modelagem, livre e multiplataforma, pode ser obtido em: http://jude.change-vision.com.

6.1.1 Fase de Concepção

Esta fase é caracterizada pelo início do desenvolvimento, onde o sistema deixa de ser apenas uma idéia, e começa a ser planejado pela empresa de desenvolvimento, com a ajuda do cliente. Esta fase conta com a presença do cliente, gerente de projeto da empresa de desenvolvimento, e uma equipe local de desenvolvimento. Supõe-se que a empresa já possua pelo menos uma equipe de desenvolvimento, caso ainda não possua, deverá formar esta equipe para auxiliar na concepção do produto.

A modelagem do sistema foi feita de forma resumida, para uma melhor compreensão do leitor, os detalhes são explicados textualmente, e constam nas práticas sugeridas para ambiente *onshore insourcing*. Leva-se em conta, ainda, que cada empresa de desenvolvimento possui peculiaridades, e como não existe um processo ideal, ela deve adaptar o processo às suas características, sempre em busca de mais agilidade no desenvolvimento.

Na fase de concepção é criada uma proposta de projeto, onde o gerente de projeto avalia as funcionalidades (requisitos) demonstradas pelo cliente, é realizada uma reunião (*workshop*) com os atores envolvidos, resultando em uma proposta. Ao final, esta proposta é avaliada pelo cliente. Caso seja aprovada, é verificado se existe a necessidade de desenvolvimento distribuído, onde parte-se para a escolha dos líderes que estarão presentes em cada unidade de desenvolvimento. Se não for viável o desenvolvimento distribuído, o sistema pode ser desenvolvido *on-site*.

Deve ser escolhido um líder para a equipe co-localizada, podendo o gerente, também, desempenhar este papel. No caso de a proposta não ser aprovada, pode ser realizado um novo ciclo, ou o projeto é abandonado. A Fig. 21 ilustra a fase de concepção do projeto.

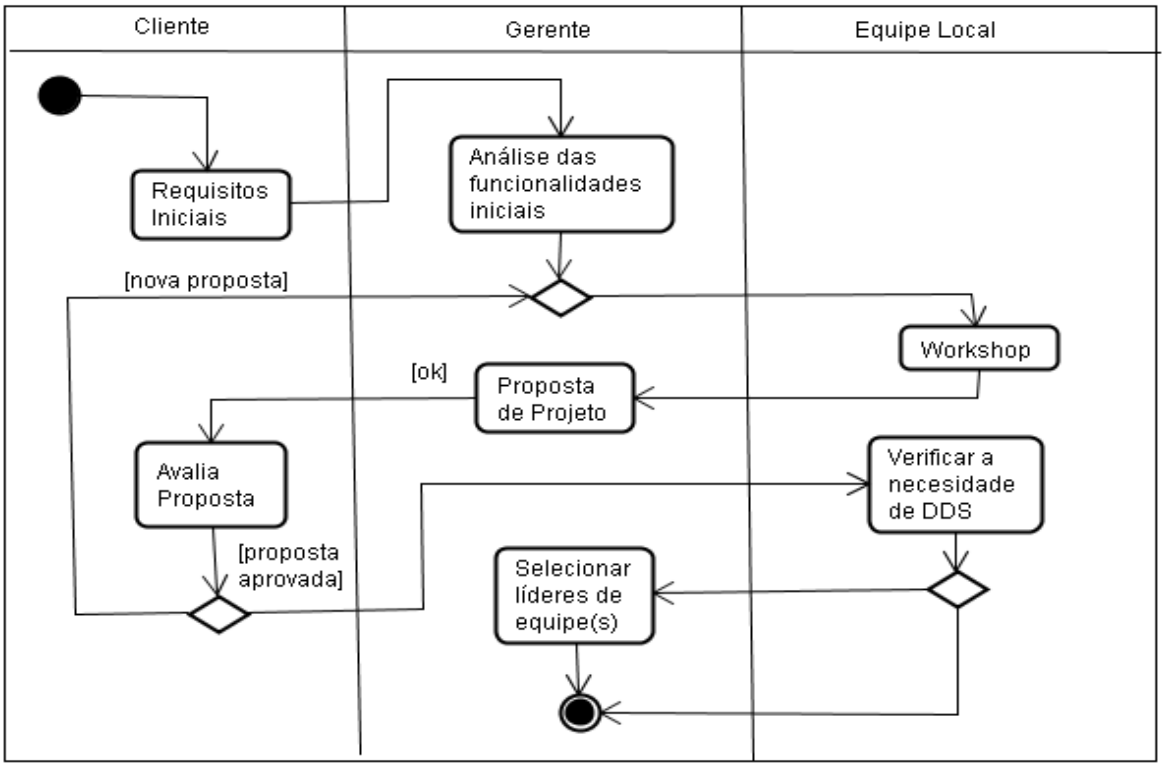

Figura 21 - Fase de concepção

6.1.2 Fase de Elaboração

Na fase de elaboração as funcionalidades iniciais do sistema são separadas de acordo com a prioridade e tamanho. Esta divisão também será utilizada na divisão de trabalho entre as equipes dispersas de desenvolvimento, ao final da fase. Após a separação das funcionalidades, a equipe local desenvolve um protótipo do sistema, para avaliação do cliente. O cliente verifica se o protótipo vai ao encontro de seus interesses, podendo sugerir modificações. Com o protótipo aprovado, parte-se para a formação das equipes de desenvolvimento, realizada através do gerente. Ao final, as funcionalidades são divididas entre os líderes de projeto de cada unidade, e estes terão o papel de atribuir responsabilidades à suas equipes de desenvolvimento. A Fig. 22 demonstra a modelagem da fase de elaboração.

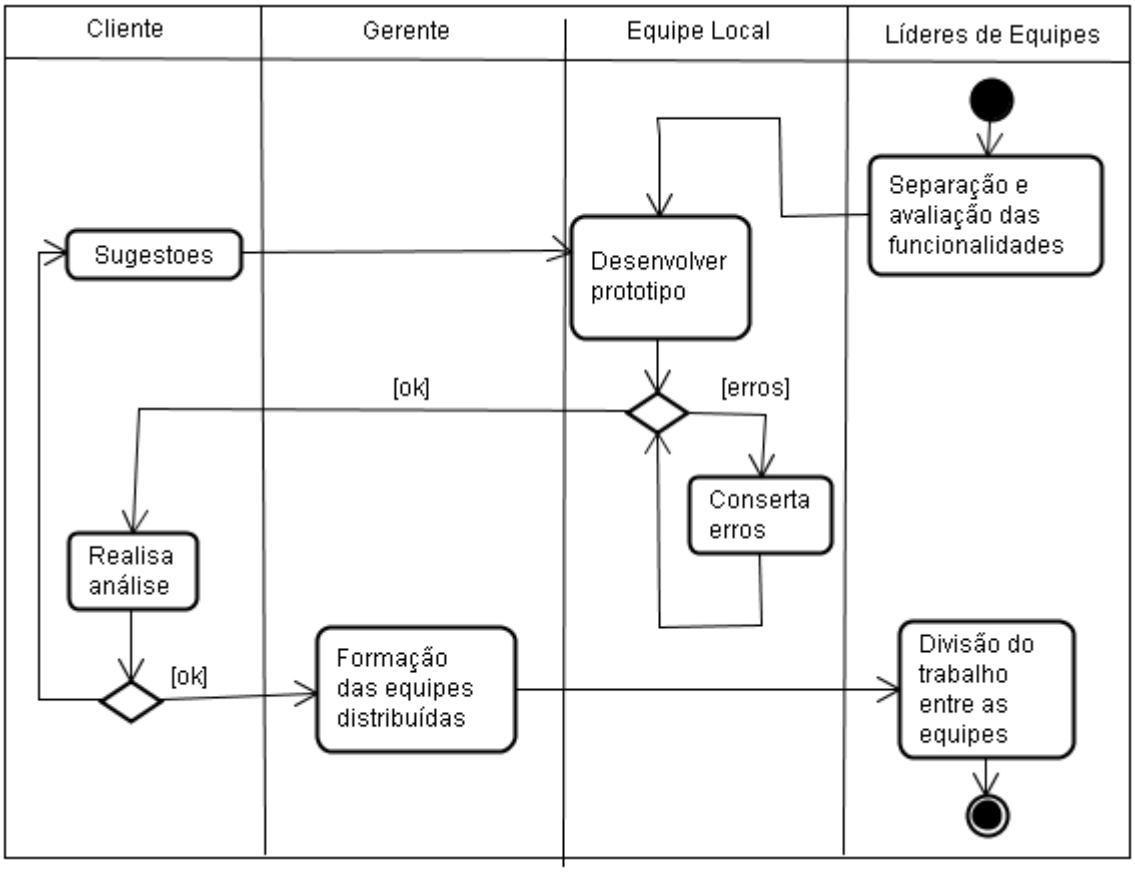

Figura 22 - Fase de elaboração

A seleção dos integrantes das equipes de desenvolvimento é realizada pelo gerente, ou a outra pessoa, conforme a realidade de cada empresa. O modelo ilustrado pela Fig. 23 também pode ser utilizado para a escolha de líderes de equipes, onde são atribuídos os papéis conforme as escolhas através do banco de dados interno da empresa: através da alocação de funcionários existentes, ou através da inclusão de novos funcionários.

