MATEUS FEIJÓ DE SOUZA

ANÁLISE DE PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE* PARA APLICAÇÃO EM DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE*

Trabalho acadêmico apresentado ao Departamento de Informática 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. Flávia Braga de Azambuja

Pelotas, 2007

Dados de catalogação na fonte:
Ubirajara Buddin Cruz – CRB-10/901
Biblioteca de Ciência & Tecnologia - UFPel

S729a Souza, Mateus Feijó de
Análise de processos de desenvolvimento de software para aplicação em desenvolvimento distribuído de software / Mateus Feijó de Souza ; orientador Flávia Braga de Azambuja. – Pelotas, 2007. – 91f : il. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2007.

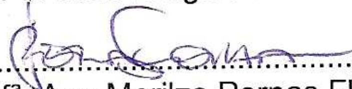
1.Informática. 2.Engenharia de software. 3.Desenvolvimento distribuído de software. 4.Processos de desenvolvimento de software. 5.RUP. I.Azambuja, Flávia Braga de. II.Título.

CDD: 005.1

BANCA EXAMINADORA:

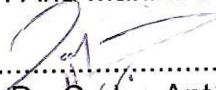
....

Profª. Flávia Braga de Azambuja, MSc. (Orientadora)



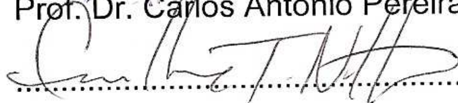
....

Profª. Ana Marilza Pernas Fleischmann, MSc.



....

Prof. Dr. Carlos Antônio Pereira Campani



....

Prof. Guilherme Tomaschewski Netto, MSc.

AGRADECIMENTOS

Agradeço a Deus, por estar ao meu lado, sempre pres

Implementação é um fluxo muito importante. Nele o sistema é codificado na linguagem de programação especificada. Também são feitos testes e a integração com módulos desenvolvidos em outros ciclos.

Testes verificam todo o sistema, analisando se este está de acordo com o que foi especificado. Valida também a integração entre objetos já implementados. Tem como meta, encontrar defeitos antes da implantação do *software*.

Implantação (Instalação) implanta o sistema para o usuário final, dando ênfase no treinamento, instalação e suporte.

Gerência de Configuração e Mudanças gerencia documentos gerados no decorrer do projeto, para que estes sempre estejam em conformidade ao que já foi desenvolvido.

Gerência de Projeto engloba o gerenciamento de riscos, planejamento e o acompanhamento do projeto.

Ambiente define como o RUP foi configurado para ser utilizado no projeto, além da organização do ambiente de trabalho para toda a equipe de desenvolvimento.

Ao contrário de outras metodologias, as disciplinas, ou *workflows*, não são só seqüenciais e são repetidas diversas vezes ao longo das várias fases do processo, em várias iterações, possivelmente realizando atividades diferentes ou com um grau variável de intensidade e detalhe (SILVA; VIDEIRA, 2005). A Fig. 10 mostra a relação entre as disciplinas (*workflows*) e as fases.

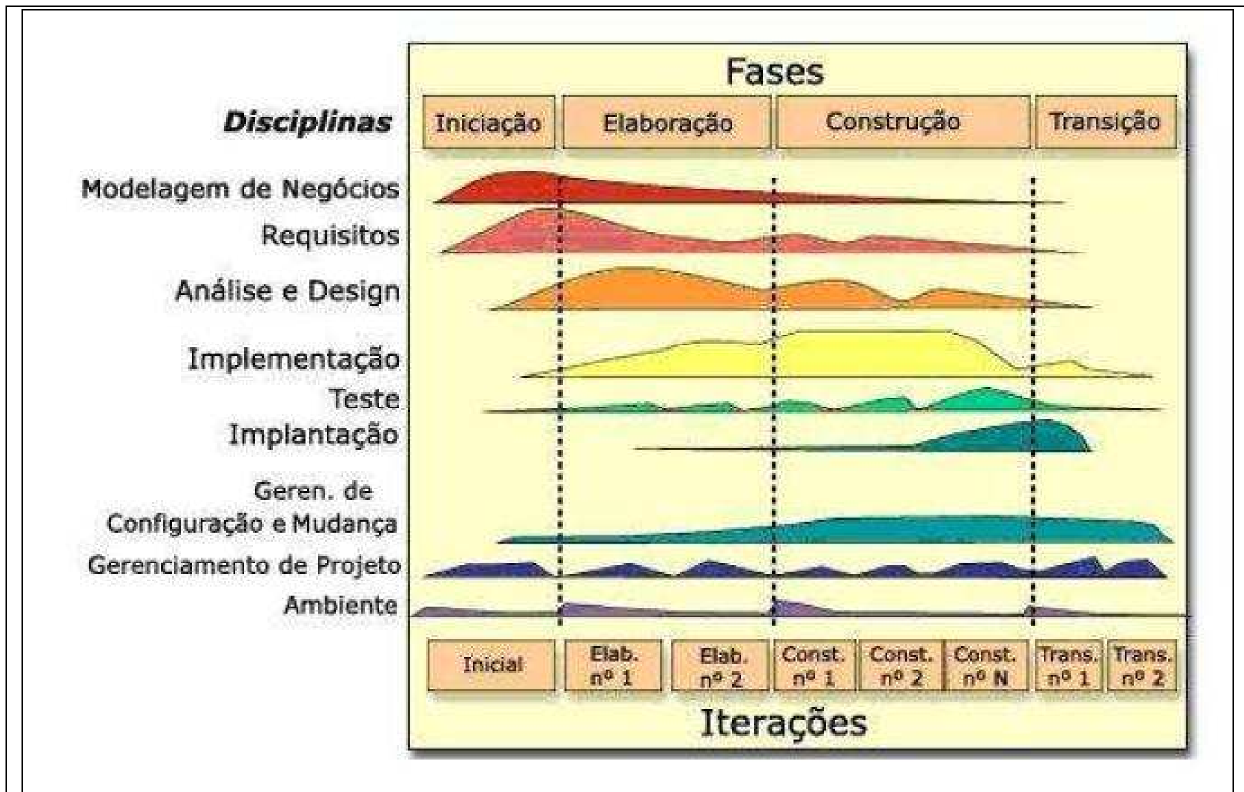


Figura 10 - Visão geral das disciplinas e fases do RUP.

Fonte: WWW.RATIONAL.COM/PRODUCTS/RUP

O RUP também utiliza artefatos, que são produtos gerados durante o desenvolvimento do projeto. Usualmente, os artefatos podem ser documentos (relatórios de riscos), modelos (casos de uso) e modelo de elementos (diagrama de classes). Esses artefatos são agrupados em disciplinas (*workflows*), pois cada uma produz um conjunto de artefatos diferentes (KRUCHTEN, 2000).

Segundo Kruchten (2000), o conceito central mais importante de todo o processo é o de atores (*workers*). Um ator define o comportamento e as responsabilidades de um membro da equipe, ou de um grupo deles trabalhando como uma equipe, dentro do contexto de uma empresa de Engenharia de *Software*. Os atores são os papéis desempenhados por pessoas dentro do projeto, que definem como estes devem trabalhar. Esses papéis irão definir as principais atividades e responsabilidades do membro da equipe, ou grupo, mostrando como estes devem se comportar.

As atividades são as ações realizadas por um ator que geram algum resultado significativo para o projeto. As atividades são usadas para medir o

rendimento de um determinado indivíduo desempenhando determinado papel. Elas têm um propósito claro que geralmente é criar, ou atualizar, algum artefato, seja ele um diagrama ou modelo. Essas atividades usualmente variam de poucas horas a poucos dias, e são designadas especificamente a um determinado papel. Servem também como elemento de decisão de planejamento (KRUCHTEN, 2000).

De acordo com Kruchten (2000), como o RUP é um processo iterativo, as atividades repetidas podem voltar a ocorrer, principalmente se tratando de alterações em artefatos de diagramas e modelagem, pois estes devem ser alterados a cada mudança na especificação de requisitos. As atividades são divididas em passos e esses passos podem se encaixar em três categorias: pensar (os atores entendem a natureza da tarefa, juntam e examinam os artefatos de entrada, formulando a saída), agir (aonde ocorre efetivamente à criação ou alteração dos artefatos) e revisar (os atores analisam o que foi desenvolvido, baseando-se em algum critério).

O RUP apresenta um metamodelo em termos de seus principais elementos e relacionamentos com objetivo de tornar possível a identificação de que elementos são permitidos e quais as relações válidas entre estes elementos. Os elementos do processo são definidos no metamodelo como classes e operações. Os elementos do tipo classe descrevem os objetos do metamodelo e elementos do tipo operação descrevem o comportamento destes objetos. A Fig. 11 ilustra o metamodelo do RUP.

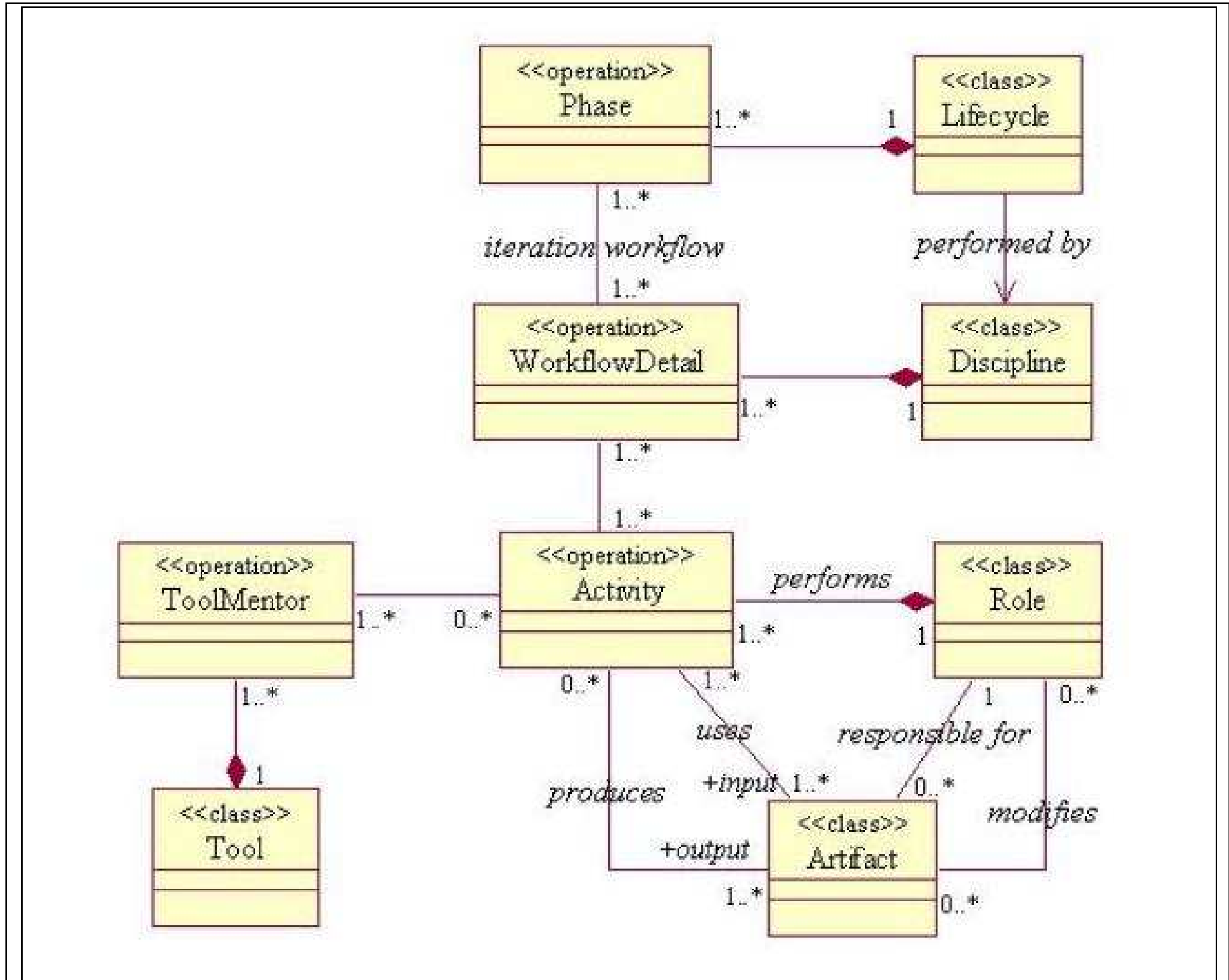


Figura 11 - Metamodelo do RUP.

Fonte: BENCOMO, 2005.

O metamodelo do RUP é um modelo de processos que identifica exatamente quais elementos são permitidos em seu processo e ainda, define como os mesmos estarão relacionados. Deve-se considerar ainda, que o RUP apresenta uma ampla definição de processos oferecendo disciplinas que cobrem desde a modelagem de negócios da organização até a entrega final do produto de *software* para seus clientes. A adaptação de processos no RUP é tratada, especificamente, em uma de suas disciplinas chamada Ambiente (PEREIRA, 2005).

A complexidade do RUP não deve assustar nem desmotivar a sua adoção como metodologia de desenvolvimento. O RUP é um *framework* metodológico muito completo, que pode e deve ser adaptado ou instanciado à realidade concreta de diferentes organizações e tipos de projeto (SILVA; VIDEIRA, 2005).

4.2 *Iconix*

A metodologia de desenvolvimento de *software Iconix* foi promovida pela empresa *Iconix Software Engineering*, cujo foco de negócio reside na formação e produção de material para a educação em tecnologias. O *Iconix* engloba as tarefas de análise de requisitos, análise e projeto preliminar, projeto e implementação (SILVA; VIDEIRA, 2005).

O *Iconix* situa-se entre o RUP e o XP.É dirigido por casos de uso, iterativo e incremental, como o RUP, mas sem boa parte do trabalho adicional associado a este. É também relativamente pequeno e compacto, como o XP, mas mantém a análise e o projeto como parceiros atuantes. O *Iconix* também faz uso eficiente da UML, ao mesmo tempo em que mantém um foco claro na rastreabilidade dos requisitos (SCOTT, 2003).

De acordo com Rosenberg e Scott (1999), o *Iconix* tem como base responder algumas questões fundamentais sobre o *software*. Desta forma, utiliza técnicas da UML que auxiliam a prover a melhor resposta. Dentre as técnicas utilizadas estão: casos de uso, diagrama de classe de alto e baixo nível, análise de robustez, diagrama de seqüência e colaboração e diagrama de estado.

