# Capítulo 1

# Processos Tradicionais de Desenvolvimento de Software

Wislayne Aires Moreira[1]

Este capítulo aborda os processos tradicionais de desenvolvimento de software. Neste contexto, alguns desses processos mais utilizados serão apresentados ao longo do capítulo, tais como: o RUP, um dos mais difundidos; o OpenUp e o MSF e Agile MSF; focando em suas origens, características, arquitetura, etapas, disciplinas e modelo de ciclo de vida.

## 1.1 Introdução

Os processos de desenvolvimento de software[2] tradicionais, também conhecidos por processos orientados a documentos, devido ao rigor na geração da documentação associada ao desenvolvimento de software [Boehm 1988], apresentam as seguintes características: abordagem voltada à documentação detalhada da execução das atividades; possuem fases sequenciais, com um conjunto de artefatos para cada fase; e dão ênfase à definição rigorosa de papéis para a execução do trabalho.

Muitas pessoas acham que os processos tradicionais de desenvolvimento de software são inviáveis por exigirem muitos recursos e tempo para a execução das tarefas, e que substituí-los por processos ágeis (Veja Capítulo 2) é a única opção para se obter um produto de qualidade. O fato é que tanto os processos tradicionais como os ágeis têm sua aplicabilidade em contextos específicos. Enquanto os processos tradicionais são mais aplicáveis a projetos de maior duração e com grandes equipes de desenvolvimento, os processos ágeis são mais aplicáveis a projetos de menor duração com equipes pequenas. Isso acontece, porque com o aumento do projeto, a comunicação face a face torna-se mais difícil. Por isso, os métodos ágeis são mais adequados para projetos com no máximo 20 a 40 pessoas.

Este capítulo dará ênfase aos processos tradicionais, enquanto o próximo capítulo abordará os processos ágeis. Dentre os vários processos tradicionais existentes, este capítulo irá abordar o RUP (*Rational Unified Process*), o OpenUp (*Open Unified Process*) e o MSF (*Microsoft Solution Framework*) por serem exemplos representativos destes tipos de processo.

---

[1] wislayne@gmail.com

[2] Uma sequência de passos necessários para o desenvolvimento ou manutenção de um software.

## 1.2 RUP

O RUP é um processo de engenharia de software que foi criado pela *Rational Software Corporation*, adquirida posteriormente pela IBM, e que descreve atividades a serem seguidas pelos integrantes da equipe que desenvolve o software.

O RUP tem suas origens no ano de 1995, quando a *Rational Software Corporation* comprou a empresa *Objectory AB*. Como resultado desta aquisição, foi criado em 1996 o *Rational Objectory Process* (ROP) 4.0, a partir da integração entre o *Rational Approach* (um processo de desenvolvimento de software interno utilizado pela Rational) e o *Objectory Process* 3.8 (um processo criado por Ivar Jacobson como resultado de sua experiência na Ericsson e posteriormente na sua empresa, a *Objectory AB*). Neste mesmo ano, James Rumbaugh e Grady Booch se uniram a Ivar Jacobson, formando o grupo conhecido como "os três amigos", os quais foram responsáveis pela integração dos métodos *OMT* e *Booch* ao processo até então existente, dando origem ao *Rational Objectory Process* 4.1, o qual foi lançado em 1997. Em 1998, o *Rational Unified Process* (RUP) 5.0 foi liberado. Esta versão é o ROP 4.1 renomeado, estendido com material de processos obtidos de outras companhias compradas pela *Rational Corporation*, bem como material desenvolvido pelo grupo liderado por Phillipe Kruchten. No ano de 1999, o RUP 5.5 foi lançado com o objetivo de abordar o desenvolvimento de sistemas de tempo real e de sistemas baseados na Web. Posteriormente, em 2000, o RUP 2000 foi lançado com várias melhorias, como adição de técnicas de engenharia de negócios para a disciplina de modelagem de negócios. Em 2002 a IBM adquiriu a Rational e em 2003 foi lançada uma nova versão do RUP aprimorando a disciplina de testes e incorporando alguns guias para se usar o RUP segundo a abordagem ágil. A Figura 1.1 mostra a evolução do RUP ao longo do tempo.