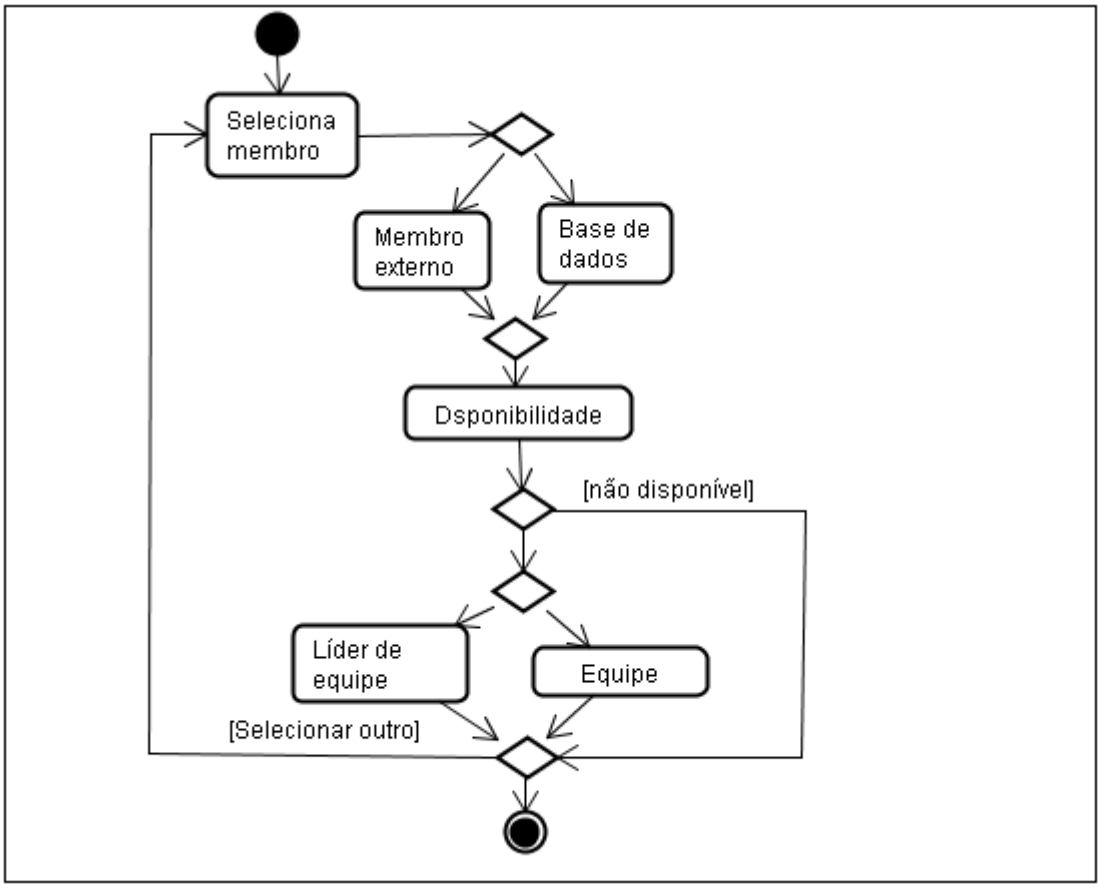

Figura 23 - Seleção da equipe
Fonte: adaptado de Teixeira (2007)

6.1.3 Fase de Construção

Nesta fase é realizado um planejamento inicial, feito através de um *workshop* com todos os *stakeholders* envolvidos. Se a distância entre as unidades for muito grande, o *workshop* pode ser feito através de vídeo-conferência. Todos devem estar conscientes de todas as funcionalidades do sistema, e deve ser incentivada a comunicação entre todas as unidades. As reuniões diárias (*feedback*) são importantes até mesmo na primeira iteração, após o *workshop,* pois incentivam a comunicação dentro de cada unidade de desenvolvimento. O desenvolvimento é feito seguindo as práticas ágeis de desenvolvimento, vistas nos capítulo 3. Ao final de cada dia é realizada a integração dos artefatos produzidos, gerando um produto de maior valor, sendo visível ao cliente a cada integração. Após a integração é verificada a necessidade de um novo ciclo, através de inspeção no código, onde é verificado o andamento do projeto, e o software é testado. A cada iteração, realizada entre duas e quatro semanas, deve ser realizada uma reunião presencial entre os

líderes, gerente e o cliente. Após a liberação da primeira versão do sistema, pode-se voltar para a etapa de elaboração, onde o cliente pode sugerir a alteração de funcionalidades, ou pode-se partir para a etapa de transição do sistema, onde o mesmo é disponibilizado para uso do cliente. A Fig. 24 ilustra o fluxo de desenvolvimento da fase de construção.

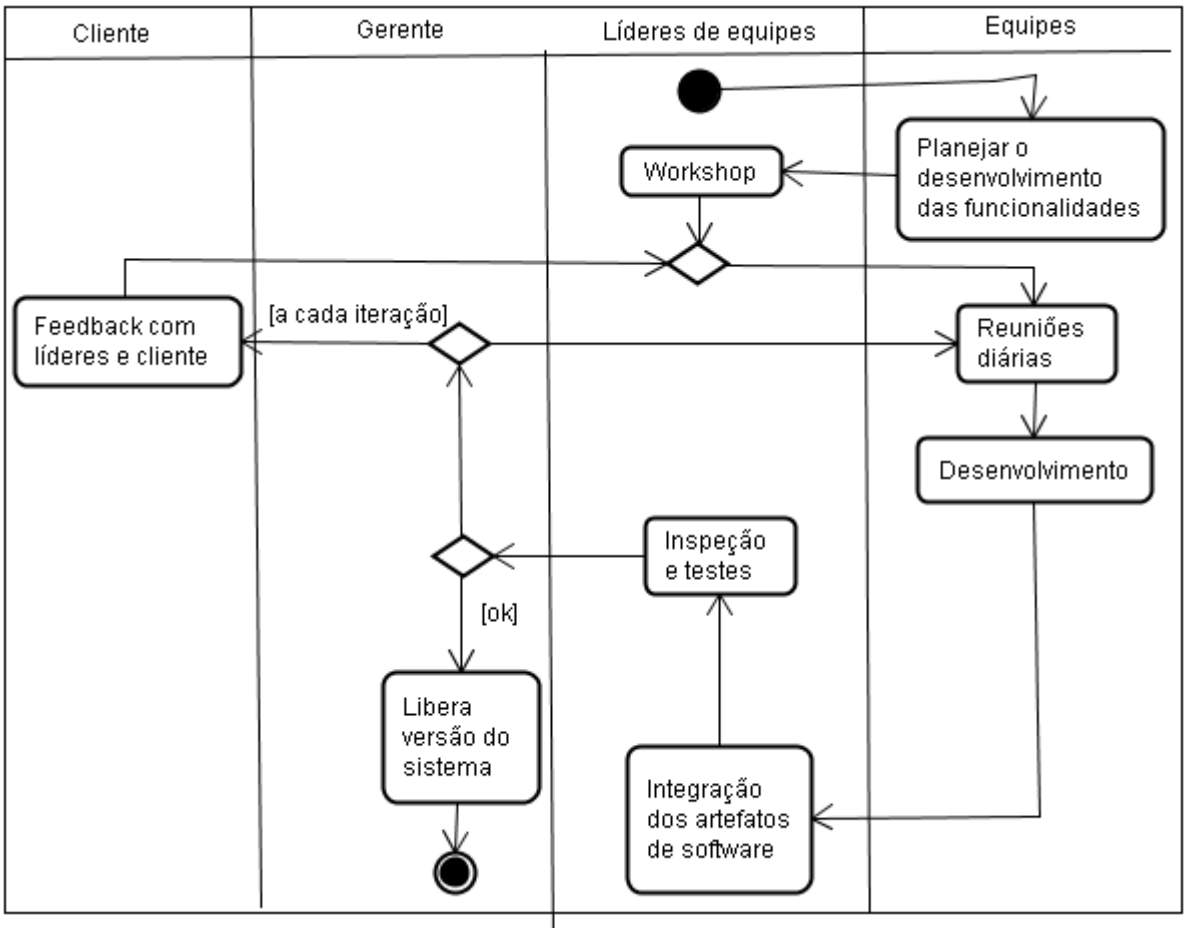

Figura 24 - Fase de construção

6.1.4 Fase de Transição

Na fase de transição o produto desenvolvido é entregue ao cliente. O material de suporte para implantação, como manuais e documentação, é disponibilizado para o cliente. São realizados testes com o usuário final, onde podem ser sugeridas novas mudanças no sistema. Nesta fase é feita a instalação do produto, distribuição e treinamento no local onde será utilizado. Após é realizado um *feedback* com todos os *stakeholders* envolvidos no processo de desenvolvimento. A Fig. 25 mostra o funcionamento da fase de construção.