A Fig. 12 ilustra a visão geral do *Iconix*, segundo Silva e Videira (2005), que é devidamente explicada a seguir. Esta figura revela um aspecto importante da utilização da UML: a implementação de um sistema depende da versão detalhada do diagrama de classes final.

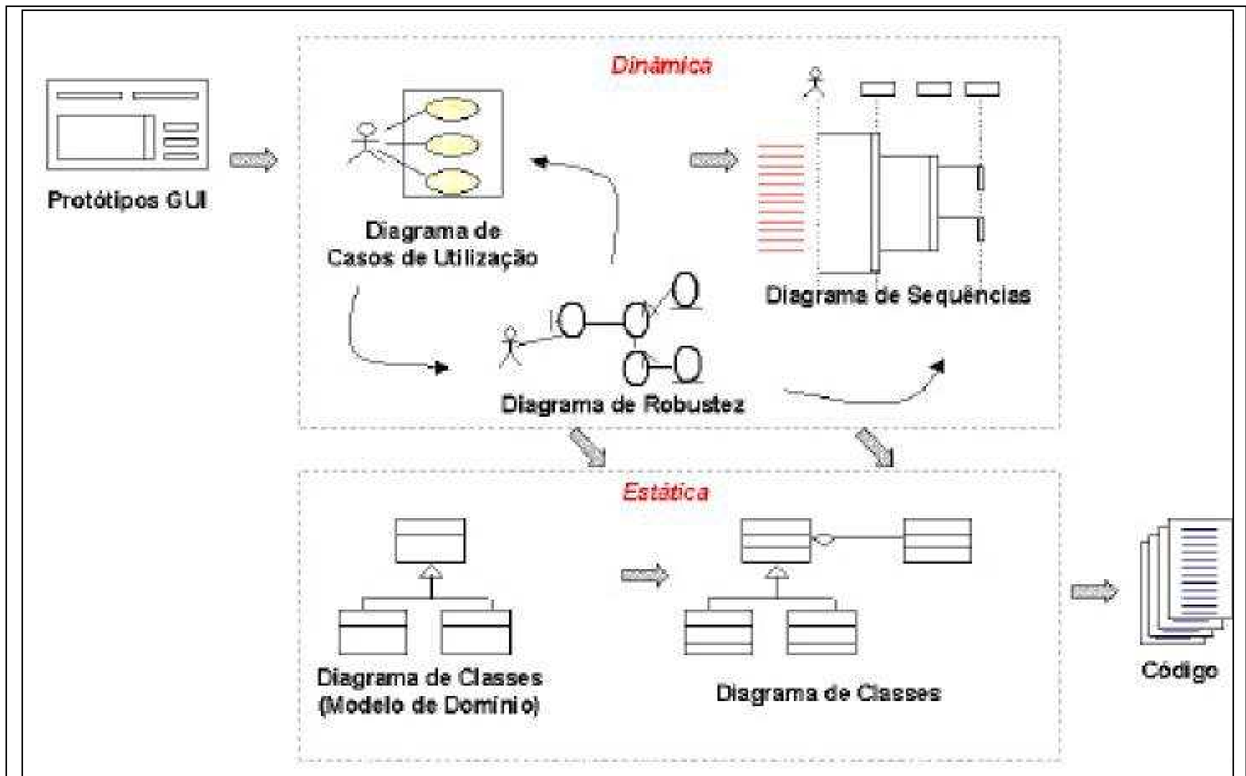


Figura 12 - Visão geral do *Iconix*

FONTE: SILVA E VIDEIRA, 2005.

A atividade de análise de requisitos é composta pelos protótipos GUI (protótipo de *interface* homem-máquina), diagrama de casos de uso e diagrama de classes (modelo de domínio). Detalhadamente ocorrem as atividades de: identificar no mundo real os objetos e todas as relações de agregação e generalização entre eles; apresentar uma prototipação rápida de *interface* do sistema, de forma que o cliente possa compreender melhor o sistema proposto; identificar os casos de uso do sistema mostrando os atores envolvidos; organizar os casos de uso em grupos; e associar requisitos funcionais aos casos de uso e aos objetos de domínio.

A atividade de análise e projeto preliminar é composta por diagrama de casos de uso, diagrama de robustez e diagrama de classes. As etapas especificadas que podem ocorrer são: escrever os casos de uso, com fluxo principal das ações, podendo conter o fluxo alternativo e o fluxo de exceção; apresentar a análise de robustez, sendo que, para cada caso de uso se deve identificar um conjunto de objetos e atualizar o diagrama de classes do modelo de domínio; e terminar a atualização do diagrama de classe.

A tarefa de projeto é composta por diagrama de robustez, diagrama de seqüência e diagrama de classes. Detalhadamente ocorrem as atividades de: especificar o comportamento através do diagrama de seqüência, para cada caso de uso, identificando as mensagens entre os diferentes objetos; terminar o modelo estático, adicionando os detalhes ao projeto no diagrama de classe; e verificar com a equipe se o projeto satisfaz todos os requisitos identificados.

A atividade de implementação é composta por diagrama de classes e código. As etapas especificadas que podem ocorrer são: utilizar diagrama de componente, se for necessário para apoiar a fase de desenvolvimento; escrever e gerar o código; realizar testes de unidade e de integração; e realizar testes de aceitação do usuário.

A metodologia *Iconix* apresenta uma série de alertas para cada atividade realizada, com o objetivo de advertirem sobre problemas e dúvidas comuns em equipes de desenvolvimento de *software*. Segundo Silva e Videira (2005) a caracterização dos alertas está relacionada com as seguintes técnicas:

- Modelo de domínio: não perder muito tempo com verificação gramatical; não adicionar multiplicidade muito cedo ao projeto; endereçar agregação e composição apenas na fase do projeto detalhado; e não desenhar um diagrama com mais de nove classes.
- Modelo de caso de uso: não tentar escrever casos de uso antes de saber o que os usuários realmente fazem; não desperdiçar tempo construindo modelos elegantes de casos de uso que não servem para construir o projeto; não perder tempo discutindo se vai usar *includes* ou *extends*; não usar *templates* textuais de caso de uso longos ou complexos.
- Análise de robustez: não tentar fazer um projeto detalhado do diagrama de robustez; e não perder tempo aperfeiçoando o diagrama de robustez à medida que o projeto evolui.
- Diagrama de seqüência: não tentar alocar comportamento aos objetos antes de saber realmente o que os objetos são; não iniciar o diagrama de seqüência antes de completar o diagrama de robustez associado; não focar a atenção na configuração de métodos *get* e *set* em vez de focalizar a atenção em métodos reais.

O *Iconix* possui uma abordagem essencialmente prática e projeta um sistema de *software* segundo o paradigma de orientação a objetos. O objetivo

principal do *Iconix* é, a partir de um conjunto de requisitos inicialmente definido, a construção do modelo de classes que suporte a implementação de determinado sistema. A idéia chave do *Iconix* é fazer o menos possível, no mais curto período de tempo, de forma a concretizar um bom sistema (SILVA; VIDEIRA, 2005).

4.3 SSADM

O *Structured Systems Analysis and Design Methodology* (SSADM) foi proposto primeiramente em 1981, e teve sucessivas revisões na década de 90. Durante muito tempo foi considerada a metodologia de referência e ensinada em diversos cursos universitários. O SSADM propõe a modelagem de um sistema segundo três perspectivas: a sua funcionalidade, a sua estrutura e a sua evolução ao longo do tempo. A primeira é representada através do diagrama de fluxo de dados, a segunda é obtida através de diagramas de entidade associação e a terceira pelos diagramas de ciclos de vidas de entidades (SILVA; VIDEIRA, 2005).

A seqüência de atividades, segundo Silva e Videira (2005), envolve:

- A realização de um estudo de viabilidade, de modo a avaliar até que ponto o sistema tem custos aceitáveis.
- A análise de requisitos de negócio.
- A especificação dos mesmos requisitos.
- A especificação lógica do sistema, de modo a determinar a forma como os requisitos identificados são implementados.
- O desenho físico do sistema.

O SSADM é uma metodologia usada nas disciplinas de análise e desenho (projeto) do desenvolvimento do *software*. Porém não integra a modelagem de processo, desenvolvimento, testes e implementação. Como esse modelo foi criado para integrar apenas as disciplinas de análise e desenho, é natural observá-lo integrado com outro processo (MARTINS; SILVA, 2004).

4.4 Metodologias ágeis

As metodologias ágeis surgiram motivadas pela observação de equipes de desenvolvimento perdidas entre os processos que existiam na época. Alguns dos

ícones da indústria do desenvolvimento de *software* se uniram para encontrar valores e princípios relacionados ao desenvolvimento, que seriam capazes de fazer com que as equipes de desenvolvimento pudessem responder de forma mais ágil às mudanças nas especificações, e que o projeto fosse desenvolvido mais rapidamente.

Segundo Fowler (2001) as metodologias ágeis são uma reação a metodologias tradicionais e conceituais. As metodologias tradicionais existem há muito tempo e, em alguns casos, são muito burocráticas. Existe um grande material a ser produzido para seguir a metodologia, em conseqüência disto, a velocidade de desenvolvimento diminui. Estas novas metodologias tentam estabelecer um compromisso útil entre nenhum processo e processo exagerado, provendo apenas processo suficiente para fornecer uma boa vantagem.

O manifesto ágil, criado por um grupo de dezessete pesquisadores, destaca quatro valores: indivíduos e interações ao invés de processos e ferramentas, software funcional no lugar de documentação detalhada, colaboração do cliente ao invés de negociação de contratos e responder a mudanças no lugar de seguir um plano. Esses valores serão detalhados a seguir de acordo com Beck (1999).

A qualidade dos desenvolvedores envolvidos no projeto afeta diretamente a qualidade do produto e o bom desempenho da equipe de desenvolvimento. O fato de ter excelentes profissionais, no entanto, não é certeza de sucesso, pois estes dependem diretamente do processo. Um mau processo pode fazer com que os melhores desenvolvedores não sejam capazes de usar todo o seu talento. Além da junção certa do processo adequado com bons profissionais, é preciso que todo o grupo possa interagir perfeitamente. É importante levar em consideração que, as ferramentas utilizadas são importantes para o sucesso final do projeto, porém elas não podem ter mais importância que seus utilizadores. Mais importante que o meio onde se vai trabalhar é a qualidade e o grau de interação da equipe.

A documentação de um projeto é muito importante, visto que o código não é o melhor meio de comunicação entre os desenvolvedores. A documentação legível, para descrever o sistema, ainda se faz necessária na tomada de decisões do projeto. É preciso uma atenção especial com a documentação, pois em excesso é pior do que a sua falta. O que é sugerido pelo manifesto ágil é que somente a documentação necessária seja gerada, e esta esteja sempre sincronizada com o sistema. O fato de só existir a documentação necessária ajuda na integração da

equipe, pois a transferência de conhecimento sobre o projeto é feita trabalhando lado a lado, utilizando-se do código, sendo que este não permite duplas interpretações das funcionalidades.

O resultado eficiente de um *software* não é obtido escrevendo suas necessidades em um papel e direcionando a empresa que vai desenvolver, esperando que tudo esteja como foi solicitado no final do prazo estipulado. Projetos tratados desta maneira são falhos. As metodologias ágeis assumem que para um projeto obter sucesso e aceitação, gerando um produto de boa qualidade, é preciso que exista sempre um *feedback* do cliente para garantir que o *software* esteja sendo desenvolvido de maneira que atenda as necessidades. Contratos que determinam requisitos, prazos e custo de um projeto são, normalmente, falhos, isso porque no decorrer do desenvolvimento alguns requisitos passam a se tornar dispensáveis enquanto surge a necessidade de se adicionar outros não previstos. Portanto, o melhor contrato é aquele que determinará como será a comunicação e o relacionamento do cliente com a equipe desenvolvedora.

Assumindo que mudanças de especificação sempre vão ocorrer em todos os projetos, melhor será o projeto, se ele conseguir se adaptar a estas mudanças. A flexibilidade é fator fundamental para o sucesso do projeto, ela determina o quão adaptável ele é. Planos devem ser traçados, porém, como não é possível prever o futuro, as visões desses planos não podem ir muito longe. Muito deve ser planejado para poucas semanas, pouco se planeja para o próximo mês e quase nada se planeja para próximos anos, pois com as alterações que invariavelmente irão aparecer, muito difícil será seguir a risca estes planos de maior duração.

Os valores citados como fundamentais dentro do manifesto ágil inspirou doze princípios básicos, que são características que diferenciam as metodologias ágeis de outras metodologias chamadas tradicionais. São esses os princípios segundo Beck (1999):

- Tem como prioridade a satisfação do cliente, lançando versões em um curto espaço de tempo e continuamente.
- Aceita constantes mudanças de requisitos, ainda que tardia. Os processos ágeis mudam para a vantagem do cliente.
- Entrega de *software* funcional freqüentemente, de duas semanas a dois meses. A velocidade da entrega é de suma importância.

- Os administradores de negócios e desenvolvedores devem trabalhar em conjunto.
- A equipe de desenvolvimento deve estar motivada para o sucesso do projeto.
- O modo mais rápido de se obter informações é através de comunicação informal.
- O progresso do projeto é quantificado pelo numero de software funcional existente.
- Os processos ágeis promovem o desenvolvimento sustentável.
- Atenção contínua a excelência técnica e boa modelagem aumentam a agilidade.
- A simplicidade é essencial.
- As melhores arquiteturas, requisitos e projetos são obtidos através de equipes organizadas.
- Em intervalos regulares a equipe debate como melhorar e ajustar para obtenção de resultados satisfatórios.

Os princípios ágeis estão presentes em qualquer metodologia ágil e definem práticas a serem seguidas para tornar o desenvolvimento eficaz, enquadrando-o em sua filosofia. Eles são os guias que orientam os métodos e conseqüentemente, o desenvolvimento. As metodologias ágeis existentes atualmente orientam o desenvolvimento do *software* de acordo com suas particularidades, mas todas, na sua maior parte, seguindo o princípio ágil. Dentre elas, destacam-se *Extreme Programming* (XP), *Scrum* e *Feature Driven Development* (FDD).

4.4.1 XP

O *Extreme Programming* (Programação Extrema), ou XP, é uma metodologia ágil para equipes pequenas e médias que desenvolvem *software* com requisitos vagos e em constante mudança (BECK, 1999). É a metodologia ágil mais popular atualmente, sua utilização visa responder a sistemas onde as mudanças de requisitos são constantes, e com base em seus valores e práticas busca fazer isso da maneira mais simples e eficiente (FOWLER, 2000).