### 1.2.1 O RUP e suas características

O RUP é um processo de desenvolvimento iterativo e incremental. Iterativo porque o produto é desenvolvido em várias iterações similares e incremental porque, em cada iteração, o produto é estendido com mais funcionalidades. Os elementos centrais do RUP são [DANTAS, 2008]:

- É um processo de engenharia de software bem definido e estruturado que fornece uma estrutura customizável. Ou seja, o RUP suporta a customização de modo que uma vasta variedade de processos, ou configurações de processos, podem ser montadas;
- Segue uma abordagem de desenvolvimento de software iterativa e incremental, centrada na definição de uma arquitetura robusta, que facilita a paralelização do desenvolvimento, reutilização e a manutenção; e em casos de uso baseados em risco;
- Utiliza UML como notação para modelagem.

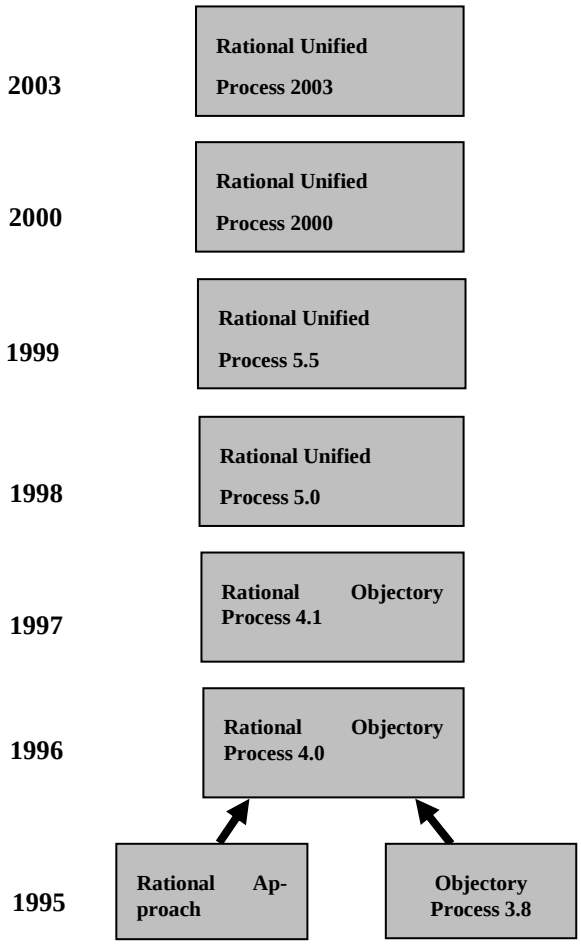


**Figura 1.1 Evolução do RUP, adaptado de [AMBLER, 2010]**

#### 1.2.1.1 Princípios básicos do RUP

Os seis princípios mais importantes do RUP são [PISKE, 2003]:

**a)Desenvolver software iterativamente**

Permite entregar um sistema completo e aumenta a funcionalidade de cada subsistema a cada release.

**b)Gerenciar requisitos**

O RUP descreve como documentar as funcionalidades, restrições de sistema, restrições de projeto e requisitos de negócio. Os casos de uso são exemplos de artefatos que são muito eficazes na captura de requisitos funcionais.

**c)Usar arquiteturas baseadas em componentes**

A arquitetura baseada em componentes permite a criação de um sistema que pode ser facilmente extensível. O RUP oferece uma forma metódica para construir esse tipo de sistema, focando em produzir uma arquitetura executável nas fases iniciais do projeto, ou seja, antes de comprometer recursos em larga escala.

**d) Modelar visualmente o software**

O uso de modelos visuais permite aos clientes que tenham um melhor entendimento de um dado problema, e dessa forma possam se envolver mais no projeto como um todo. A linguagem UML é amplamente utilizada no RUP para modelar o sistema em desenvolvimento.