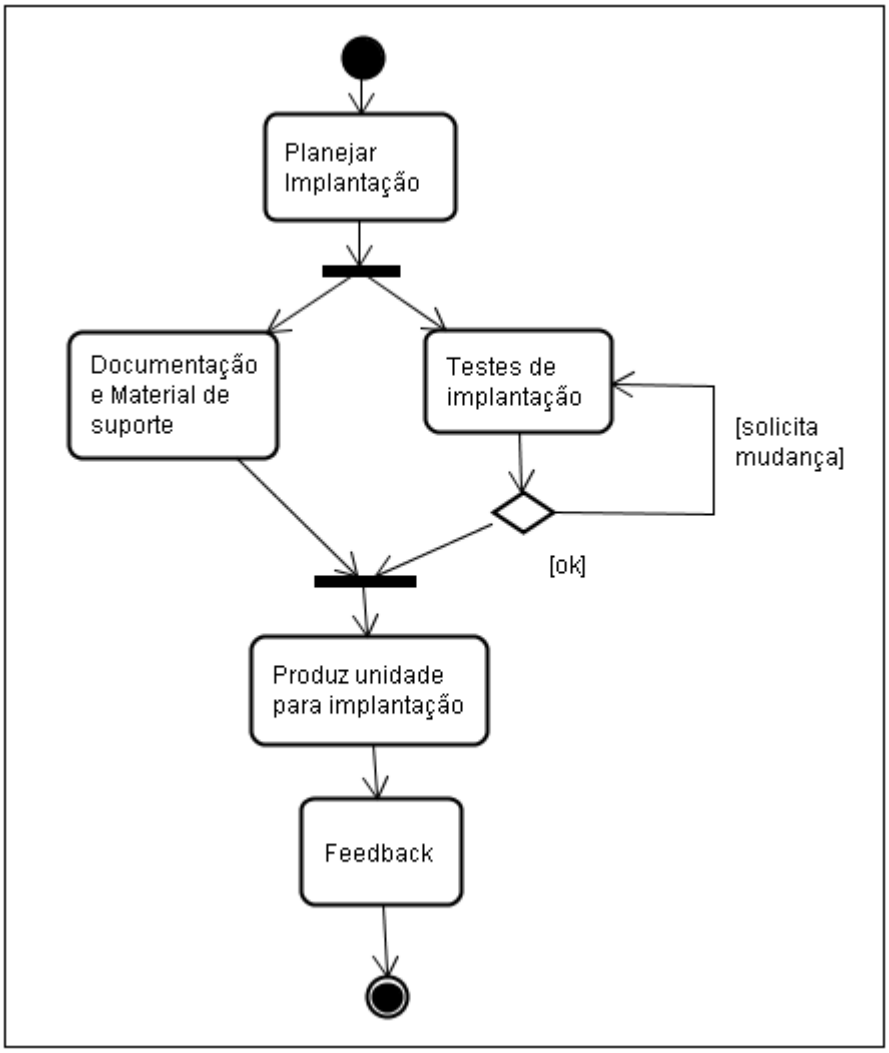

Figura 25 - Fase de transição
Fonte: adaptado de Reis e Souza (2008).

O modelo de avaliação e *feedback* é muito importante em qualquer processo de desenvolvimento, pois permite que sejam enumeradas as características que podem ser melhoradas para um futuro ciclo de desenvolvimento. De acordo com Prikladnicki (2003), esta avaliação se torna mais importante por se tratar de equipes fisicamente dispersas, onde se deve realizar uma análise crítica das estratégias utilizadas e do processo de desenvolvimento utilizado. Deve ser feita uma avaliação com todos os envolvidos no processo, realizando uma avaliação no produto desenvolvido, onde é feito um balanço entre os prazos e custos de desenvolvimento, estratégias adotadas e sugestões para o desenvolvimento de novos projetos. Segundo Prkladnicki (2003), a conclusão de um projeto dá-se pela avaliação final, e este modelo deve ser aplicado sempre ao final dos projetos, fornecendo retro alimentação para novos projetos. As informações importantes são consolidadas e

incluídas no repositório de informações. A Fig. 26 demonstra um modelo com as etapas de avaliação e *feedback*.

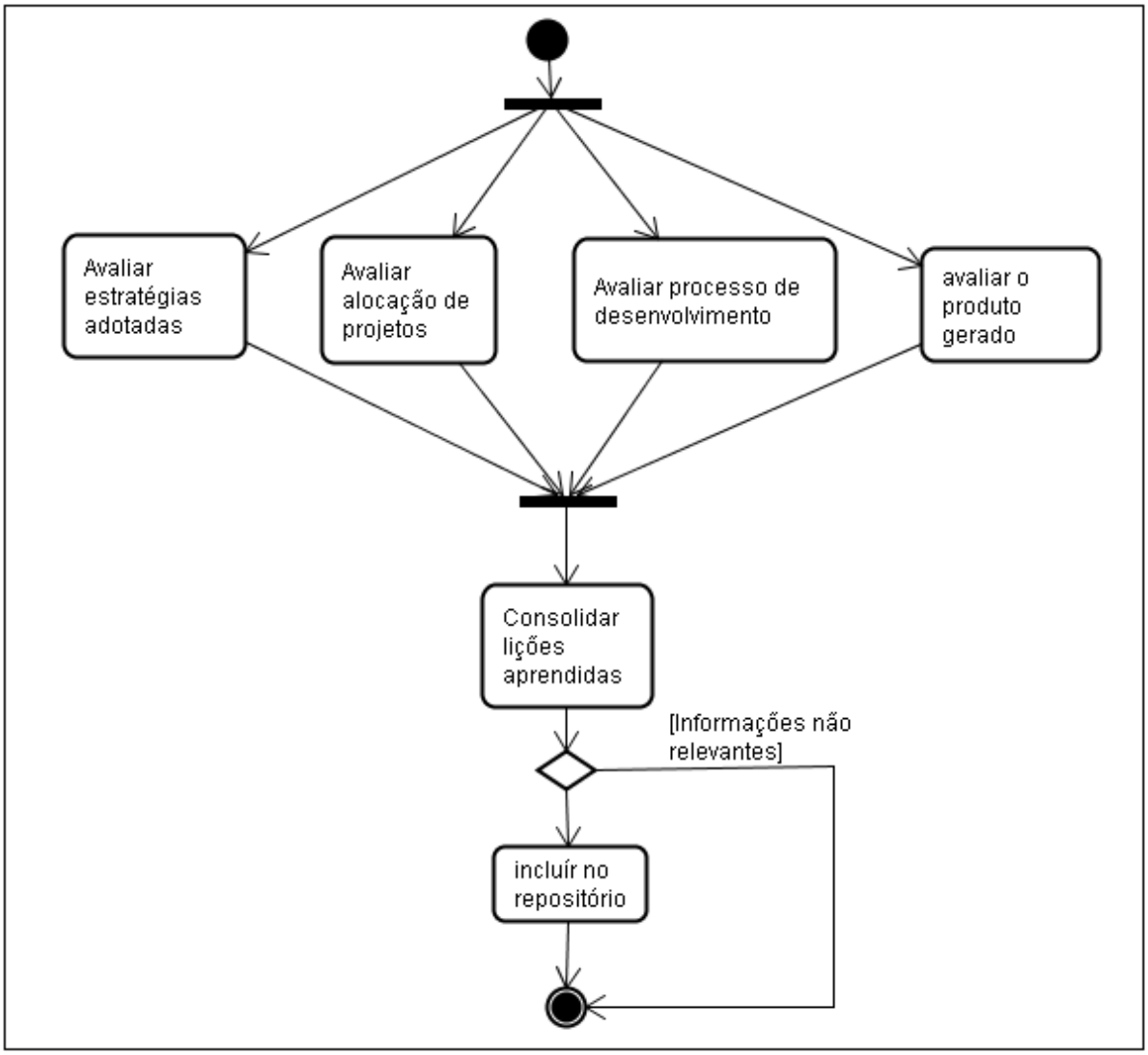

Figura 26 - Processo de avaliação e *feedback*
Fonte: Prikladnicki (2003).