A metodologia XP foi criada por Beck (1999), que no início dos anos 90 tentava descobrir caminhos melhores para desenvolver *software*. Em 1996, começou um projeto usando novos conceitos em desenvolvimento de *software*. O resultado era a metodologia XP (WEELS, 2004).

O XP envolve todos os participantes para um trabalho de equipe com práticas simples, com *feedback* suficiente para capacitar o time a se localizar e a convergir para práticas em uma solução única. No XP, cada contribuinte do projeto é um integrante do time. Os times usam uma forma simples de planejamento e acompanhamento para decidir qual a próxima tarefa a ser realizada e antecipar quando o projeto será feito. Definidas as regras de negócio, a equipe produz o *software* em uma série de pequenas versões, que passam por todos os testes que o cliente definiu. Os programadores codificam de forma consistente para que todos possam entender e melhorar o código quando necessário (JEFFRIES, 2001). A Fig. 13 ilustra de forma geral a metodologia XP.

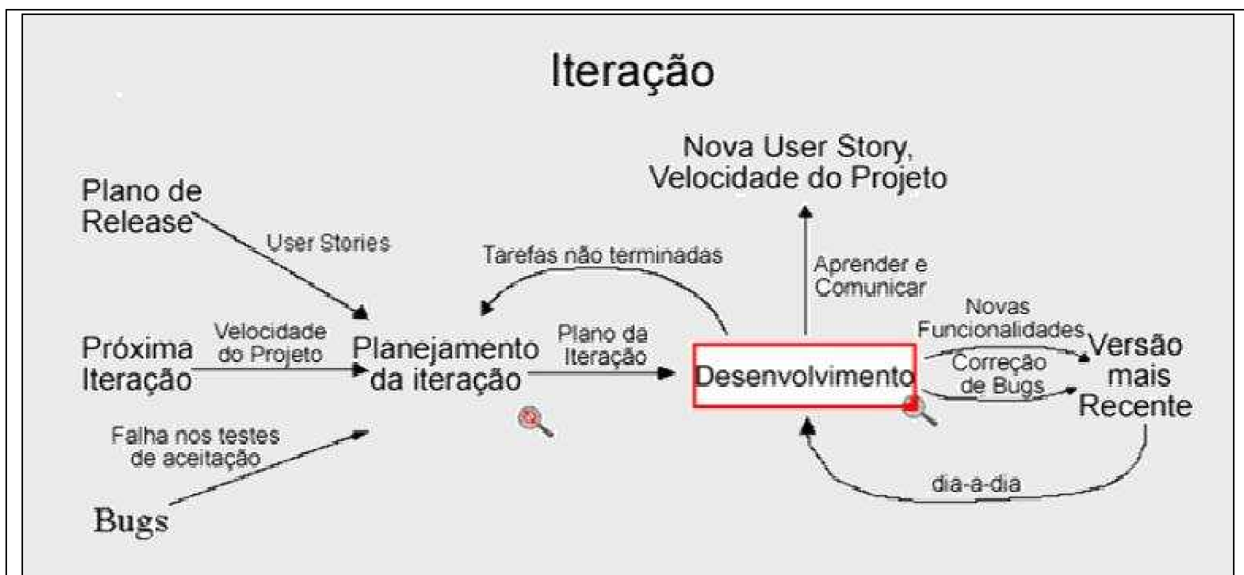


Figura 13 - Visão geral do XP.

Fonte: WEELS, 2004.

O XP segue um conjunto de valores e princípios que visam alcançar eficiência e efetividade no processo de desenvolvimento de *software*. Nos valores estão fundamentados alguns princípios básicos como: *feedback* rápido, simplicidade,

mudanças incrementais, compreensão às mudanças e qualidade do trabalho realizado. A seguir serão apresentados os valores segundo Beck (1999):

- Comunicação: comunicações freqüentes, formais ou não. Segue a linha de que o *e-mail* é melhor do que uma carta, *chat* é melhor do que um *e-mail*, uma conversa é melhor do que um *chat,* conversas em equipe são melhores que conversas individuais.
- Simplicidade: busca reduzir a complexidade do sistema e, desde que o objetivo seja alcançado, deve-se seguir a solução simplificada. Contudo, simplicidade não quer dizer facilidade, as soluções mais simples podem ser de difícil implementação. Simplicidade e comunicação estão diretamente relacionadas, pois quanto maior a comunicação, mais claro fica de se visualizar o que realmente precisa ser feito.
- *Feedback* (retorno): o constante *feedback* determina a resposta para uma ação ou prática adotada no projeto, como testes de unidade e planejamento, ajuda e agiliza na detecção de erros e nas possíveis respostas a eles, decisões são tomadas mais rapidamente, também permitindo a constante reavaliação dos requisitos. Todo problema deve ser evidenciado rapidamente para que possa ser corrigido antecipadamente.
- Coragem: a coragem é necessária para a completa realização dos outros três valores. Alterar um código funcionando para deixá-lo mais simples, a comunicação franca com o cliente, pedir auxílio quando necessário e o desligamento dos processos formais de desenvolvimento de software são exemplos de formas de coragem que devem estar presentes no projeto e desenvolvimento.

As práticas do XP apóiam umas as outras. O ponto fraco de uma é compensado pelos pontos fortes das outras. Todas as práticas descritas não são novidades, originam-se de outros processos, sendo que algumas foram abandonadas desses processos por ter custos elevados. Estas práticas ditam como e para onde ele irá, e como os valores e sua correta aplicação é fator chave para o sucesso do trabalho e, conseqüentemente, do projeto. As práticas serão descritas a seguir de acordo com Beck (1999).

Jogo de Planejamento: o cliente é quem define um plano para o projeto, definindo escopo e prazos dos *releases*. Baseado nas iterações anteriores define-se

o que será desenvolvido, bem como qual será a forma de desenvolvimento. Rapidamente determina o escopo das próximas versões combinando a prioridades do negócio e estimativas técnicas.

Pequenas Versões (*Small Realeses*): a cada pequeno ciclo de desenvolvimento é disponibilizado uma versão do sistema com as funcionalidades estipuladas pelo cliente na última iteração. A equipe de desenvolvimento coloca rapidamente um sistema simples em produção, e o atualiza freqüentemente em ciclos curtos.

Metáforas (*User Stories*): metáforas ou histórias de usuários, são descrições do funcionamento do sistema, utilizadas para auxiliar os envolvidos no projeto a ter uma visão de seu funcionamento e entender os elementos básicos do projeto e seus relacionamentos. Guiam todo o desenvolvimento e a comunicação com o cliente utilizando um sistema de nomes e uma descrição do sistema.

Projeto Simples: processos complexos devem ser removidos, bem como o uso de classes e métodos devem ser minimizados. Os testes ajudam a manter o projeto simples a cada iteração. O sistema precisar ser projetado satisfazendo os requisitos atuais de forma que não o deixe complexo.

Teste: testes unitários são combinados em conjuntos de testes, os quais devem ser rodados com freqüência, sendo que todos os testes devem aprovados com sucesso. Clientes escrevem testes de aceitação para o que foi definido para a iteração. As equipes de desenvolvimento focalizam a validação do *software* durante todo o processo. Os programadores escrevem primeiro os testes, após continuam o desenvolvimento que deve atender os requisitos destes testes.

Refatoração (*Refactoring*): aperfeiçoamento contínuo do projeto. O código é reestruturado, removendo-se duplicações e adicionando-se flexibilidade. Um projeto que no início é simples e organizado deve continuar com estas mesmas características a cada iteração. Procura-se manter a clareza do *software*, sem ambigüidade, com alta comunicação, simples, porém completo.

Programação em pares: o código do sistema é escrito por dois programadores na mesma máquina. Assim, o código é revisado pelo menos por mais um programador, tendendo a ser desenvolvido um código de melhor qualidade. Os pares devem ser formados dinamicamente, ou seja, duas pessoas que, pela manhã, formam um par, a tarde formarão par com outras pessoas.

Propriedade Coletiva: código alterado por qualquer programador a qualquer momento. Todo o código pertence a toda equipe de desenvolvimento.

Integração Contínua: o código é integrado aos diversos módulos do projeto diversas vezes. A cada integração o código deve ser testado. Este processo visa evitar problemas de integração de código, que podem acontecer quando a integração é feita, por exemplo, semanalmente.

Semana de quarenta horas: fazer horas extras durante um tempo contínuo pode ser um sinal de problema no projeto. O bem estar da equipe é fator importante para uma boa produtividade e qualidade no trabalho, pois programadores exaustos cometem mais erros.

Cliente Presente (*On-site Customer*): todos os envolvidos no projeto dividem o mesmo ambiente, formando uma equipe, incluindo o cliente. Os programadores localizam-se no centro do ambiente e a equipe de desenvolvimento tem o cliente disponível continuamente, que determina os requisitos e atribui as prioridades.

Código Padrão: os programadores escrevem código respeitando as regras que enfatizam comunicação através do código. Todos os programadores escrevem o código da mesma forma, de acordo com regras que asseguram a clareza.

O projeto ideal em XP é aquele que inicia por uma curta fase de desenvolvimento, seguida de uma longa fase de produção e refinamentos. O ciclo de vida do XP é bastante curto e, à primeira vista, difere dos padrões dos modelos de processo tradicionais. Entretanto, esta abordagem faz sentido somente em um ambiente onde as mudanças de requisitos do sistema são fatores dominantes. No caso extremo, os requisitos podem mudar no meio da versão, para atender funcionalidades mais importantes do que as definidas no planejamento original (BECK, 1999).

Um projeto XP tem seu inicio com as histórias dos usuários, onde o cliente especifica funcionalidades que pretendem que o sistema contenha, além do desenvolvimento da base da arquitetura. O escopo do projeto é definido com base nos requisitos iniciais, retirados das histórias de usuários, que também servirão para o planejamento de versões e definições de prazos, onde cliente e desenvolvedores concordam em uma data para implementação das histórias de usuário selecionadas. Essas histórias são um guia básico da metodologia XP, elas fornecem requisitos de alto nível para o sistema e são subsídios cruciais para o processo de planejamento (AMBLER, 2004).

Com base nas histórias de usuários são planejadas e definidas tarefas para cada uma delas. O foco principal do trabalho no XP, e onde se concentra o maior volume de trabalho é na construção das iteratividades, incluindo programação, testes e integração. Nesta etapa verifica-se, com maior entonação, a dinâmica da mudança de funcionalidades e requisitos. A cada iteração podem surgir novas histórias de usuários e com isso aparecerem novas tarefas (AMBLER, 2004).

No decorrer das iterações, vão sendo feitos testes de aceitação, que se aprovados pelo cliente geram uma pequena amostra do produto. Nesta etapa são feitos testes maiores, e o processo de desenvolvimento desacelera. Assim, o desenvolvimento XP prossegue, sendo todos processos repetidos várias vezes, desde as primeiras iterações até *n* iterações, à medida que vai ocorrendo o desenvolvimento da produção. Processos adicionais e orientados a produção e suporte podem ser inseridos (AMBLER, 2004).

Basicamente, o ciclo do XP é uma grande manutenção, pois essa idéia está presente em todas as etapas, com grande comunicação entre membros da equipe e no próprio projeto, pois há uma grande interatividade entre tarefas e ações. Na manutenção deve-se, simultaneamente, produzir novas funcionalidades, manter o sistema existente rodando, substituir membros do time que partem e incorporar novos membros à equipe (BECK, 1999).

Quando não mais existir novas histórias, é o momento de finalizar o projeto. É o momento de escrever algumas poucas páginas sobre a funcionalidade do sistema, um documento que auxilie no futuro como realizar alguma alteração no sistema. Uma boa razão para finalizar o projeto é o cliente estar satisfeito com o sistema e não ter mais nada que consiga prever para o futuro (BECK, 1999).

4.4.2 Scrum

O *Scrum* é uma metodologia ágil que visa confeccionar e entregar o *software* com a maior qualidade possível dentro de séries, compostas por *sprints*. Seu objetivo é fornecer um processo conveniente para projeto e desenvolvimento orientado a objeto (SCHWABER; BEEDLE, 2002). O termo *Scrum* é uma metáfora com o jogo de *rugby*, onde as equipes lutam pela posse da bola em um círculo, buscando atingir uma meta. Na busca desta meta os integrantes de cada equipe atuam em conjunto, ocorrendo freqüentes trocas de bola entre os companheiros.

Segundo Highsmith (2002), enquanto a XP focaliza a programação, o *Scrum* dá ênfase ao gerenciamento do projeto. O *Scrum*, usado principalmente para projetos de *software*, também pode ser usado para projetos sem relação com *software*, pois seus princípios são aplicados a qualquer projeto.

O *Scrum* divide o desenvolvimento em iterações (chamadas de *sprints*) de trinta dias. Equipes pequenas, de até sete pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades (requisitos) definidas no início de cada *sprint*. A equipe é responsável pelo desenvolvimento desta funcionalidade. É importante destacar que o ponto é estabilizar os requisitos durante o *sprint*. Reuniões de quinze minutos, de freqüência diária, onde o time expõe à gerência o que será feito no próximo dia, e nestas reuniões os gerentes podem levantar os fatores de impedimento (*bottlenecks*), além de avaliar o progresso geral do desenvolvimento. Esta metodologia é interessante porque fornece um mecanismo de informação de *status* que é atualizado continuamente, e porque utiliza a divisão de tarefas dentro da equipe de forma explícita (FOWLER, 2001).

Segundo Highsmith (2002), o ciclo de desenvolvimento *Scrum* não possui atividades e práticas que orientam o processo na hora do desenvolvimento (*sprint*), seu enfoque é para o controle do gerenciamento do projeto. Este ciclo pode ser dividido em três partes: pré-*sprint*, *sprint* e pós-*sprint*.

O pré-*sprint* inicia-se com a definição da lista do *backlog* de produto (lista das funcionalidades que estarão no *software*), através de uma reunião de planejamento, onde o usuário vai definir e classificar as funcionalidades desejadas para o *software* por nível de prioridade. A cada novo ciclo poderão ser alterados os itens deste *backlog*, podendo-se inserir, mudar ou excluir requisitos antes especificados. Após a definição do *backlog* do produto, organiza-se o *backlog* do *sprint* (*backlog* que será executado durante o ciclo do *sprint*). Nesta etapa serão definidos, a partir do *backlog* do produto, quais atividades serão realizadas no *sprint*. Definido o trabalho que será realizado no ciclo do *sprint*, parte-se para o projeto e desenvolvimento, fase que dura em média trinta dias.