6.2 Comparação entre as metodologias ágeis estudadas e o modelo proposto

A sugestão de práticas para serem utilizadas em ambiente distribuído *onshore insourcing,* tem por objetivo auxiliar a suprir dificuldades durante a escolha e aplicação de processo para desenvolvimento de software. Atualmente, conta-se com vários modelos de processos disponíveis, que podem ser processos mais pesados e burocráticos, ou metodologias mais leves e ágeis. As metodologias ágeis foram concebidas para melhorar o ciclo de desenvolvimento em ambientes co-

localizados, este fato pressupõe que os principais desafios inerentes ao desenvolvimento distribuído não são cobertos, individualmente, por cada uma das metodologias ágeis.

Neste trabalho foi realizada uma análise das características das principais metodologias ágeis, chegando-se a constatação de que nenhuma delas possui todos os atributos necessários para uso em ambiente distribuído, sendo necessário verificar quais características de cada uma delas poderia ser utilizada neste ambiente.

Para projetos distribuídos do tipo *offshore*, e (ou) *outsourcing*, é importante utilizar processos mais detalhados, com um maior número de papéis pré-estabelecidos. Souza (2007) constatou que o UP, com alguns ajustes, é um processo adequado para grandes projetos distribuídos, onde existe grande necessidade de documentação, gerenciamento de requisitos, e gerenciamento relacionado a diferentes culturas e fusos-horários. O ciclo de vida do UP é indicado para projetos onde as iterações podem durar até seis meses, nos projetos ágeis é indicado o uso de pequenas iterações com duração de duas a quatro semanas. Um maior número de iterações deixa o processo mais ágil. O desenvolvimento *onshore insourcing*, possui menos pontos críticos, comparado aos outros tipos (vide Fig. 13) possibilitando o uso de processos mais leves.

O modelo sugerido propõe o uso das quatro fases da metodologia UP, demonstrado na Fig. 18. A utilização de fases bem definidas facilita o desenvolvimento distribuído, todos os *stakeholders* saberão claramente em que fase o projeto se encontra, e facilita o trabalho do gerente do projeto e líderes de equipes. O uso de todos os papéis e disciplinas do UP pode deixar o processo burocrático.

O modelo sugerido propõe práticas para fomentar a comunicação e o espírito de equipe em todas as fases de desenvolvimento, pois é destinado para aplicações distribuídas. Práticas como esta não estão explicitadas nas metodologias estudadas devido a sua natureza, de aplicação *on-site* (uso em equipes co-localizadas).

O XP é indicado para equipes pequenas, e projetos pequenos e médios. O foco é na satisfação do cliente, onde o mesmo deve estar presente em todas as etapas de desenvolvimento. No XP praticamente não existe documentação, e em

muitos casos o projeto é iniciado com a construção de um protótipo. O tempo de desenvolvimento indicado para o XP é inferior a um ano, esse tempo pode ser pequeno para projetos distribuídos.

Para um melhor gerenciamento no contexto distribuído, se faz necessário um maior cuidado com a documentação, porém o foco deve ser sempre o software funcional. O modelo de processo sugerido utiliza-se de práticas do XP na etapa de construção, porém nas outras fases são utilizadas características de outras metodologias.

Apesar da metodologia Scrum ter se mostrado adequada para o gerenciamento do processo, ela carece de práticas específicas para o DDS. Scrum destaca a comunicação e *feedback*, extremamente importantes para o sucesso de qualquer projeto, e propõe papéis que se mostram adequados para o desenvolvimento *onshore insourcing*, sem que se perca agilidade durante o desenvolvimento. As práticas de gerenciamento do Scrum foram sugeridas para aplicação, porém as fases do UP se mostraram melhores por serem 4 fases simples, e bem definidas. Assim como na fase de construção, onde foram sugeridas as práticas do XP.

Comparado a metodologia DSDM, o modelo sugerido possui semelhanças, pois também se baseia em práticas encontradas nas metodologias ágeis como XP, que possui práticas parecidas com a da DSDM. As fases da DSDM são similares às etapas de concepção, elaboração, construção, e transição. O modelo sugerido possui características especificas para *onshore insourcing*, dando mais ênfase para a comunicação entre as equipes e clientes, através de ferramentas de apoio, dado a dificuldade de se fomentar a comunicação informal entre os *stakeholders*. O modelo sugerido também não estipula número máximo de integrantes por equipe como a DSDM, somente o número mínimo.

A metodologia FDD é indicada para vários tipos de projetos, inclusive pra projetos considerados críticos. Seu funcionamento é bastante simples, e pode ser facilmente comparado com as fases das outras metodologias. A metodologia FDD não possui características específicas para o desenvolvimento distribuído. O modelo sugerido descreve meios para fomentar a comunicação entre as equipes, como workshops e reuniões através de vídeo-conferência, e também descreve meios para

gerenciar as equipes dispersas, através das funções do gerente e dos líderes de equipes.

6.3 Ferramenta para Desenvolvimento Colaborativo

O uso de ferramentas para projetos distribuídos, também se mostra muito importante. As ferramentas, dentre outras funcionalidades, podem auxiliar na comunicação e integração entre as unidades de desenvolvimento. Um dos princípios do manifesto ágil enfatiza que deve-se dar prioridade aos indivíduos e interações, ao invés de processos de ferramentas, porém a utilização de ferramentas não deve ser descartada. Não foi encontrada nenhuma ferramenta que se mostrou ideal para todos os tipos de projetos. Pollice (2004) destaca que em algumas empresas, a própria equipe desenvolve suas próprias ferramentas para aplicação em seus projetos. Na aplicação de DDS, existe a necessidade de ferramentas e técnicas para apoiar os problemas relativos a distribuição.

O título ferramentas para desenvolvimento colaborativo se dá porque essas ferramentas são necessárias tanto no DDS quanto no desenvolvimento co-localizado. Porém, no desenvolvimento ágil deve-se ter cuidado para que o foco permaneça sempre nos nas pessoas envolvidas, diferentemente dos processos tradicionais onde, em alguns casos, os indivíduos são vistos como ferramentas para se chegar aos objetivos do projeto.

O uso de ferramenta *wiki* traz muitas facilidades ao DDS utilizando práticas ágeis:

- É de fácil criação, e pode ser desenvolvida de acordo com as características de cada projeto;
- De fácil acesso a todos os integrantes, pois basta ter acesso a internet e um *browser;* são fáceis de usar e configurar;
- A *Wikis* não possui uma estrutura fixa, a equipe deverá estruturá-la conforme as suas necessidades;
- Através dela, a equipe pode compartilhar informações sobre funcionalidades, fases e unidades de desenvolvimento, outras informações que se julgarem importantes para o projeto.

Devem ser atribuídas responsabilidades a integrantes da equipe, para realizarem o gerenciamento da mesma, onde deverão eliminar informações não mais necessárias para o decorrer do projeto, e realizar o monitoramento sobre o seu funcionamento.

O objetivo da criação da wiki é demonstrar com maior facilidade como funciona a interação entre os integrantes do projeto, utilizando as práticas sugeridas. A Fig. 27 mostra a tela de inicial da *wiki* desenvolvida. O *login* de usuário deve ser informado. Somente usuários previamente cadastrados tem acesso a informações da *wiki.* Cada tipo de usuário possui níveis de acesso diferenciados, com autorização para escrita, leitura e alteração das configurações.

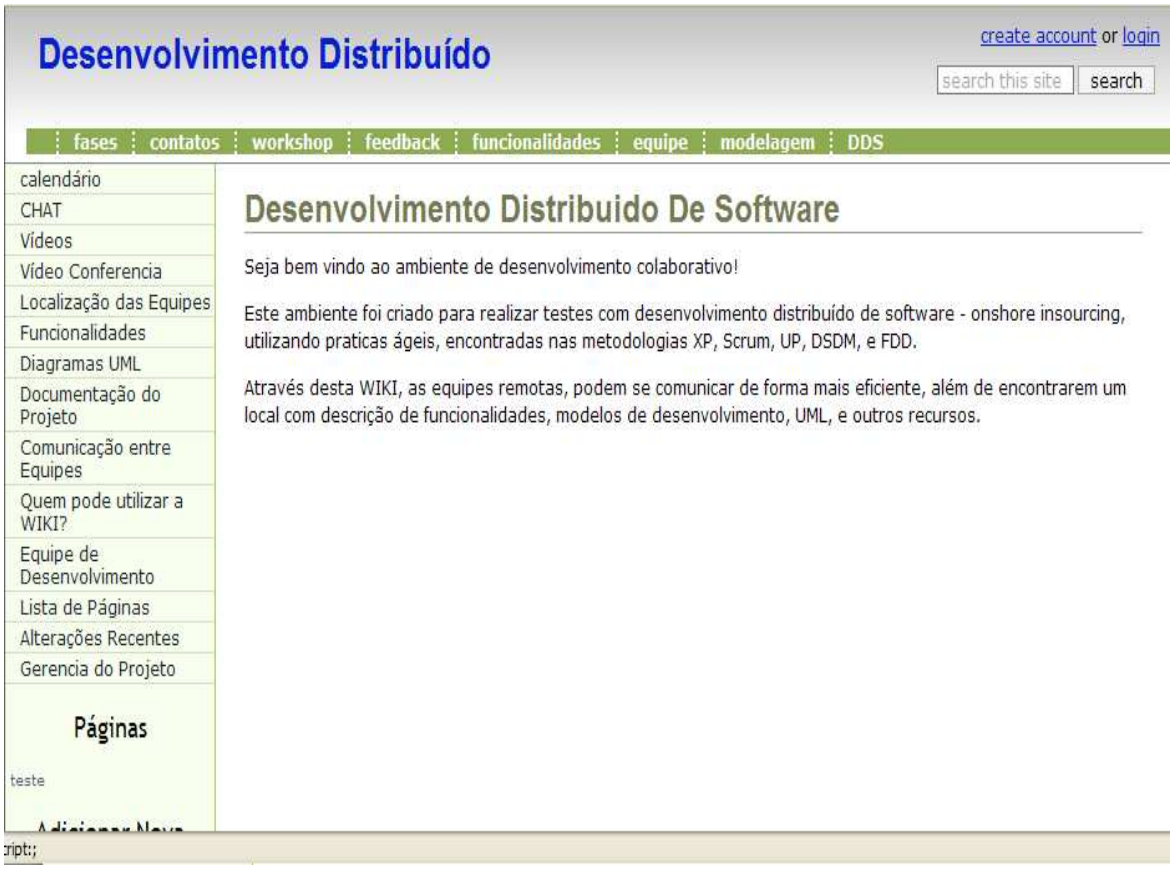

Figura 27 – Página inicial da *wiki*

Na ferramenta wiki criada para este trabalho foram incluídos diversos ítens que vão ao encontro das características descritas na sugestão (seção 5.3):

- Na seção de modelagem, a equipe tem acesso aos modelos das fases e dos papéis envolvidos, e também pode incluir e (aperfeiçoar) modelos para

desenvolvimento;

- Podem ser postados diagramas UML das funcionalidades;
- Qualquer equipe pode saber como está o andamento do projeto, a cada integração dos artefatos de software a *wiki* deve ser atualizada;
- Todos podem saber do andamento das fases de concepção, elaboração, construção e transição;
- Para facilitar o entendimento dos requisitos, pode ser utilizado a vídeos, onde são postados vídeos explicativos sobre as funcionalidades do sistema;
- Chat, para fomentar o espírito de equipe entre dos os integrantes;
- Fórum de discussão, onde qualquer integrante das equipes pode levantar assuntos relativos ao projeto;
- Através da *wiki,* é possível verificar a localização das equipes atuantes no projeto;
- O calendário de atividades auxilia a manter as equipes cientes das datas importantes, presentes no projeto, como datas previstas para início e fim das iterações, prazos para desenvolver funcionalidades, dentre outros itens;

Percebe-se a facilidade de edição de páginas existentes e na implementação de novas páginas e funcionalidades nesta ferramenta, que pode ser facilmente desenvolvida durante um projeto de software. A Fig. 28 demonstra a página com acesso a e-mails dos integrantes do projeto.

Figura 28 – Acesso a integrantes da equipe

A Fig. 29 demonstra o acesso a ferramenta de *chat* entre os integrantes do projeto. É importante que a comunicação informal seja incentivada entre todos os *stakeholders* envolvidos no projeto.

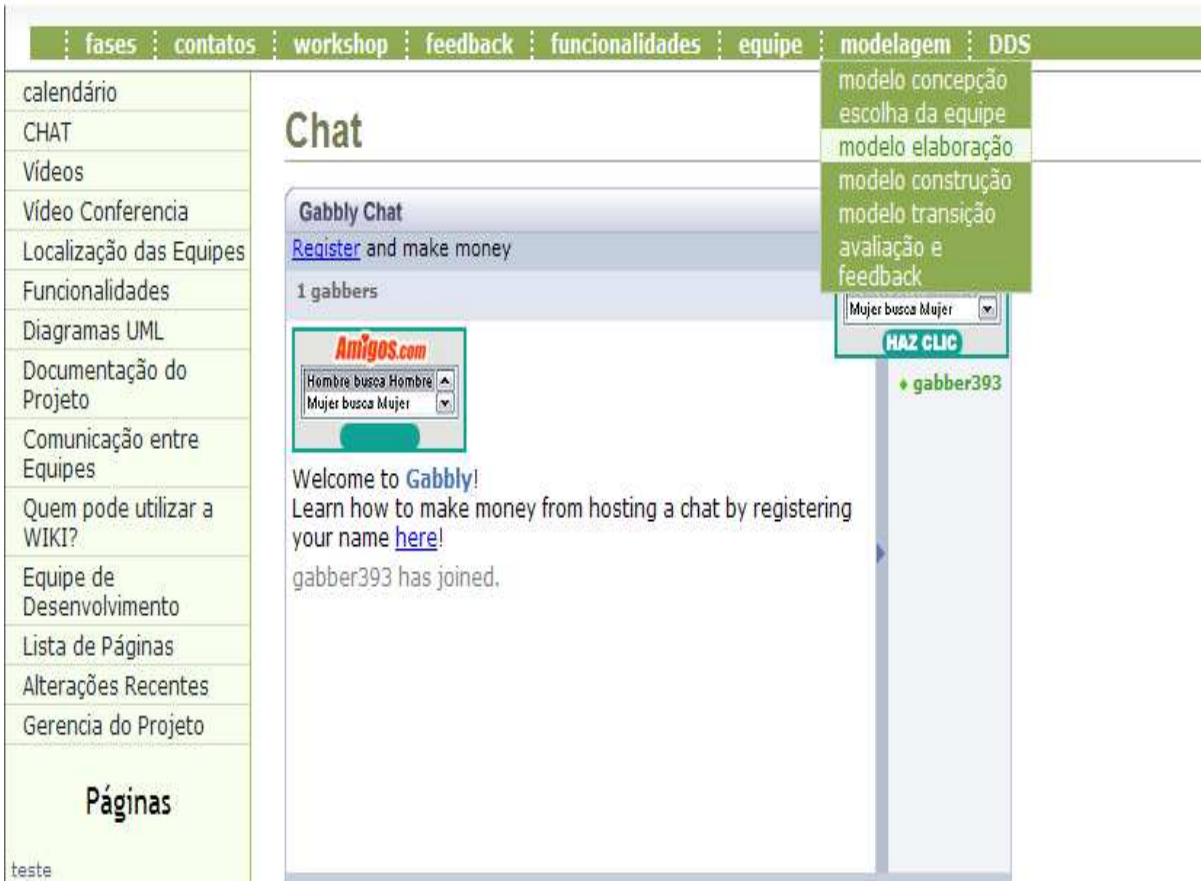

Figura 29 – Interface de comunicação

Existem outras ferramentas que podem ser utilizadas no DDS, essas ferramentas podem auxiliar, por exemplo, durante a integração dos artefatos de software, e na etapa de análise dos requisitos.

É importante que cada empresa desenvolva e avalie o processo e ferramentas para desenvolvimento. O foco deve ser o desenvolvimento de software com agilidade, mas nunca deve-se esquecer que o produto final deve ter qualidade. O *feedback* constante permite que as práticas adotadas e as ferramentas utilizadas sejam avaliadas e aperfeiçoadas durante o desenvolvimento de software. Toda equipe deve saber de todas as características do processo, e saber das práticas e valores descritos para o desenvolvimento ágil (Apêndice A). O fato de simplesmente aplicar um determinado modelo de ciclo de vida de software, não significa que estará realizando um processo de desenvolvimento ágil.