A parte de *sprint* é a fase de desenvolvimento. Uma vez iniciada só os desenvolvedores envolvidos poderão fazer mudanças nas atividades especificadas. Estas mudanças, feitas pela equipe de desenvolvimento, são alterações e expansões nas tarefas definidas no *backlog* de *sprint* e visam a melhor adaptação

ao trabalho a ser realizado. O objetivo do *sprint* é completar as tarefas definidas para ele e entregar uma pequena parte funcional do *software*, também conhecida como incremento. Para tanto a equipe é livre para desenvolver, desde que os objetivos sejam alcançados.

Para fechar o ciclo (pós-*sprint*) é apresentado ao cliente o produto da iteração, ou seja, o pedaço executável de *software* ou incremento é analisado pelos usuários. Este incremento deverá estar testado e com as funcionalidades especificadas para ele. Também deverá ter a sua documentação, que será entregue ao cliente. Nesta fase também é analisado o progresso do projeto. Nela são gerados gráficos de monitoramento, que indicam o sucesso do projeto.

A Fig. 14 reproduz graficamente o ciclo do *Scrum*. Os pontos 1, 2 e 3 formam o pré-*sprint* e representam, respectivamente, a definição e classificação das funcionalidades, definição das funcionalidades e tarefas que serão realizadas durante o *sprint*, classificação e possível expansão das tarefas pela equipe de desenvolvimento. Os pontos 4 e 5 formam o *sprint* e representam, respectivamente, o ciclo de desenvolvimento onde serão realizadas as tarefas especificadas no *backlog* do *sprint* e a reunião diária onde a equipe debate sobre o andamento do projeto. Finalizando, o ponto 6 forma o pós-*sprint* e representa a entrega de uma funcionalidade demonstrável ao cliente.

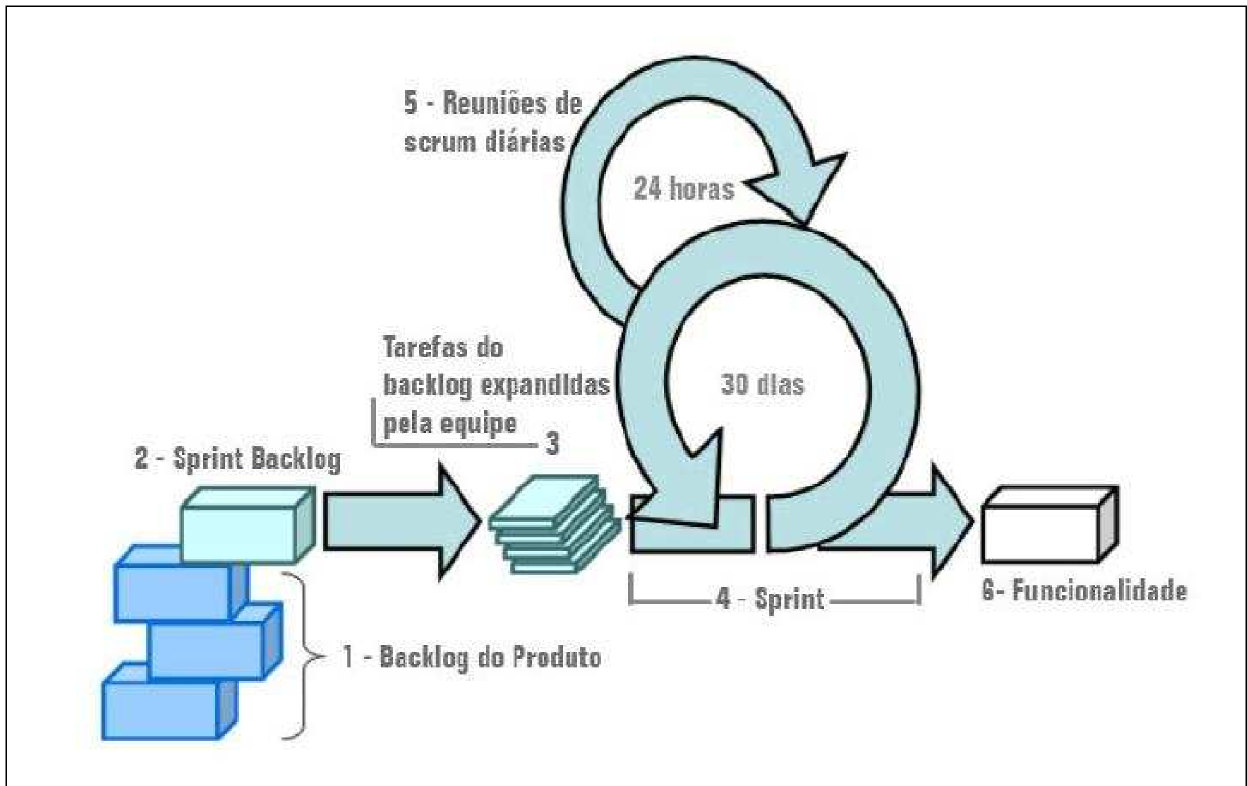


Figura 14 - Ciclo do *Scrum*

Fonte: WWW.CONTROLCHAOS.COM

Atualmente, a metodologia de desenvolvimento *Scrum* tem rapidamente adquirido reconhecimento como uma ferramenta eficaz para desenvolvimento produtivo de *software*. Mesmo quando o *Scrum* é utilizado de forma conjunta com outros padrões de projetos existentes, ele é altamente adaptável, ainda que em organizações de desenvolvimento de *software* bem estruturadas (SUTHERLAND, 2000).

4.4.3 FDD

O *Feature Driven Development* (Desenvolvimento Guiado por Funcionalidades) é uma metodologia ágil para gerenciamento e desenvolvimento de *software*. Essa metodologia de desenvolvimento é dirigida a características, que são funcionalidades indicadas pelos clientes que tenham algum valor para o *software*. O FDD nasceu a partir da experiência de análise e modelagem orientadas por objetos e do gerenciamento de projetos (COAD, 1999).

Tanto desenvolvedores quanto gerentes e clientes são focalizados pelo FDD. Segundo Highsmith (2002), no FDD os desenvolvedores finalizam e ganham trabalho em intervalos de duas semanas, gerentes têm uma medida do plano, incluindo marcos expressivos e freqüente redução de riscos, retornando resultados tangíveis, e clientes conseguem acompanhar e entender o projeto. Estas seriam características que fariam do FDD um processo bem visto por todas as partes envolvidas.

Segundo Highsmith (2002), o FDD tem algumas características peculiares como:

- Resultados úteis a cada duas semanas ou menos.
- Blocos pequenos de funcionalidade de grande valorização pelo cliente, chamados *features*.
- Planejamento detalhado.
- Rastreabilidade e relatórios com precisão.
- Monitoramento detalhado dentro do projeto, com resumos de alto nível para clientes e gerentes, tudo em termos de negócio.
- Fornece uma forma de saber, no início de um projeto, se o plano e a estimativa são sólidos.

O FDD é uma metodologia muito objetiva. Possui apenas duas fases que se dividem em concepção e planejamento (pensar um pouco antes de fazer), e construção (fazer de forma iterativa). A Fig. 15 ilustra a estrutura do FDD.

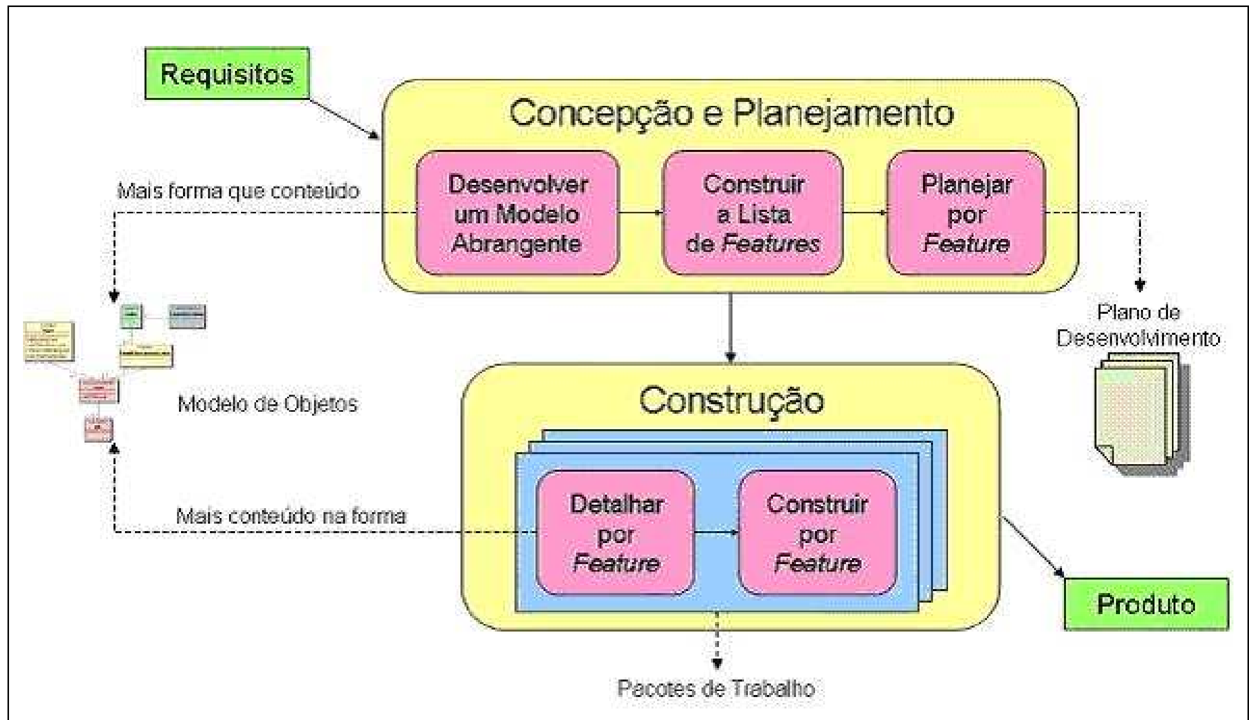


Figura 15 - Estrutura do FDD
Fonte: WWW.HEPTAGON.COM.BR

O FDD possui cinco processos executados em ordem seqüencial, que orientam sua evolução e desenvolvimento. Estes processos são executados em ciclos de no máximo duas semanas e devem ter como resultado a implementação de uma característica (funcionalidade) previamente definida. Esses processos serão detalhados a seguir, segundo Highsmith (2002).

Desenvolver um modelo abrangente: consiste em desenvolver modelos de áreas específicas do negócio. Estes modelos são unidos formando um modelo final. Para que o processo chegue ao fim deverá ser aprovado e, também, deverá conter as classes do domínio, seu relacionamento e seus atributos. As tarefas envolvidas neste processo incluem a formação de um time de modelagem, um guia do domínio, documentos de estudo, o desenvolvimento do modelo, o refinamento do modelo de objeto e descrição de notas do modelo.

Construir uma lista de funcionalidades: uma equipe formada pelos programadores-chefe define todas as características que o *software* deverá conter. Estas características são organizadas de forma hierárquica e sua classificação é definida por prioridades e tamanho. Entre as tarefas incluem-se a formação da

equipe da lista de características, a identificação das mesmas, sua classificação e decomposição, se necessário.

Planejar por funcionalidade: o gerente de projeto, o gerente de desenvolvimento e os programadores-chefe fazem um plano de projeto, que será usado nos processos seguintes, e determinará a seqüência de desenvolvimento e as datas que cada característica deverá estar completada. Esta seqüência é definida pelas ligações existentes entre as características e sua complexidade. Entre as tarefas deste processo incluem-se a formação da equipe de planejamento, a definição da seqüência das características e a nomeação das classes para seus proprietários.

Detalhar por funcionalidade: o programador-chefe define um grupo de características, identifica as classes que estarão envolvidas e contata os seus respectivos proprietários. Cada característica selecionada irá passar por esta etapa, onde a equipe de características define um diagrama de seqüência detalhado para elas. Os proprietários de classes estruturam suas classes e métodos. Por fim a equipe faz uma inspeção no projeto. Entre as tarefas deste processo incluem-se a formação da equipe de projeto e de um guia de domínio, a construção do diagrama de seqüência, a estruturação das classes e métodos e a inspeção do projeto.

Construir por funcionalidade: a equipe de características faz a implementação das classes e métodos, a inspeção do código, os testes de unidade e fazem o desenvolvimento, para cada característica ou conjunto delas. Entre as tarefas deste processo incluem-se a implementação das classes e métodos, a inspeção do código, os testes de unidade e a verificação e construção da tarefa.

Em suma o funcionamento básico do FDD é dado seguindo os processos descritos anteriormente, de forma seqüencial até a etapa de planejamento por funcionalidade, sendo as duas últimas etapas repetidas até a finalização das funcionalidades. Primeiramente, é desenvolvido um modelo abrangente do projeto, para dar a todos os envolvidos uma visão global, facilitando assim o seu entendimento. Depois de completado este processo, segue-se para a construção da lista de características, definidas, segundo Coad (1999), como funcionalidades com valor para o cliente.

Depois de definida a lista de funcionalidades é feita o plano de características, que irá determinar datas e a seqüência de desenvolvimento. Por fim são executadas as etapas de projeto e construção das funcionalidades. Cada

característica do conjunto atual de características vai passar por cada uma destas últimas etapas durante o ciclo, sendo que é nestes dois processos onde geralmente se desprende o maior tempo, pois são repetidos para cada característica durante o ciclo (COAD, 1999).

5 ANÁLISE DOS PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Após apresentar, individualmente, cada uma das principais metodologias de desenvolvimento de *software*, descrevendo as suas características, funcionalidades e todas as suas fases, das mais tradicionais até as mais ágeis, neste capítulo será apresentada uma análise dos processos (metodologias) de forma caracterizá-los e adaptá-los ao desenvolvimento distribuído.

Primeiramente, serão feitos alguns comparativos entre as metodologias apresentadas no capítulo 4 juntamente com uma análise crítica. A análise qualitativa foi feita através de duas técnicas, uma delas é a análise de conteúdo qualitativa, que visa analisar o conteúdo de maneira sistemática, por meio de um sistema de categorias, desenvolvido a partir do material e guiado por teoria, e a outra técnica é a *grounded theory*, que parte da suposição que na análise, durante a coleta dos dados, já desenvolve, aprimora e interliga conceitos teóricos e hipóteses, de tal maneira que o levantamento e a análise se sobrepõem (MAYRING, 2002). Posteriormente, as principais dificuldades e os problemas com o desenvolvimento distribuído de *software,* vistos no capítulo 3, serão detalhados e analisados neste capítulo com o objetivo de que as características dos processos possam se adaptar a essas dificuldades, encontrando soluções cabíveis para minimizar tais problemas.