7 CONCLUSÃO

De acordo com o estudo realizado, pode-se constatar que é possível realizar o DDS *onshore insourcing* utilizando metodologias ágeis. Esta afirmação é facilmente comprovada baseada no fato de já existir aplicações utilizando a metodologia como o XP neste tipo de contexto. Como foi demonstrado no capítulo 5, a metodologia UP também se mostra adequada para este tipo de aplicação. Além do XP e do UP, as outras metodologias possuem características importantes, e se mostram adequadas para suprir possíveis deficiências encontradas na aplicação distribuída *onshore insourcing.*

A investigação empírica sobre as práticas das metodologias ágeis, e seu efeito no DDS *onshore insourcing*, resultaram em um conjunto de lições aprendidas. O objetivo específico de aprofundar o conhecimento em relação as metodologias ágeis e ao DDS foi atingido, conforme demonstrado na fundamentação teórica nos capítulos 3 e 4.

Não existe uma metodologia que se mostre ideal para todos os tipos de aplicação, porém é importante que haja um processo comum para todas as equipes. Nenhuma metodologia, em sua forma original, se mostrou eficiente e completa para aplicação *onshore insourcing*, onde foi necessário realizar a análise das melhores práticas para este tipo de aplicação.

A sugestão de práticas para aplicação *onshore insourcing*, juntamente com a modelagem de cada uma das fases de desenvolvimento, fornece um auxilio para a sua aplicação neste contexto. Sabe-se que cada empresa e projeto possuem peculiaridades, onde deve-se avaliar possíveis adaptações e melhorias no PDS.

Também não foi encontrada nenhuma ferramenta completa o suficiente para todas as aplicações, sendo necessário um estudo mais aprofundado com relação a

sua usabilidade em ambiente distribuído. Verificou-se que o uso de *wikis*, tem dado bons resultados, onde se sugere que a própria equipe de desenvolvimento desenvolva sua própria ferramenta *wiki,* para desenvolvimento colaborativo.

Este trabalho pode auxiliar no desenvolvimento distribuído demonstrando uma forma para desenvolver software baseada nas principais metodologias ágeis. A sugestão de práticas e fases de desenvolvimento, aliada a modelagem, aplicados na prática podem trazer vantagens para o processo de desenvolvimento.

De forma geral, percebe-se que ainda existe muito a ser pesquisado envolvendo esta área da engenharia de software. Estudos futuros podem ser realizados dando continuidade a este trabalho, como: estudos relacionados a ferramentas para aplicação em DDS; a análise de outras metodologias ágeis existentes, assim como a análise em relação a outros tipo de DDS; aplicação prática entre diferentes unidades de desenvolvimento, podendo ser realizada em empresas reais, ou entre unidades de ensino da nossa região, do pais, ou em conjunto com outros países (*offshore*).

Verifica-se, também, que as metodologias ágeis e o DDS são áreas crescentes e muito promissoras, mostrando-se necessárias nos presentes currículos dos diferentes cursos relacionados a área tecnológica.

REFERÊNCIAS

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. ***Agile Software Development Methods: Reviews and Analysis***. Espoo: VTT *Publications*, 2002. Disponível em: <http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf> Acesso em: 11 setembro de 2008.

BECK, Kent. ***Extreme Programming Explained***.,*New York*: 2000, *Addison Wesley*.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML.** 2. ed. Rio de Janeiro: Campus, 2007.

BOOCH, G. ***Object Solutions: Managing the Object-Oriented Project***. *Addison-Wesley*. 1996.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. Rio de Janeiro: Elsevier, 2005.

BORBOREMA, Thiago. **Impacto da aplicação da metodologia XP nas organizações de desenvolvimento de software**. Monografia - Curso de Sistemas de Informação, Faculdade de Filosofia Ciência e Letras Eugênio Pacelli, Universidade do vale do Sapucaí, Pouso Alegre, 2007.

BONA, C. **Avaliação de processos de software:** um estudo de caso em XP e Iconix. 2002. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-Graduação em Engenharia de Produção, UFSC, Florianópolis.

BOEHN, B. **A view of 20th and 21st century software engineering**. Proceedings of 28th International Conference on Software Engineering (ICSE), p. 12-29, Xangai, 2006.

BRAGAGLIA, Ulisses**. Ferramenta de apoio ao ensino colaborativo em disciplinas de graduação, baseada em WIKI.** Monografia de Conclusão de Curso de Bacharelado em Sistemas de Informação; PUC-RS, Porto Alegre-RS, 2006

CARMEL, E. ***Global Software Teams - Collaborating Across Borders and Time Zones***. *New Jersey: Prentice Hall*, 1999.

COAD, P. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1996.

COAD, Peter; LEFEBVRE, Eric; LUCA, Jeff. ***Java Modeling In Color With UML: Enterprise Components and Process***. *Upper Saddle River, N.J.: Prentice Hall*, 1999.

DAMIAN, D.; WILLIAMS, L.; LAYMAN, L.; BURES, H. ***Essential communication practices for extreme programming in a global software development team.*** *Information & Software Technology* 48(9), 781–794, 2006.

DAMIAN, D.; ZOWGHI, D. ***The impact of stakeholders' geogrephical distribution on managing requirements in a multi-site organization***. *Procedings of the international Conference on Requierements Engineering, p. 319-328. Essen*, 2002.

EBERT, C.; PARRO, C. H.; SUTTELS, R.; KOLARCZYK, H. ***Improving validation activities in a global software development,*** *in '23rd International Conference on Software Engineering', IEEE Computer Society, CA*, pp. 545–554, 2001.

EVARISTO, Roberto; SCUDDER, Richard. ***Geographically distributed project teams: a dimensional analysis***. In: HICSS, 2000, *Havaí. Proceedings… EUA*, 2000. 15 p.

FARIAS, T. M. de M. **Aplicação de padrões ao processo de desenvolvimento de Software RUP**. Trabalho de Conclusão de Curso Engenharia da Computação. Universidade de Pernambuco, 2006.

FOWLER, Martin***. The New Methodology***. 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html> Acesso em: 18 de julho de 2008.

FOWLER, Martin. ***Using an agile software process with offshore development***. 2006. Disponivel em: <http://martinfowler.com/articles/agileoffshore.html> Acesso em : 2 de setembro de 2008.

GEFEN, David; SENN, James A. ***The Correlation Between Outsourcing and the Business Value of Information Technology****, IEEE Software Proceedings*, 2003, 3pp.

HERBSLEB, J. D.; MOITRA, D. ***Global software development***; IEEE *Software,* 2001.

HERBSLEB, J. D.; MOCKUS, A.; FINHOLT, T; GRINTER, R. E. ***An empirical study of global software development: distance and speed***. *Procedings of the 23º International Conference on Software Engineering (ICSE), p. 81-90. Toronto*, 2001

HERZUM, P., Sims, O. **Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise**, OMG Press, 2000.

HEPTAGON. **Heptagon, tecnologia da informação**. 2008. Disponível em:

<http://www.heptagon.com.br/> Acesso em 15 de setembro de 2008.

HIGHSMITH, J. ***Agile software development ecosystems***. *Boston, MA., Pearson Education,* 2002.

HIGA, W.; NETO, A. F.; FURLAN, J. D. **Engenharia da Informação - Metodologia, Técnicas e Ferramentas** - Editora Mc Graw Hill, p 18 – 52, 1996.

HOLMSTROM, H.; CONCHÚIR, E.; ÅGERFALK, P.; FITZGERALD, B. ***Global software development challenges: A case study on temporal, geographical and socio-cultural distance***, *in 'International Conference on Global Software Engineering', IEEE ComputerSociety, BR*. 2006.

JACOBSON, I. et al. ***The unified software development process***. *Addison-Wesley*, 1999.

KAROLAK, D. W. ***Global software development – managing virtual teams and environments*****,** *IEEE Computer Society, EUA*, 1998.

KEARNEY, A. T. ***Offshoring for Long-Term Advantage, Global Services Location Index, 2007.*** Disponivel em: <http://www.atkearney.com/res/shared/pdf/GSLI_2007.pdf> Acesso em 28 de agosto de 2008.

KHAN, Naureen; CURRIE, Wendy L.; WEERAKKODY, Vishanth; DESAI, Bhavini. ***Evaluating Offshore IT Outsourcing in India: Supplier and Customer Scenarios***. *In: Proceedings of the 36th Hawaii International Conference on System Sciences, IEEE Computer, 2003*, 10pp.