5.1 Comparação e análise crítica dos processos

Um desenvolvimento de *software* que resulte em um produto de boa qualidade depende de alguns fatores como recursos, custos e pessoas. As metodologias de desenvolvimento visam estabelecer uma melhor organização do trabalho e integração destes fatores, através da adoção de práticas e modelos, como papéis e características, implementados segundo filosofias pertinentes a cada metodologia, de acordo com o que foi apresentado no capítulo 4.

As metodologias tradicionais consistem no desenvolvimento através de regras definidas para componentes e requisitos previamente especificados. Estas metodologias focalizam e priorizam o processo, que deve ser seguido em sua

totalidade para cada projeto. As metodologias ágeis têm como principal característica o desenvolvimento focado na mudança de requisitos, priorizando pessoas em vez de processos, estas metodologias baseiam-se menos em padrões, visando a eficácia das funcionalidades do *software* e a satisfação dos clientes. Também procuram deixar os projetos mais adaptáveis, dando espaço para redefinições dos requisitos durante o seu andamento.

A Fig. 16 mostra um comparativo entre as metodologias tradicionais e ágeis, demonstrando o seu comportamento em relação a desenvolvedores, clientes, requisitos, arquitetura, tamanho e objetivo. Nota-se uma maior diferença na área de requisitos e no objetivo principal, explicitando grande diversidade entre as filosofias de cada tipo de metodologia. Equipes pequenas, colaboração dedicada do cliente e retorno rápido são outras características pertinentes às metodologias ágeis e que divergem das características das metodologias tradicionais.

Área	**Metodologias Tradicionais**	**Metodologias Ágeis**
Desenvolvedores	Orientados ao plano Habilidades adequadas Acesso ao conhecimento externo	Ágeis Colaborativos
Clientes	Colaborativos Representativos Com poder no projeto	Colaborativos Dedicados Com poder no projeto
Requisitos	Estáveis	Emergentes Rápidas Mudanças
Arquitetura	Projetada para os requisitos previstos	Projetada para os requisitos
Tamanho	Equipes e produtos grandes	Equipes e produtos pequenos
Principal Objetivo	Alta confiança	Retorno rápido

Figura 16 - Comparativo entre as metodologias tradicionais e ágeis.

Fonte: Desenvolvido pelo autor

Definir qual dessas abordagens de desenvolvimento é mais eficiente é uma tarefa demasiadamente relativa, pois nenhum projeto é igual ao outro e depende de qual o tipo e o objetivo do projeto. Uma avaliação empírica não se faz necessária, visto que não está no contexto e se distancia do objetivo deste trabalho. Uma das formas de avaliação é através do estabelecimento de itens que servirão de critério

para as comparações, levando-se em conta os fatores de desenvolvimento citados acima.

Cada metodologia possui características que determinam a forma de funcionamento e desenvolvimento do trabalho para cada projeto, como foram vistas no capítulo 4. Estas características podem determinar o sucesso ou fracasso de um projeto, devendo ser bem analisadas para a escolha de determinado processo. Outro ponto a se destacar são as deficiências apresentadas pelas metodologias, que também influenciam no resultado final de um projeto. As falhas de um processo podem dificultar sua utilização a determinado projeto, podendo ser um fator decisivo na sua escolha.

A Fig. 17 mostra um comparativo das metodologias apresentadas neste trabalho. Este comparativo apresenta como itens de critério: pontos-chave, principais características e as falhas de cada metodologia.

Metodologia	Pontos-chave	Principais Características	Falhas
RUP	Processo de desenvolvimento completo. Inclui suporte a ferramentas.	Modelo de negócios e suporte a ferramentas.	Burocrático
Iconix	Desenvolvimento dirigido a casos de uso.	Pequeno, prático e simples. Utiliza diagramas e técnicas da UML.	Não faz modelagem de processos de negócios.
SSADM	Faz a modelagem do sistema segundo a funcionalidade, a estrutura e a sua evolução.	Usada para análise e projeto.	Não faz modelagem de processo, desenvolvimento, teste e implementação.
XP	Desenvolvimento dirigido pelo cliente. Equipes pequenas e construções diárias.	Refatoração, o redesenho do sistema melhora o desempenho e é responsável pelas mudanças.	Práticas de gerenciamento têm pouca atenção.
Scrum	Desenvolvimento por equipes com ciclos de trinta dias. Independente, pequeno e auto-organizável.	Metodologia focada no gerenciamento de processos.	Falta de testes de aceitação e integração.
FDD	Cinco processos base. Orientado a objetos e iterações curtas.	É uma metodologia simples. Faz desenvolvimento por funcionalidades e modelagem de objeto.	Foco somente no projeto e implementação, precisa de outros suportes.

Figura 17 - Análise das metodologias de desenvolvimento de software.

Fonte: Desenvolvido pelo autor.

Destaca-se neste comparativo a diversidade de pontos e características apresentados, como prova disto não existem metodologias com pontos-chave idênticos. Cada uma possui a sua particularidade, tanto nos pontos-chave e características, como nas falhas. Estas particularidades servem de guia para a escolha de um determinado processo.

Um fator a ser salientado é a possibilidade de integração das metodologias. A falha de uma pode ser característica de outra, suprindo, com isso, as suas deficiências. Um exemplo desta integração é a do *Scrum* com o XP.Uma das falhas na metodologia XP é a pouca atenção às práticas de gerenciamento, justamente uma característica do *Scrum*.

O SSADM é uma metodologia incompleta, obrigatoriamente, precisa ser integrada com outras que suprem as suas falhas. O RUP é uma metodologia completa de desenvolvimento, ou seja, possui todas as etapas do processo de *software*, como definição dos requisitos, análise, projeto e implementação.

O *Scrum* é dentre as metodologias ágeis a mais voltada para o gerenciamento de um projeto como um todo, ou seja, ela define estratégias em um âmbito global para um projeto, visando a organização deste. Suas práticas e papéis não especificam atividades para o desenvolvimento e sim para o seu processo organizacional, fazendo com que seja indicado para projetos com deficiência neste setor. Ao utilizar o *Scrum* um projeto não terá tarefas específicas definidas, como a programação em pares do XP, mas o trabalho será focado em seu ciclo organizacional, como as etapas que o compõe.

O *Iconix* fica em um meio termo entre as metodologias tradicionais e as ágeis, pois é bem completo, porém compacto. Já a FDD é a mais simples e direta entre as metodologias ágeis apresentadas.

As metodologias ágeis atuais têm suas limitações e vantagens aparentes, como a falta de suporte adequado a projetos grandes e a adaptabilidade às mudanças, respectivamente, como foi apresentado na seção 4.4. Processos tradicionais são estruturados em etapas definidas e não nasceram com o foco na mudança de requisitos. Comparar os dois seguindo características próprias a ambos pode não transparecer o real conceito de agilidade de cada um.

Em relação à agilidade dos processos, a Tab. 1 mostra um status de agilidade das metodologias ágeis apresentadas neste trabalho em relação ao RUP, que é a metodologia tradicional mais completa. Esta análise quantitativa possui

como base alguns itens-guia genéricos que auxiliaram na avaliação de cada metodologia, de acordo com a definição de Highsmith (2002), as notas vão de 1 a 5, quanto maior a nota, mais ágil é o processo.

Tabela 1 - Comparativo de agilidade entre o RUP e as metodologias ágeis.

Agilidade	**RUP**	**XP**	**Scrum**	**FDD**
Resultados	3	5	5	4
Simplicidade	2	5	5	5
Adaptatividade	3	3	4	3
Técnica	4	4	3	4
Práticas	3	5	4	3
Média	**3**	**4.4**	**4.2**	**3.8**

Fonte: Adaptado de HIGHSMITH, 2002.

É notável a vantagem das metodologias ágeis nesta comparação, pois nasceram para atingir metas de agilidade e têm sua estrutura voltada para este objetivo. Portanto, esta comparação só é válida para demonstrar e conceituar a diferença entre estes tipos de abordagem, esclarecendo pontos decisivos para a escolha de um ou outro.

A seguir é apresentada uma comparação que mostra as disciplinas das metodologias tradicionais em relação ao XP, que é a metodologia ágil mais utilizada atualmente, conforme a Fig. 18.

	Modelagem de Negócio	**Requisitos**	**Análise**	**Projeto**	**Implementação**	**Teste**	**Instalação**	**Gestão de Alterações**	**Gestão de Projeto**	**Gestão do Ambiente**
RUP	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Iconix		**X**	**X**	**X**	**X**	**X**	**X**		**X**	
SSADM		**X**	**X**	**X**						
XP	**X**			**X**	**X**	**X**			**X**	

Figura 18 - Comparativo das disciplinas entre as metodologias tradicionais e o XP.

Fonte: Adaptado de MARTINS e SILVA, 2004.

As metodologias analisadas apresentam disciplinas em comum, porém algumas são mais completas, pois as exigências dos projetos a que se destinam requerem outros tipos de atividades. Sendo o RUP um dos processos mais completos é natural que inclui uma maior variedade de disciplinas. O SSADM, por ser uma metodologia que visa apenas à análise e o projeto, é a que possui menos disciplinas.

Detalhando melhor alguns aspectos importantes dentro do desenvolvimento do *software*, cabe agora uma análise e comparação das duas principais e mais utilizadas metodologias dentre as analisadas: o RUP e o XP.Os aspectos discutidos são: alocação de tempo e esforço, artefatos, atividades, disciplinas e papéis. A Fig. 19 ilustra este comparativo.

Área	RUP	XP
Alocação de tempo e esforço	Curto tempo de iteração para projetos pequenos e longo tempo de iteração para projetos grandes. Perde um tempo maior nas duas primeiras fases.	Curto tempo de ciclos e releases para projetos pequenos e não cobre projetos grandes. Entrega de releases mais rapidamente e com maior freqüência.
Artefatos	Grande número de artefatos, mas a maioria dos documentos são gerados quando existe uma grande necessidade.	Poucos artefatos.
Atividades e disciplinas	Cada atividade está relacionada com uma disciplina, com o objetivo de trabalhar cada artefato.	Quatro atividades básicas.
Papéis	Especifica melhor e com um maior número.	Poucos e mais abrangentes.

Figura 19 - Comparativo entre o RUP e o XP.

Fonte: Desenvolvido pelo autor.

A alocação de tempo e esforço precisa de uma atenção especial. Projetos muito longos com uma equipe pequena podem causar vários problemas.Os projetos dentro do limite padrão do XP tornam visíveis que as iterações do RUP coincidem com os *releases* do XP, em relação a duração dos projetos. Com projetos de grande porte, onde os padrões estão fora dos limites especificados pelo XP, estas semelhanças não ocorrem. Esses projetos, geralmente, têm como característica a grande duração de suas iterações, por isso, são direcionados pelo RUP.Neste tipo de projeto, as equipes são divididas para que se possa desenvolver paralelamente.

Normalmente, cada equipe ficará com uma parte do projeto, sendo que cada uma dessas partes, geralmente, se encaixam nos padrões do XP, porém como é necessário que o projeto seja tratado como um todo, o escopo foge os limites do XP.Na metodologia XP, é necessário uma análise preliminar do escopo do projeto, antes dos ciclos e *releases* começarem, para definir se ele pode ocorrer de fato. Determinar esforço, custo e tempo tem a mesma razão que a fase de concepção no RUP.Para o início de um projeto é necessária essa análise, porém, o que se difere nessas duas metodologias é que, enquanto no XP, isso deve ser feito de maneira rápida e objetiva, o RUP, entretanto, considera que esta fase levará o tempo que for necessário, pois é feita a análise de risco, fator determinante no processo.

A metodologia XP não especifica um desenvolvimento em longo prazo, pois considera que apenas as funcionalidades previstas para iteração que está sendo realizada devem ser planejadas. Já o RUP, normalmente utilizado em grandes projetos e baseado na gerência de riscos, não pode deixar este ponto de arquitetura e infra-estrutura sem planejamento adequado, pois refazer código pode levar mais tempo que o previsto, considerando o grau de complexidade do projeto.

Em relação ao ciclo de vida de ambas as metodologias, é possível afirmar que o XP entrega os *releases* mais rapidamente e com uma maior freqüência, exatamente como é citado nas práticas da metodologia, pois usa um pequeno tempo nas fases de exploração e planejamento, o RUP nas duas primeiras fases, já gasta um tempo maior. Esta razão se deve ao fato de que o RUP faz uma análise de risco, que consome algumas iterações na fase de elaboração, garantindo que a arquitetura desenvolvida não seja falha, impedindo que no futuro seja necessária alguma alteração que resulte em trabalho dobrado e perdas de tempo muito grandes.

No que diz respeito aos artefatos, uma das principais características do RUP, criticada pelos adeptos do XP, é o excesso de documentos e de burocracia que a metodologia tradicional gera. Na realidade, muitos deles não precisam ser gerados, sendo apenas escritos quando existe uma grande necessidade, seguindo o princípio ágil. O motivo de especificar com vários documentos o projeto é deixar o processo mais objetivo e claro, e também com metas bem definidas. Uma das razões para as diferenças entre as documentações e artefatos é que as abordagens têm escopos diferentes. Enquanto o RUP pode ser usado, com alguns ajustes e adaptações, em todos os tipos de projetos, o XP tem seu foco nos projetos pequenos, que acabam não necessitando de uma maior documentação. Outro

possível motivo é a forma como o XP trata a etapa inicial de busca e investigação dos requisitos. Enquanto esses requisitos são coletados através das histórias de usuários, no RUP isso é feito através de entrevistas e questionários, que serão transformados em outros artefatos no decorrer do processo. Para projetos pequenos o RUP gera poucos documentos, assim como o XP.

As atividades, no processo de desenvolvimento, são as tarefas exercidas por um determinado papel, utilizando e modificando documentos, bem como produzindo novos artefatos. No RUP, cada atividade dessas está relacionada com uma disciplina (*workflow*). Essas atividades têm como objetivo trabalhar cada artefato, fazendo com que eles se tornem menos abstratos e mais focados nas metas do projeto, auxiliando a determinar variáveis como custo, tempo e esforço. Esses artefatos servirão de base no decorrer do projeto. No XP, por ter uma visão mais simplificada do projeto, existem apenas quatro atividades básicas, onde cada atividade utiliza técnicas para atingir melhores resultados no desenvolvimento do projeto. Essa forma de abordagem, que o XP trata as atividades, acaba se assemelhando com as disciplinas do RUP.