KROLL, P.; KRUCHTEN, P. ***The Rational Unified Process Made Easy: A Practitioner ' s Guide to the RUP***. *Addison Wesley*, 2003.

KRUCHTEN, P. ***Analyzing Intercultural Factors Affecting Global Software Development–A Position Paper***. *3rd Int. Workshop on Global Software Development (GSD 2004), Edinburgh, Scotland,* pp. 59-62, 2004.

LARMAN, Craig. ***Agile and iterative development : a manager's guide***. *Addison-Wesley*, 2003.

LIMA, Adailton M. **Execução descentralizada de processos de software baseada em contratos eletrônicos no ambiente WEBAPSEE**. Bacharelado em Ciência da Computação, Belém 2007.

MARTIN, Robert C. **Agile Processes**. 2002. Disponível em: <http://www.objectmentor.com/resources/articles/agileProcess.pdf> Acesso em 15 julho de 2008

MAYRING, P. **Introdução à pesquisa qualitativa: uma introdução para pensar qualitativamente.** 5a ed. *Weinheim: Beltz*, 2002.

MENDONÇA, Rosângela Míriam L. O. **Usabilidade de Processos**. II Workshop Um Olhar Sociotécnico sobre a Engenharia de Software – WOSES, 2006.

NAPHTA. **Metodologias ágeis**. Artigo, 2007. Disponível em <http://www.naphta.com.br/xpmanager/xpmanager_metodologiasageis.html> Acesso em 21 de Julho de 2008.

MANIFESTO, 2001. ***Manifesto for Agile Software Development***, 2001. Disponível em: <http://www.agilemanifesto.org/> Acesso em 22 de Junho de 2008.

MANHÃES, Vinícius T. ***Extreme Programming* - Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. Rio de Janeiro: 2004, Novatec Editora.

MARQUARDT, M. J.; HORVATH, L. ***Global Teams: how top multinationals span boundaries and cultures with highspeed teamwork****. Davies-Black Publishing, USA,* 2001.

MARCHESI, M.; SUCCI, G.; WELLS, D.; WILLIAMS, L. ***Extreme Programming Perspectives****, Michigan: Addison Wesley Professional*, 640p, 2002.

PALMER S. R., FELSING J. M. ***A Practical Guide to Feature-Driven Development*** *(The Coad Series) , Prentice Hall* PTR, USA, 2002

PAULISH, D. J.; BASS, M.; HERBSLEB, J. D. ***Global software development at siemens: Experience from nine projects****. in '27th International Conference on Software Engineering (ICSE'05)', IEEE Computer Society, USA*, 2005.

PILATTI, Leonardo S. M. **Estrutura e características para análise de ambientes de desenvolvimento global de software em organizações *offshore insourcing***. Mestrado em Ciencia da Computação, PUC-RS – Porto Alegre, 2006.

PILATTI, Leonardo; PRIKLADNICKI, Rafael; AUDY, J. L. N. ***Global software development: standadization of the developing phase based on the MSF framework in a global CMM level 3 context****. Procedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE), p. 235-240, Taipei,* 2005.

POLLICE, Gary; AUGUSTINE, Liz; LOWE, Chris; MADHUR, Jas. ***Software Development for Small Teams: A RUP-Centric Approach****. Addison Wesley*, 2004.

PRESSMANN, Roger. S**. Engenharia de software.** Rio de Janeiro: McGraw-Hill, 1995.

PRIKLADNICKI, Rafael. **MuNDDoS Um Modelo de Referência para Desenvolvimento Distribuído de Software**. Dissertação de mestrado, PUC-RS. Porto Alegre, 2003.

PRIKLADNICKI, Rafael; AUDY, Jorge. **Desenvolvimento distribuído de software**. Rio de Janeiro: Elsevier, 2008.

RAYMOND, E. S. **The Cathedral and the Bazaar.** 1998. Disponível em: <http://www.firstmonday.org/issues/issue3_3/raymond/> Acesso em 10 de agosto de 2008

REIS, Rodrigo; SOUZA, Cleidson de. **Processo Unificado.** Apresentação de aula, UFPA. Disponível em: <http://www2.ufpa.br/cdesouza/teaching/cedai/10-rup.pdf>. Acesso em 15/10/2008

REPONEN, Tapio. ***Outsourcing or Insourcing?,*** *Communications of the ACM, vol 57. no 5,* 2002 12pp.

ROCHA Thayssa Á. da; OLIVEIRA, S. R. B.; VASCONCELOS, Alexandre M. L. de. **Adequação de Processos para Fábricas de Software**. 2004. UFPE, Recife – PE; UNAMA, Belém – PA. Disponivel em: <http://www.simpros.com.br/Apresentacoes_PDF/Artigos/Art_12_Simpros2004.pdf> Acesso em 23 de agosto de 2008.

ROCKENBACH, A. ***Troubleshoot your way to success in remote team management!*** *Project Management Institute* 13(4), 145–161, 2003.

SAUR, Ricardo A. C. **Perspectivas e projeções da indústria global de software e serviços; O futuro da indústria de software: a perspectiva do Brasil**. coletânea de artigos / Ministério do Desenvolvimento, Indústria e Comércio Exterior, Instituto Euvaldo Lodi / Núcleo Central. Brasília : MDIC/STI : IEL/NC, 2004.

SCHWABER, Ken. ***Agile Project Management with Scrum****. Microsoft Press*, 2004.

SLIGER, Michele; BRODERICK, Stacia. ***The Software Project Manager's Bridge to Agility***. *Addison Wesley Professional*, 2008.

SOARES, Felipe S. F.; MARIZ, Leila M. R. de S.; CAVALCANTI, Yguaratã C. C.; RODRIGUES, Joseane P. R.; NETO, Mário G.; BASTOS Petrus R.; ALMEIDA, Ana Carina M.; PEREIRA, Daniel Thiago V. P.; ARAÚJO, Thierry da S.; CORREIA, Rafael S. M.; ALBUQUERQUE, Jones. **Adoção de SCRUM em uma Fábrica de Desenvolvimento Distribuído de Software.** Centro de Informática – UFPE – Recife – PE – Brasil, 2007.

SOARES, Michel dos S. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004.**
Disponível em <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> Acesso em 21 de Agosto 2008.

SOUZA, Mateus F. de**. Análise de processos de desenvolvimento de software para aplicação em desenvolvimento distribuído de software**. Graduação em Ciência da Computação. UFPEL, 2007.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Addison Wesley, 2003.

STANDISH GROUP INTERNATIONAL, Inc., **The, Extreme Chaos.** 2001. Disponível

em
<http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf> Acesso em:12 de setembro de 2008.

STANDISH GROUP INTERNATIONAL, Inc **CHAOS Report Shows Project Success Rates Have Improved by 50%**. 2003. Disponível em: <http://www.standishgroup. com/press/article. php? id=2> Acesso em 31 de setembro de 2008.

TAMAKI, Paulo A. O.; HIRAMA Kechi**. Melhoria de Processos de Desenvolvimento de Software Aplicando *Process Patterns***. *Journal of Computer Science volume 6, número 1, 2007 p.80-89*. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v6.1/art09.pdf> Acesso em 28 de setembro de 2008.

TANAKA, Sergio A.; BANKI,André. *RUP 7.0 Alignment with the agile manifesto values.* E-Tech: Atualidades Tecnológicas para Competitividade Industrial, Florianópolis, 2008. Disponível em: <http://revista.ctai.senai.br/index.php/edicao01/article/download/19/17> Acesso em 12 de maio de 2008.

TEIXEIRA, Daniel D.; PIRES, Fernando Jorge A.; PINTO, José Pedro G. de S.; SANTOS, Tiago Alexandre G. P. **DSDM – *Dynamic Systems Development Methodology.*** Faculdade de Engenharia da Universidade do Porto, 2005.

TEIXEIRA, Juliano Machado. **Modelagem de um Ambiente de Desenvolvimento Distribuído de Software baseado em *Workflow***. Graduação em Ciência da Computação. UFPEL, 2007.

URDANGARIN, Roger G. **Uma Investigação sobre o Uso de Práticas *Extreme Programming* no Desenvolvimento Global de Software**. Dissertação PUC-RS; Porto Alegre, 2008.

WAZLAWICK, Raul S. **Análise de sistemas de informação orientados a objetos**. Rio de Janeiro: Elsevier, 2004.

WELLS, D. **XP material**. Disponível em: <www.extremeprogramming.org>. 2001. Acesso em 28 de julho de 2008.