Os papéis desempenham as atividades do processo, e também tem a responsabilidade de elaborar e alterar alguns determinados artefatos, que são gerados pelas atividades que estes papéis desempenham. Tanto no RUP quanto no XP os papéis podem ser exercidos por uma única pessoa da equipe ou por um grupo, sendo que a mesma pessoa, ou grupo, pode realizar mais de um papel. O que mais diferencia as metodologias, neste quesito, é a quantidade de papéis apresentados, sendo que o RUP apresenta uma quantidade maior de papeis do que o XP.O RUP define melhor cada um desses papéis, sendo que isso facilita quando se procura uma pessoa com determinado perfil para assumir um papel, especializando melhor as funções de quem vai se adaptar a este papel. No XP, os poucos papéis que existem, são mais gerais, englobando vários papéis dentro do RUP.

5.2 Análise do desenvolvimento distribuído de software

Após a comparação e a análise crítica dos processos de uma forma geral, nesta parte do trabalho serão inseridos os aspectos do desenvolvimento distribuído de *software*. Serão identificados e descritos os aspectos não técnicos e técnicos do

desenvolvimento distribuído e as características dos processos apresentados com esses aspectos. Os aspectos não técnicos do desenvolvimento distribuído cobrem, principalmente, tudo que se refere às pessoas envolvidas no projeto e as suas interações e questões físicas que não envolvem o projeto em si, mas que possam interferir no seu andamento. Os aspectos técnicos dizem respeito as mais diversas atividades dentro do desenvolvimento, diretamente relacionado com o produto e o seu processo.

A Fig. 20 mostra uma análise dos aspectos do desenvolvimento distribuído com as metodologias apresentadas, indicando qual metodologia cobre ou prevêem as necessidades de cada área do desenvolvimento distribuído, indicando um *sim,* com eventuais observações. Quando a metodologia não cobre de forma integral, ou não prevê no seu processo, a palavra *não* foi utilizada.

Área	**RUP**	**Iconix**	**SSADM**	**XP**	**Scrum**	**FDD**
Diferenças sociais	Sim, de forma indireta.	Sim, de forma indireta.	Sim, de forma indireta.	Não	Não	Não
Diferenças culturais	Não	Não	Não	Não	Não	Não
Dispersão geográfica e temporal	Sim	Sim, desde que haja um suporte maior.	Não	Não	Sim, desde que haja um suporte maior.	Sim, desde que haja um suporte maior.
Padronização do desenvolvimento	Sim	Sim	Sim	Sim	Sim	Sim
Gerência de Projeto	Sim	Sim	Não	Sim	Sim	Sim
Complexidade e tamanho do projeto	Sim	Sim	Sim	Não	Não	Não
Tecnologia e infra-estrutura de comunicação e informação	Sim, com adaptações.	Não	Não	Não	Não	Não
Gestão do Ambiente físico de desenvolvimento	Sim, com adaptações.	Não	Não	Não	Não	Não

Figura 20 - Análise entre os aspectos do desenvolvimento distribuído e as metodologias.

Fonte: Desenvolvido pelo autor.

Os seguintes aspectos não técnicos presentes no comparativo serão detalhados a seguir: Diferenças sociais, Diferenças culturais e Dispersão geográfica

e temporal. Os aspectos técnicos são os seguintes: Padronização do desenvolvimento, Gerência de projeto, Complexidade e tamanho do projeto, Tecnologia e infra-estrutura de comunicação e informação e Gestão do ambiente físico de desenvolvimento.

5.2.1 Diferenças sociais

O desenvolvimento distribuído, como já foi apresentado no capítulo 3, possui uma instância que é o desenvolvimento global de *software*. Nesse tipo de desenvolvimento as questões sociais são de grande importância, principalmente no que se refere à confiança entre as equipes envolvidas. Para que um projeto tenha um desenvolvimento eficiente, é necessário que os integrantes localizados em ambientes diferentes tenham confiança entre os participantes e também no projeto. O inicio da confiança entre a equipe pode começar a partir de um processo de desenvolvimento mais tradicional e seguro. Nesse aspecto as metodologias tradicionais ganham destaque, pelo fato de possuírem características que asseguram maior tranqüilidade e confiança no decorrer do trabalho. Por exemplo, o RUP e o *Iconix* têm características que proporcionam maior tranqüilidade e segurança para que o projeto transcorra tranqüilamente, de forma indireta no desenvolvimento.

5.2.2 Diferenças culturais

No desenvolvimento global de *software*, existem muitas diferenças culturais entre os mais diversos países que possam estar inseridos no projeto. As diferenças culturais podem influenciar em diversas decisões dentro do projeto devido a determinadas tradições e costumes de cada país, conforme apresentado na seção 3.2. Em termos de idioma torna-se mais fácil a resolução devido a globalização da língua inglesa. Nenhum dos processos de desenvolvimento de *software* analisados gerencia as diferenças culturais, portanto, necessariamente, deve haver uma extensão ou adaptação da metodologia escolhida, de forma que as diferenças culturais sejam minimizadas ao máximo para que não interfiram no resultado final do produto.

5.2.3 Dispersão geográfica e temporal

A dispersão geográfica e temporal pode ser um problema se não for administrada da melhor forma possível. Devido à distância e a diferença de fusos horários quem possam existir, a comunicação, interação e troca de informações entre as equipes de desenvolvimento devem acontecer através de um processo de desenvolvimento que disponibilize ferramentas (como por exemplo, baseadas em *web*, *framework* e controle de versões), e juntamente com um suporte, para que o projeto consiga atingir os objetivos previstos. Dependendo da metodologia adotada, pequenas modificações ou apenas extensões habilitam as metodologias tradicionais a cobrirem esses problemas. As metodologias ágeis precisariam adotar meios mais formais de comunicação para poder se adaptar com essa forma de trabalho. O XP, por exemplo, não prevê meios formais de comunicação e interação da equipe, o que acaba inviabilizando o projeto nesse aspecto.

5.2.4 Padronização do desenvolvimento

A utilização de uma metodologia de desenvolvimento de *software* em projetos distribuídos é imprescindível e fundamental. O não uso pode provocar diferentes modos de trabalho e desenvolvimento, ocasionando dificuldade na integração das diversas partes do projeto e um resultado final diferente do esperado. A padronização pode ser feita adotando qualquer uma das metodologias de desenvolvimento, dando preferência para aquelas que cobrem o maior número de atividades e que possam também trazer resultados satisfatórios. A adaptação do processo escolhido se faz necessária visto que nenhum cobre todos os aspectos de forma integral. O RUP e o *Iconix* são os mais completos para a padronização de todo processo de desenvolvimento de *software*.

5.2.5 Gerência de projeto

No desenvolvimento distribuído de *software* a gerência de projeto, no que diz respeito à coordenação e controle, se torna extremamente difícil. A integração entre as diversas partes e módulos do projeto deve acontecer de modo eficiente para que o fato de estarem longe fisicamente não interfira. Novamente as metodologias que

prevêem esse tipo de gerenciamento e ferramentas devem ser valorizadas. O *Scrum* focaliza bastante o gerenciamento dos processos, mas não cobre alguns testes e a integração, tornando a sua escolha direcionada para projetos mais genéricos. O RUP, por ser mais completo e por fazer uma gerência bem abrangente, deve ser o preferido nesse aspecto.

5.2.6 Complexidade e tamanho do projeto

Normalmente, projetos com desenvolvimento distribuído de *software* são complexos e de grande porte, devido ao fato do investimento inicial neste tipo de projeto ser elevado, e podendo envolver diversos países. O inverso também ocorre, ou seja, atualmente projetos complexos e grandes, normalmente, são distribuídos, pelas facilidades e ganhos já descritos no capítulo 3. A utilização de metodologias tradicionais e mais completas é necessária nesse aspecto, principalmente, por melhor gerenciar equipes grandes e abranger as mais diversas áreas do desenvolvimento, como por exemplo, o RUP.As metodologias ágeis, como o XP e o FDD, são para equipes pequenas, dificultando a utilização quando a complexidade e o tamanho do projeto aumentam.

5.2.7 Tecnologia e infra-estrutura de comunicação e informação

A comunicação, em todos os seus sentidos, é um dos grandes problemas do desenvolvimento distribuído. É preciso que haja meios eficientes que ultrapassem as barreiras impostas pelo desenvolvimento em locais diferentes. Como foi visto na seção 3.2, projetos com desenvolvimento distribuído empobrecem a comunicação, pois em projetos no mesmo local muitas questões são discutidas informalmente, facilitando a resolução dos problemas. As metodologias ágeis sofrem grande desvantagem neste aspecto, pois a comunicação informal é um dos grandes princípios destas metodologias, o que acaba inviabilizando esse tipo de processo em relação a um melhor gerenciamento da comunicação. O RUP com algumas adaptações e melhorias, como a criação de uma disciplina de desenvolvimento distribuído, acaba tornando-se a melhor alternativa.

5.2.8 Gestão do ambiente físico de desenvolvimento

As dificuldades na gestão do ambiente físico podem causar problemas na integração de sistemas devido ao planejamento mal elaborado. Em um ambiente distribuído o uso de recursos e ferramentas deve ser gerenciado de modo que não atrase nem prejudique a integração. A padronização de todos os instrumentos de trabalho é necessária, juntamente com um suporte técnico adequado. O RUP, por exemplo, prevê um gerenciamento do ambiente de desenvolvimento, mas precisa de adaptações e acréscimos, direcionando para o desenvolvimento distribuído. As demais metodologias apresentadas não prevêem este tipo de gerenciamento.

6 ADAPTAÇÃO DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE PARA DESENVOLVIMENTO DISTRIBUÍDO

A falta de um processo padrão específico para o desenvolvimento distribuído de *software* pode acarretar dificuldades durante o decorrer do projeto. Os estudos para desenvolver essa área estão crescendo, mas ainda não existem padrões mundialmente aceitos. A adoção de um processo, já existente, adaptado para esse tipo de desenvolvimento é uma opção que pode se tornar eficiente.

Após a análise das metodologias de desenvolvimento de *software*, feita no capítulo 5, o RUP foi identificado como a metodologia que melhor se adapta ao desenvolvimento distribuído e cobre grande parte de suas necessidades, embora precise de algumas modificações. Essas adaptações e especificações serão detalhadas no decorrer deste capítulo. A adaptação do RUP é baseada nos conceitos teóricos apresentados na seção 2.5, juntamente com o modelo genérico de melhoria dos processos proposto por Sommerville (2003). A criação da disciplina de desenvolvimento distribuído deste trabalho é de acordo com os padrões das demais disciplinas que o RUP especifica. Os diagramas, aqui apresentados, foram feitos através da ferramenta *JUDE UML Modeling Tool*, versão 3.2.1.

6.1 Adaptação do RUP para uso em ambientes distribuídos

Nos tempos atuais, é naturalmente aceito adaptações em processos para os mais diversos tipos de projeto. As adaptações prevêem excluir, modificar ou adicionar novos elementos no processo, como visto no capítulo 2. Como nenhum dos processos estudados cobre todas as necessidades do desenvolvimento distribuído a metodologia escolhida foi a que se aproxima mais do resultado esperado.

O RUP foi a metodologia selecionada, principalmente, pela sua facilidade de adaptação e por ter completado o maior numero de requisitos necessários para um bom desenvolvimento em locais dispersos. Os diversos aspectos do desenvolvimento distribuído direcionaram a escolha do RUP.

Entre os aspectos do desenvolvimento distribuído, as diferenças sociais (confiança no projeto e entre as equipes de desenvolvimento) são fatores importantes e que influenciam durante o decorrer do projeto. O RUP é a metodologia mais tradicional, e ao longo dos anos, padronizou vários projetos importantes. Um processo seguro e completo gera mais tranqüilidade, minimizando assim possíveis desconfianças que possam existir.

A complexidade e a grande extensão dos projetos com desenvolvimento distribuído fez do RUP a opção para a adaptação também neste aspecto, pois é, entre os analisados, o único que foi projetado para grandes equipes e projetos de grande porte. A gerência do projeto também favorece a escolha do RUP, as metodologias que enfatizam a gerência não cobrem outros aspectos que o RUP especifica, facilitando assim a integração das diversas partes do projeto.

A Gerência de Projeto do RUP é uma disciplina muito importante, pois engloba o gerenciamento de riscos, planejamento e o acompanhamento do projeto. Além disso, pode dispor de uma atenção especial para a comunicação e as trocas de informações, que são fundamentais para o bom desenvolvimento de forma distribuída. Outra disciplina importante é a de Ambiente, pois define como o RUP foi configurado para ser utilizado no projeto, além da organização do ambiente de trabalho para toda a equipe de desenvolvimento, facilitando assim novas formas de desenvolvimento e os mais variados tipos de projeto.

O RUP foi a metodologia escolhida, mas ele não cobre todos os aspectos abordados no capítulo 5. Através da análise foi identificado que o processo precisa ser acrescido de um gerenciamento de culturas diferentes, gerenciamento do ambiente físico de desenvolvimento e um aprimoramento das tecnologias e infra-estrutura de comunicação e informação. A Fig. 21 ilustra como acontece o acréscimo destas melhorias.

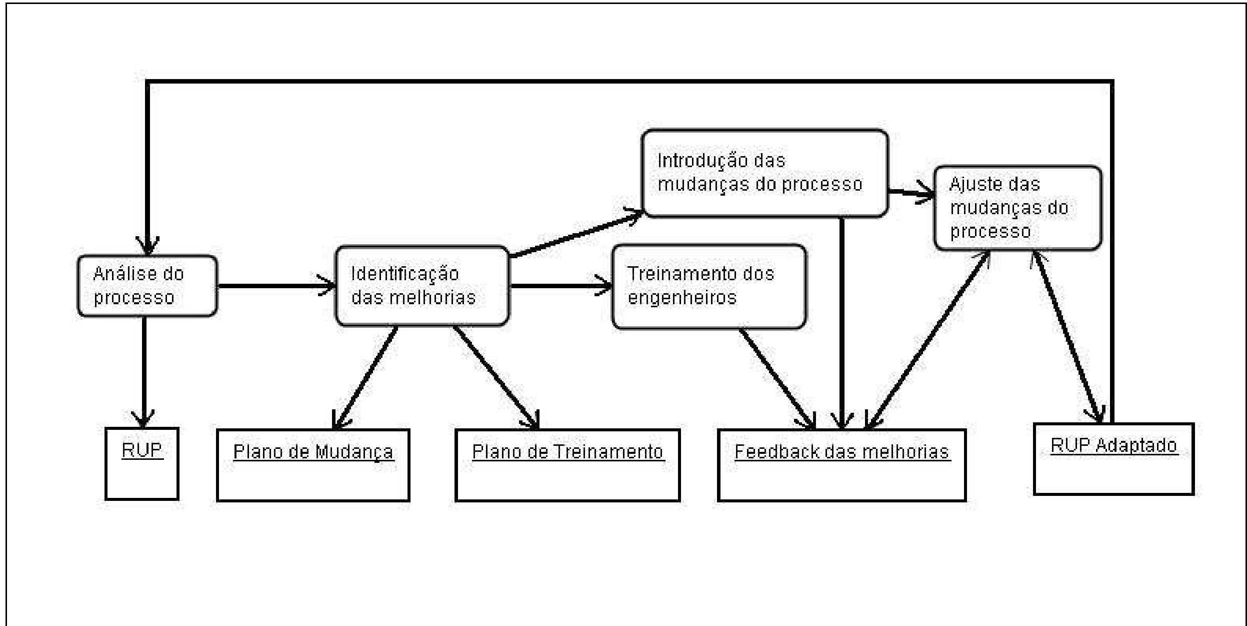


Figura 21 - Modelo de Adaptação do Processo RUP.

Fonte: Adaptado de SOMMERVILLE, 2003.

A partir da análise do capítulo 5, vários aspectos e características foram observados. Para um projeto tradicional, o RUP não precisaria de nenhuma adaptação, de acordo com as necessidades previstas. Para projetos em ambientes distribuídos, é preciso percorrer todo o processo de adaptação para chegar a um resultado satisfatório. Através das análises qualitativa e quantitativa, bons resultados são gerados, ocorrendo o acréscimo de informações extras ao modelo de processo.

Na identificação das melhorias, que são necessárias para o desenvolvimento distribuído, são utilizados os resultados da análise dos processos para averiguar os fatores que vão influenciar no resultado final do projeto. As melhorias do processo podem propor novas disciplinas e atividades para um melhor desempenho do projeto. As identificações das melhorias devem ser documentadas com os seus devidos planos de adaptação (descrição detalhada de todo processo de adaptação) e de treinamento (especificação dos passos do treinamento para toda equipe de desenvolvimento).

A introdução de adaptações no processo significa implantar novas disciplinas, atividades, ferramentas, e integrá-las com outras atividades já existentes no processo. É importante dar tempo suficiente para introduzir as melhorias e garantir que elas sejam compatíveis com outras atividades de processo e com os procedimentos e padrões organizacionais.

Sem um devido treinamento das adaptações do processo, não é possível obter os plenos benefícios das mudanças do processo. Elas podem ser rejeitadas pelos gerentes e engenheiros responsáveis pelos projetos de desenvolvimento. É muito comum que as adaptações de processo sejam impostas sem treinamento adequado e que os efeitos dessas mudanças resultem na degradação e não na melhoria da qualidade do produto.

As adaptações de processo propostas nunca serão inteiramente eficazes assim que forem introduzidas. Há a necessidade de uma fase de ajuste, em que os problemas menores sejam descobertos e modificações no processo sejam propostas e introduzidas. Essa fase de ajuste pode durar bastante tempo, até que os engenheiros de desenvolvimento estejam satisfeitos com o novo processo. Para facilitar a fase de ajuste é utilizada a documentação gerada com o retorno desta própria fase, juntamente com o retorno da fase de introdução e treinamento das adaptações. O resultado final de todo esse processo é o RUP adaptado para o desenvolvimento distribuído.

6.2 Especificação da disciplina de gerenciamento do desenvolvimento distribuído

Como parte integrante da adaptação, após a devida explicação dos passos para adaptar o RUP em ambientes distribuídos, nesta fase final do trabalho foi acrescida uma nova disciplina no RUP para complementar o desenvolvimento, com a finalidade de atingir bons resultados no desenvolvimento distribuído. A nova disciplina proposta é a de Gerenciamento do Desenvolvimento Distribuído, que possui atividades não previstas no processo normal de desenvolvimento do RUP.A Fig. 22 mostra o diagrama de atividades com o fluxo de trabalho do Gerenciamento do Desenvolvimento Distribuído. O diagrama começa pelo estado inicial, seguido da atividade de Planejar Ambiente de Desenvolvimento Distribuído, após essa atividade ocorrem as atividades de Gerenciar Culturas Diferentes, Gerenciar Ambiente Físico de Desenvolvimento e Aprimorar as tecnologias e infra-estrutura de comunicação e informação, que são executadas de forma independente e concorrente. A última atividade é a de Finalizar Projeto Distribuído, encerrando no estado final. É importante ressaltar que as demais disciplinas propostas pelo RUP devem ser seguidas normalmente. Além da implementação desta nova disciplina deve ocorrer

uma atenção especial na disciplina de Requisitos, que é de fundamental importância dentro do desenvolvimento distribuído.

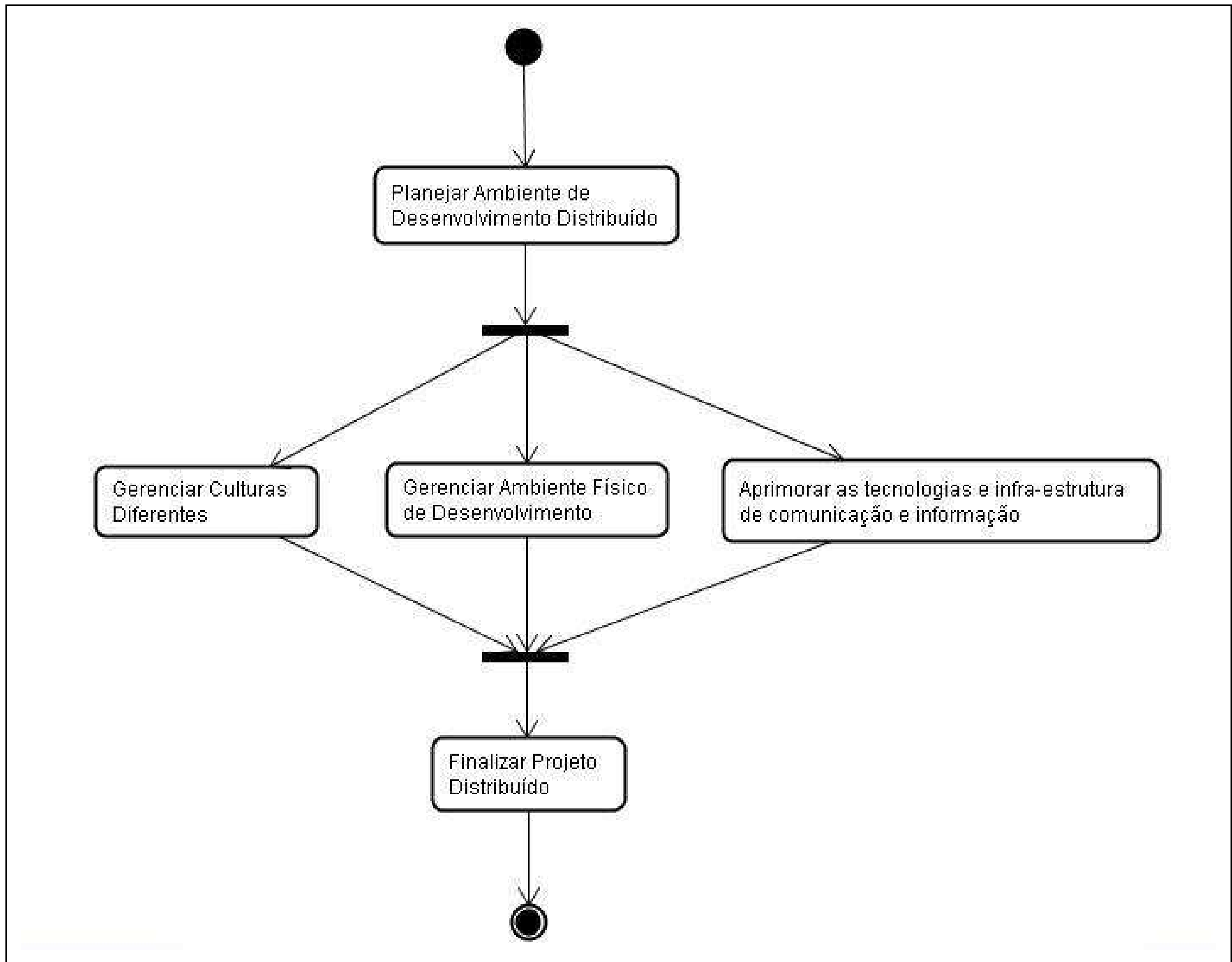


Figura 22 - Diagrama de atividades do fluxo de trabalho da disciplina de gerenciamento do desenvolvimento distribuído.

Fonte: Desenvolvido pelo autor.

6.2.1 Planejar ambiente de desenvolvimento distribuído

A atividade de Planejar Ambiente de Desenvolvimento Distribuído tem a característica importante de avaliar a organização atual e descrever como desenvolver o processo. Tem como finalidade principal descrever o status atual da organização de *software* em termos de processos atuais, ferramentas, competências e atitudes das pessoas, clientes, concorrentes, tendências técnicas, problemas e áreas de melhoria. Possui os seguintes passos:

- Iniciar uma Avaliação: é recomendado que a avaliação inicie com um *workshop*, através de vídeo conferência, com as pessoas importantes dentro do projeto. A principal finalidade desse *workshop* virtual é fazer com que os engenheiros de processo conheçam todos os envolvidos no projeto para que possam fazer uma lista abrangente dos problemas a partir de suas perspectivas.
- Identificar os Envolvidos: identificação dos envolvidos que estão fora da organização, como por exemplo, os clientes, os concorrentes, e possíveis outros envolvidos. É também necessária, uma identificação dos envolvidos que estão dentro da organização de desenvolvimento, como por exemplo, os membros do projeto, gerente, *marketing* e pessoas de outros departamentos de desenvolvimento.
- Descrever a Organização Interna: descrição da organização interna, especificamente os papéis e as equipes atuais. Além de considerar o relacionamento entre as diferentes partes da organização de desenvolvimento.
- Identificar Competências: fazer uma lista das áreas de competência relevantes. Para cada uma dessas áreas, avaliar o conhecimento, a habilidade e a experiência das pessoas da organização.
- Definir Escopo: definir quais serão as disciplinas abordadas no caso de desenvolvimento, verificando os aspectos da avaliação da organização.
- Mapear Papéis para Cargos: o mapeamento de cargos para papéis pode facilitar o entendimento do pessoal da organização sobre como empregar o RUP e ajudá-los a entender que papéis e cargos não são iguais.
- Analisar a Descrição do Processo de Desenvolvimento: analisar as descrições existentes do processo para entender a profundidade do processo de desenvolvimento em vigor. Para cada disciplina, identificar os tipos de descrição existentes.
- Caracterizar o Projeto e o Aplicativo: descrever as características dos projetos e aplicativos típicos da organização.
- Identificar as Ferramentas de Suporte: identificar as ferramentas que estão sendo usadas pelos projetos. Identificar as áreas nas quais os membros do projeto carecem de ferramentas de suporte.

- Identificar Possíveis Problemas: reunir diversas pessoas-chave para uma sessão de identificação de problemas, não deixando de cobrir todas as áreas do processo de desenvolvimento. Identificar os efeitos negativos que cada problema tem e identificar as causas originais de cada problema para entender melhor como eliminá-lo ou reduzi-lo.
- Fazer Recomendações: é necessária como parte da avaliação e deve descrever como o projeto deve avançar para implementar o novo processo e as novas ferramentas.

Todas estas atividades são realizadas pelo Engenheiro de Processo e gera como artefato a avaliação da organização de desenvolvimento e o caso de desenvolvimento.

6.2.2 Gerenciar culturas diferentes

A atividade de Gerenciar Culturas Diferentes tem como característica principal tratar os diversos aspectos que envolvem a interferência da diversidade das culturas dos países no resultado final do projeto. Tem como finalidade desenvolver um plano de projeto com países de culturas diferentes. Possui os seguintes passos:

- Levantamento de Características: fazer um levantamento prévio das características dos países envolvidos no projeto, com suas formas e costumes de desenvolvimento.
- Pesquisar Formas de Desenvolvimento: fazer uma pesquisa de formas de integração dos projetos entre os países envolvidos.
- Interação Prévia entre os Envolvidos: integrar os países antes do desenvolvimento do projeto, com palestras, vídeo conferências e seminários.
- Gerenciar troca de informações: gerenciar todas as trocas de informações entre as equipes de desenvolvimento dispersas, para que não haja problemas de interpretação e informação errada.
- Manter histórico de informações: manter um histórico de informações através da ferramenta escolhida para utilização em projetos futuros.

As atividades são realizadas pelo Gerente de Projeto e gera como artefato o plano de culturas diferentes.

6.2.3 Gerenciar ambiente físico de desenvolvimento

A atividade de Gerenciar Ambiente Físico de Desenvolvimento tem como característica principal padronizar e gerenciar os ambientes de desenvolvimento distribuído. Tem como finalidade projetar ambientes físicos padronizados, juntamente com as ferramentas de desenvolvimento. Possui os seguintes passos:

- Projetar Ambientes Padronizados: projetar ambientes físicos iguais de acordo com o número de integrantes da equipe de desenvolvimento.
- Adaptar Ambientes já Construídos: adaptar os ambientes já construídos para que se aproximem do padrão recomendado.
- Padronizar Materiais de Suporte: padronizar os mais diversos materiais de suporte como micro-computadores, impressoras e materiais de escritório.
- Padronizar Ferramentas: padronizar as ferramentas de desenvolvimento de software.
- Organizar e Manter o Ambiente Físico: a organização facilita um ambiente mais favorável para um bom desenvolvimento, necessitando que seja mantido durante todos os projetos.

As atividades são realizadas pelo Gerente de Projeto e gera como artefato a padronização dos ambientes físicos de desenvolvimento.