Apêndice

APÊNDICE A – O Manifesto Ágil

O Manifesto Ágil

Os 4 Valores do Manifesto Ágil

O Manifesto Ágil se baseia nestes quatro valores:

- **Indivíduos e interações** ao invés de processos e ferramentas
- **Software Funcionando** ao invés de documentação
- **Colaboração do cliente** ao invés de negociação de contrato
- **Respostas rápidas as mudanças** ao invés de seguir um plano

Indivíduos e Interações ao Invés de Processos e Ferramentas: a programação é uma atividade humana (LARMAN, 2003). O manifesto ágil não enxerga as pessoas como simples peças que podem ser substituídas, ele encoraja a interação, fortalecendo o espírito de equipe. É importante que todos os envolvidos no processo se relacionem entre si, desde desenvolvedores até clientes. Os métodos tradicionais enfatizam que os envolvidos no processo possuem posições e atribuições fixas durante o desenvolvimento, como analistas ou programadores. Os métodos ágeis focam em qual tarefa será desempenhada, podendo haver a troca de funções entre os envolvidos, mesmo depois de iniciado o projeto.

O manifesto ágil indica que deve-se priorizar as pessoas e interações, ele não diz que deve-se esquecer dos processos e ferramentas, pois os mesmos também são importantes. Ter bons profissionais envolvidos no desenvolvimento é importante para formar uma boa equipe e chegar a bons resultados, porém, um processo inadequado pode fazer com que bons profissionais não cheguem a bons resultados. As pessoas são mais importantes que os processos, pessoas utilizando um bom processo, renderão mais do que pessoas que não utilizam algum tipo de processo (BOOCH, 1996). As ferramentas também têm sua importância, mas nunca serão mais importantes que o seus utilizadores.

Software Funcionando ao Invés de Documentação: nenhum documento deve ser gerado, a não ser que seja de extrema importância. De nada adianta termos uma extensa documentação, completa e detalhada se o software não estiver funcionando. No desenvolvimento ágil, novas versões do software são desenvolvidas em intervalos pequenos de tempo, é importante que o código seja simples e objetivo, diminuindo assim a carga de documentação. A documentação deve ser feita em sincronia com o desenvolvimento, e somente o necessário deve ser gerado.

No desenvolvimento ágil, a transferência de conhecimento entre os membros da equipe é feita através da interação, no decorrer do projeto, entre todos os envolvidos. A codificação e a documentação são feitas com o auxilio desta interatividade, ajudando a eliminar a falta de objetividade ou dupla interpretação das funcionalidades do sistema.

Colaboração do Cliente ao Invés de Negociação de Contrato: o desenvolvimento ágil de software requer inovação e responsabilidade. É importante compartilhar o conhecimento dentro da equipe de desenvolvimento, como também com o cliente. O desenvolvimento ágil fomenta a colaboração entre desenvolvedores, clientes e usuários para que o processo tenha qualidade e agilidade (LARMAN, 2003). É importante firmar contratos, porém, esta não é a única ou a melhor forma de comunicação com o cliente, pois além do contrato, é importante manter contato e incentivar a sua colaboração durante todo o processo de desenvolvimento. Ninguém melhor do que o cliente para saber de todos os requisitos do sistema, requisitos estes que podem variar durante o desenvolvimento. Agilidade no desenvolvimento envolve entrega da software desde o seu início, mediante prazos definidos, reduzindo assim o risco do não cumprimento de contratos. Além disso, um simples contrato não garante que o software desenvolvido satisfaça as necessidades do cliente. Para se ter qualidade é importante que o cliente obtenha um *feedback* durante o desenvolvimento a fim de avaliar se o mesmo está atendendo as suas necessidades.

Respostas Rápidas as Mudanças ao Invés de Seguir um Plano: desenvolvimento de software é muito comum haver variação nos requisitos, , podem ser alterados, eliminados, e também deve haver a possibilidade de inclusão de novos requisitos. Um bom projeto será aquele que se adaptar as mudanças

inerentes ao seu desenvolvimento. Um processo ágil disponibiliza respostas rápidas frente aos imprevistos que podem ocorrer durante o desenvolvimento, pois não seguem um plano inicial rígido, são iterativos e incrementais. O manifesto ágil não rejeita a utilização de processos, planejamento, documentação, ou contratos, ele mostra que são apenas pontos secundários.

Os 12 Princípios do Manifesto Ágil

Dentro destes quatro valores do manifesto ágil, foram criados 12 princípios básicos (MANIFESTO, 2001), estes princípios servem para caracterizar as metodologias ágeis, e as diferenciam das metodologias conhecidas como tradicionais.

"Nossa maior prioridade é satisfazer o cliente por meio da entrega contínua e antecipada de software de qualidade." O desenvolvimento é iterativo, e é prioridade entregar partes do sistema funcionando durante as etapas do desenvolvimento. A entrega contínua mantém o cliente satisfeito e ajuda a manter o foco do desenvolvimento, pois o software é avaliado continuamente pelo cliente, podendo fazer alterações nos requisitos do mesmo e guiando os próximas etapas de desenvolvimento.

"Receber bem as mudanças de requisitos, mesmo que em fase avançada do desenvolvimento." Permite que ocorram mudanças nos requisitos em qualquer etapa do desenvolvimento. Requisitos podem ser incluídos, alterados, ou eliminados. Não é necessário saber de antemão, no início do projeto, todos os requisitos do sistema.

"Entregar software funcionando freqüentemente, de algumas semanas até dois meses, de preferência para a escala de tempo mais curta." As metodologias ágeis são iterativas. A cada iteração devem ser produzidos artefatos de software funcional para o cliente. Esta característica ajuda a manter o cliente satisfeito, e permite uma melhor avaliação com relação ao andamento do projeto.

"Cliente e desenvolvedores têm de trabalhar juntos diariamente durante o projeto." O cliente deve estar presente durante o desenvolvimento do sistema,

assim, poderá auxiliar em possíveis dúvidas relativas, se manterá atualizado e, possivelmente, satisfeito sobre o andamento do projeto.

"Executar projetos em torno de indivíduos motivados. Dê a eles o ambiente e recursos que eles precisam, e confie neles para que o trabalho seja feito." A equipe pracisa estar sempre motivada, e deve ter acesso a todos os recursos necessários para o trabalho. Torna-se necessário o espírito de equipe. Deve-se levar em conta que cada indivíduo possui características e potencialidades diferentes, mas que trabalhando em equipe obterão bons resultados.

"O método mais eficiente e efetivo de transmitir informação para uma equipe de desenvolvimento e difundi-la internamente é a conversação cara-a-cara." Uma boa comunicação é imprescindível para o sucesso do projeto. A melhor forma de comunicação é a conversa cara-a-cara com os integrantes da equipe. Essas conversas podem ser desde assuntos técnicos até conversas informais de corredor.

"Software funcionando é a medida primordial de progresso." A melhor avaliação com relação ao desempenho do projeto é através do software funcional, onde a equipe e o cliente poderão avaliar o progresso do desenvolvimento de forma mais clara.

"Processos ágeis promovem desenvolvimento sustentável. Patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente." O desenvolvimento ágil ajuda a reduzir custos e o tempo do projeto. Os desenvolvedores devem ter condições e tempo de trabalho que permitam manter um bom ritmo, com qualidade. Nas metodologias tradicionais o custo e prazos de entrega, são estipulados no início do projeto. Nas metodologias ágeis o custo pode ser subdividido de acordo com as iterações e com o software funcional desenvolvido.

"Atenção contínua a excelência técnica, e boas práticas de projeto aumentam a agilidade." Deve ser desenvolvido software com qualidade técnica, a equipe deve seguir as mesmas práticas aumentando a agilidade do processo. Os artefatos de software produzidos devem estar sempre atualizados.

"Simplicidade – a arte de maximizar o montante de trabalho não feito – é essencial." Deve ser produzido somente o que é necessário, somente os requisitos

propostos pelo cliente devem ser desenvolvidos, a não ser que sejam solicitadas novas funcionalidades.

"As melhores arquiteturas, requisitos e projetos surgem das equipes auto-organizadas." No desenvolvimento ágil, a equipe tem total liberdade para se auto-organizar, podendo haver trocas de funções entre os integrantes, se for constatado um melhor aproveitamento. Este fato não exclui a figura do líder de projeto.

"Em intervalos regulares, a equipe reflete em como se tornar mais eficiente, então sintoniza e ajusta seu comportamento adequadamente." Deve haver um *feedback* das atividades realizadas, com a intenção de tornar o desenvolvimento mais eficiente. A equipe deve se reunir periodicamente para fazer avaliações e melhorias, tornando o desenvolvimento mais ágil.