6.2.4 Aprimorar as tecnologias e infra-estrutura de comunicação e informação

A atividade de aprimorar as tecnologias e infra-estrutura de comunicação e informação tem como característica principal buscar as melhores ferramentas e melhorar cada vez mais as formas de comunicação e informação entre as equipes de desenvolvimento distribuído. Tem como finalidade especificar, selecionar, adquirir e manter as melhores ferramentas para comunicação e troca de informações. Possui os seguintes passos:

- Identificar Necessidades e Restrições: identificar quais são as necessidades de ferramentas para comunicação, bem como o seu suporte, e quais as restrições, observando os aspectos técnicos do desenvolvimento.

- Coletar Informações sobre Ferramentas: coletar informações sobre as sugestões de ferramentas e seus respectivos fornecedores. Algumas dessas informações são dados que podem ser coletados com o fornecedor ou nas revisões.
- Especificar Ferramentas: criar uma tabela de características e funções de ferramentas de comunicação e troca de informações e criar critérios envolvendo as informações que a ferramenta deve ter de acordo com o seu fornecedor. Também é necessário avaliar os custos associados à aquisição da ferramenta.
- Comparar Ferramentas: combinar fatores e selecionar as melhores ferramentas baseando-se em uma tabela de características.
- Selecionar Ferramentas: selecionar a ferramenta que melhor atende os aspectos de comunicação e informação e que se adapte as restrições do projeto.
- Adquirir Ferramentas: a aquisição da ferramenta deve considerar os seguintes aspectos: assistência na instalação, o suporte oferecido, o comprometimento do fornecedor, a manutenção, o treinamento, a evolução do produto e a licença.
- Pesquisar as Melhores Formas de Comunicação e Informação: manter uma constante pesquisa das melhores formas de comunicação e troca de informações para aprimorar cada vez mais a ferramenta para o próximo projeto distribuído.

Todas estas atividades são realizadas pelo Especialista em Ferramentas e geram como artefato a nova ferramenta proposta e o plano de aprimoramento da ferramenta.

6.2.5 Finalizar projeto distribuído

A atividade de Finalizar Projeto Distribuído tem como características principais a revisão do projeto e preparar a finalização do projeto. Tem como finalidade revisar o projeto e a aceitação formal dos artefatos liberados, bem como concluir as formalidades associadas à aceitação e à finalização do projeto. Possui os seguintes passos:

- Programar Reunião de Revisão de Projeto: após a identificação dos participantes da reunião, definir data e hora para a reunião, podendo ocorrer de forma virtual, com um tempo de antecedência suficiente para que os participantes possam revisar o material que será utilizado para a decisão de aprovação.
- Administrar Reunião de Revisão de Projeto: os participantes revisam os resultados dos testes e das revisões de aceitação, usando os critérios de aceitação devidamente documentados. No final da reunião, os revisores devem tomar a decisão de aprovação. Se algum critério de aceitação não foi atendido, o cliente poderá aceitar o projeto somente se determinadas ações corretivas forem executadas. Se o projeto não for aceito, isso significa que não atende aos critérios de aceitação e requer trabalho adicional, sendo necessário executar outro ciclo de aceitação do projeto.
- Registrar Decisão: no final da reunião, um registro de revisão é concluído, capturando todas as discussões ou itens de ação importantes e registrando o resultado da revisão da aceitação do projeto. Se o resultado não foi aceito, uma nova reunião de revisão da aceitação do projeto deverá ser programada para uma data posterior.
- Atualizar Plano de Finalização do Projeto e Programar Atividades: deve garantir que uma programação formal para atividades de finalização de projeto distribuído seja criado e acordado com o cliente e a própria equipe do projeto. Essa programação deve ser capturada no plano de desenvolvimento do *software*.
- Programar Auditoria de Configuração Final: organizar as auditorias de configuração funcional e física de modo que elas sejam conduzidas de acordo com a realização da auditoria de configuração.
- Finalizar o Projeto Distribuído: o gerente do projeto executa as tarefas administrativas restantes de finalização do projeto distribuído como: garantir que o projeto será formalmente aceito, fazer o levantamento das finanças do projeto, arquivar toda a documentação e registros do projeto e reatribuir as tarefas do pessoal que continua no projeto.

Estas atividades são realizadas pelo Gerente de Projeto e pelo Revisor do Projeto, gerando como artefatos o registro de revisão, a avaliação de *status*, plano de desenvolvimento distribuído e a lista de problemas.

7 CONCLUSÃO

O correto desenvolvimento de *software*, independente de qual metodologia de desenvolvimento se utilize, terá sempre como foco a qualidade final do produto e a satisfação do cliente. Para chegar a uma excelência nos processos de desenvolvimento de *software*, muita pesquisa precisa ser feita, pois os diversos tipos de projeto requerem um desenvolvimento especial, e dificilmente um processo trará satisfação total para a equipe de desenvolvimento. Esses aspectos refletem um dos vários motivos para o contínuo estudo dos processos de desenvolvimento de *software*.

O estudo sobre desenvolvimento distribuído de *software* mostrou a necessidade de uma atenção especial para este tipo de desenvolvimento, pois os projetos em ambientes dispersos apresentam diferenças e dificuldades em relação aos projetos tradicionais. A aplicação de processos de desenvolvimento torna-se necessária, mas ainda não existem padrões reconhecidos mundialmente, portanto, a criação ou adaptação de um processo é fundamental.

As metodologias apresentadas (RUP, *Iconix*, SSADM, XP, *Srum* e FDD) se diferenciam nos mais diversos aspectos de desenvolvimento, principalmente confrontando as principais características das mais tradicionais, como o RUP, com as ágeis, como o XP.Essas metodologias foram criadas para os mais diversos propósitos e para tipos de projetos diferentes. Para o uso em determinados tipos de projetos uma análise precisa ser feita, de forma a contribuir para uma eficiente escolha.

A análise dos processos realizada neste trabalho auxilia a escolha de qual melhor metodologia aplicar em desenvolvimento distribuído, contendo as informações necessárias para tal escolha, juntamente com comparativos e comentários. A análise inicial realizada também é útil para escolher uma metodologia para qualquer tipo de projeto, de acordo com as necessidades e especificações de um determinado projeto.

Neste trabalho, a avaliação, através da análise, indicou a escolha do RUP como a melhor metodologia, dentre as apresentadas, para o desenvolvimento

distribuído. Mesmo sendo considerado um processo de desenvolvimento completo, a adaptação do RUP foi realizada devido a não abrangência total das necessidades do desenvolvimento distribuído. Seguindo a seqüência de passos proposta, a adaptação do RUP para ambientes distribuídos tem grandes chances de torna-se extremamente satisfatória. A inclusão da disciplina de Gerenciamento do Desenvolvimento Distribuído se faz necessária como parte integrante da adaptação para o desenvolvimento distribuído, incluindo as atividades de Planejar Ambiente de Desenvolvimento Distribuído, Gerenciar Culturas Diferentes, Gerenciar Ambiente Físico de Desenvolvimento, Aprimorar as tecnologias e infra-estrutura de comunicação e informação, e por fim, Finalizar Projeto Distribuído.

Com a conclusão deste trabalho, novas perspectivas e projeções para trabalhos futuros surgiram. A complementação deste trabalho, a partir dos resultados obtidos, pode ser feita através dos seguintes acréscimos e recomendações:

- Maior especificação e detalhamento da disciplina de Gerenciamento do Desenvolvimento Distribuído, com a inclusão de mais passos e de forma mais detalhada com diagramas UML.
- Acréscimo de aspectos importantes das metodologias ágeis na disciplina proposta, de forma a tornar um modelo híbrido, podendo ser adaptado em outros processos.
- Aplicação da análise e da adaptação do RUP para o desenvolvimento distribuído em empresas ou grupos de pesquisa, de forma a fazer um teste empírico do trabalho, ou seja, baseado na experiência.

Este trabalho serve como base para que outras pesquisas e estudos possam ser realizados, tanto para aplicação de determinada metodologia, como para o uso de processos no desenvolvimento distribuído de *software.* Além da contribuição do acréscimo de uma nova disciplina proposta para utilização específica em ambientes distribuídos. O propósito dos trabalhos futuros é deixar o trabalho mais abrangente e com um potencial maior de sua aplicação.

REFERÊNCIAS

AMBLER, S. **Modelagem ágil:** práticas eficazes para programação extrema e o processo unificado. Porto Alegre: Bookman, 2004.

BECK, K. **Extreme programming explained:** embrace change. Upper Saddle River: Addison-Wesley, 1999.

BENCOMO, A. **Extending the RUP:** Part 1: process modeling, 2005. Disponível em: <http://www-128.ibm.com/developerworks/rational/library/05/323_extrup1>. Acesso em: 11 dez 2006.

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, Universidade Federal de Santa Catarina, Florianópolis.

BOOCH, G., RUMBAUGH, J.; JACOBSON, I. **UML**: Guia do Usuário. São Paulo: Campus, 2006.

CARASIK, R.; GRANTHAM, P.Constructing real-time collaborative software engineering tools using CAISE, an architecture for supporting tool development. **ACM International Conference Proceeding Series**, v.171; Proceedings of the 29th Australasian Computer Science Conference, v.48, p.267-276, Hobart, Australia, 2006.

CARMEL, E. **Global Software Teams:** collaborating across borders and time-zones. [s.l.]: Prentice Hall, 1999. 269p.

COAD, P.; LEFREBVE, E.; DE LUCA, J. **Java modeling in color with UML**. [s.l.]: Prentice Hall. 1999.

CYRILLO, L.C. **GESPRODS:** um modelo de gestão de projetos distribuídos de software. 2005. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo.

EVARISTO, R.; FENEMA, P.C.V. A typology of project management: emergence and evolution of new forms. **International Journal of Project Management**, v.17, n.5, p.275, 1999.

FOWLER, M. **The new methodology.** 2001. Disponível em: <http://www.martinfowler.com/articles/newmethodology.html>. Acesso em: 8 dez 2006.

GINSBERG, M. P., QUINN, L. H. **Process tailoring and the software capability maturity model.** [s.l.]: Technical Report, November 1995.

HAYWOOD, M. **Managing virtual teams:** pratical techique for high-tecnology project managers. Boston: Artech House, 1998. 199p.

HERBSLEB, J. D.; MOITRA, D. **Global software development**. Los Alamitos: IEEE Software, March/April, EUA, 2001. p.16-20.

HIGHSMITH, Jim. **Agile software development ecosystems**. Upper Saddle River: Addison-Wesley, 2002.

INTERNATIONAL WORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT (DISD 2005), 29 August 2005 co-located with the 13th IEEE Requirements Engineering Conference 2005. Disponível em: <http://www.infosys.tuwien.ac.at/Staff/sd/DiSD2005/DiSD-2005.html>. Acesso em: 5 dez 2006.

ISO / IEC TR 15504. **Information technology:** software process assessment. [s.l.]: Technical Report, 1998.

JACOBSON, I., BOOCH G., RUMBAUGH J. **The unified software development process.** Upper Saddle River: Addison Wesley, 2001.

JEFFRIES, R. **XP magazine contents**: what is Extreme Programming?, 2001. Disponível em: <http://www.xprogramming.com/xpmag/index.htm>. Acesso em: 11 dez. 2006.

KAROLAK, D. W. **Global software development:** managing virtual teams and environments. Los Alamitos: IEEE Computer Society, EUA, 1998, 159p.

KIEL, L. Experiences in distributed development: a case study. In: Workshop on Global Software Development at ICSE, **Proceedings...** Oregon, EUA, 2003, 4p.

KRUCHTEN, P.**The rational unified process:** an introduction. 2nd Edition, Upper Saddle River: Addison-Wesley, 2000.

MARQUARDT, M. J., HORVATH, L. **Global teams:** how top multinationals span boundaries and cultures with high-speed teamwork. Palo Alto: Davies-Black, 2001.

MARTINS, P., SILVA, A. Comparação de metamodelos de processos de desenvolvimento de software. In: CONFERÊNCIA PARA A QUALIDADE NAS TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÕES, 5., 2004. **Anais da...** [s.l.]: Instituto Português de Qualidade, 2004. p.179-186.

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

OMG (Object Management Group). **Software process engineering metamodel specification:** version 1.1. 2005. Disponível em: <http://www.omg.org/technology/documents/formal/spem>. Acesso em: 11 dez 2006.

OPPENHEIMER, H. L. Project Management Issues in Globally Distributed Development**.** In: WORKSHOP ON GLOBAL SOFTWARE DEVELOPMENT (ICSE 2002), **Proceedings…** 24., Florida, 2002.

PEREIRA, E.B. **Uma proposta para adaptação de processos de desenvolvimento de software baseados no rational unified process.** 2005. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação da Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

PRESSMAN, R.S. **Software engineering:** a practitioner’s approach. 5 ed. McGraw-Hill: EUA, 2001.

PRIKLADNICKi, R.; AUDY, J. MuNDDoS: um modelo de referência para desenvolvimento distribuído de software. In: SIMPÓSIO BRASILEIRO ENGENHARIA DE SOFTWARE, 18., 2004, Brasília. **Anais do...** Brasília 2004. p.289-304.

ROSENBERG, D.; SCOTT, K. **Use case driven object modeling with UML:** a practical approach. Massachusetts: Addison-Wesley Longman, 1999.

SCHWABER, K.; BEEDLE, M. **Agile software development with SCRUM**. [s.l.]: Prentice Hall, 2002.

SCHWARTZ, J.I. Construction of software. In: **Practical strategies for developing large systems**. Menlo Park: Addison-Wesley, 1975.

SCOTT, K. **O processo unificado explicado**. Porto Alegre: Bookman, 2003.

SILVA, A.; VIDEIRA, C. **UML:** metodologias e ferramentas CASE. 2.ed. Lisboa: Centro Atlântico, 2005. 578 p.

SOMMERVILLE, I. **Software engineering**. 6.ed. Menlo Park: Addison Wesley, 2003.

SUTHERLAND, J. **SCRUM software development process.** 2000. Disponível em: <http://jeffsutherland.com/scrum/index.html>. Acesso em: 19 dez. 2006.

WELLS, D.. **Extreme programming**: a gentle introduction. 2004. Disponível em: <http://www.extremeprogramming.org>. Acesso em: 11 dez. 2006.

XU, P., RAMESH, B. Knowledge Support in Software Process Tailoring**.** In: **Proceedings of the 38th Hawaii International Conference on System Sciences** (HICSS), 2005.

ZESAR, K. D. et al. Performance and quality aspects of virtual software enterprises. In: EUROMICRO CONFERENCE (EUROMICRO 1998), 24, Sweden, 1998. **Proceedings**… IEEE Computer Society, 1998. p.20824-20829.