**e) Verificação contínua da qualidade**

A preocupação com a qualidade é um dos fatores de grande importância nos projetos computacionais. As pessoas que estão envolvidas em um projeto, devem se preocupar com a qualidade do produto final, que é de responsabilidade de todos envolvidos no projeto. O RUP faz uso deste principio, para controlar o planejamento da qualidade, analisar a construção de todo o processo e integrar todos os membros da equipe de desenvolvimento.

**f) Controle de mudanças**

Em todos os projetos de software a existência de mudanças é inevitável. O RUP define métodos para controlar e monitorar mudanças. Como uma pequena mudança pode afetar aplicações de formas inteiramente imprevisíveis, o controle de mudanças é essencial para o sucesso de um projeto. O RUP também propõe a definição de áreas de trabalho seguras, garantindo a um programador que as mudanças efetuadas em outro sistema não afetarão o seu sistema.

### 1.2.2 Visão Geral do RUP

A arquitetura do RUP é dividida em duas dimensões, propiciando duas visões da descrição de um sistema, que são: aspectos dinâmicos e aspectos estáticos. A figura 1.2 mostra a visão geral do RUP.

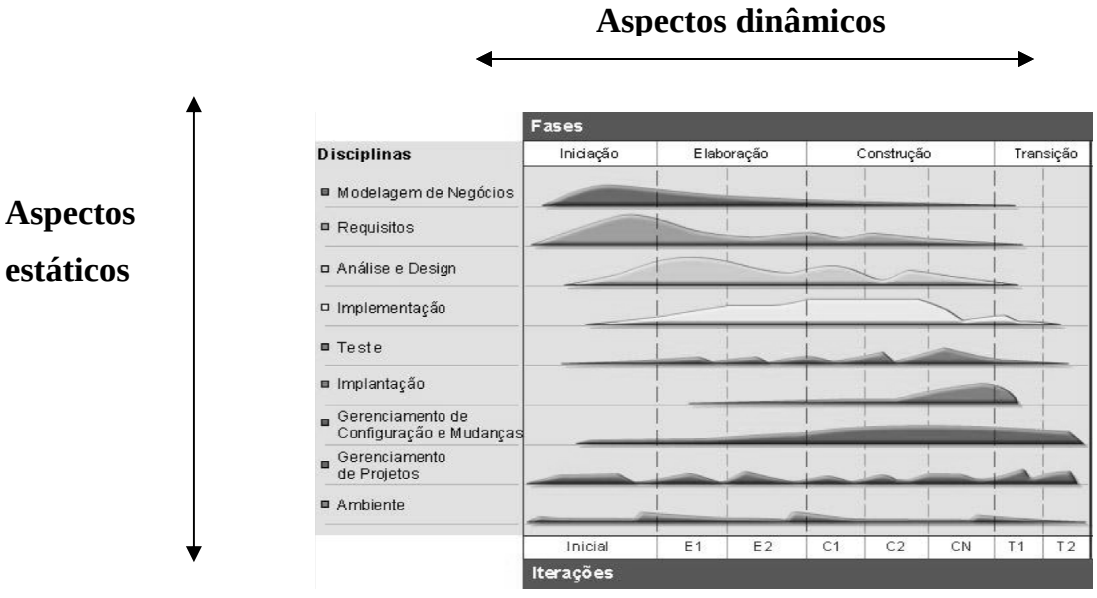


**Figura 1.2 Arquitetura do RUP, adaptado de**

**[http://www.wthreex.com/rup/portugues/index.htm.]**

A dimensão horizontal representa o tempo e os aspectos do ciclo de vida à medida que o software é desenvolvido. O processo é descrito em termos de fases (iniciação ou concepção, elaboração, construção e transição), iterações e marcos de referência [DANTAS, 2008] & [RATIONAL SOFTWARE CORPORATION, 2001]. Estes termos estão conceituados logo abaixo:

Na dimensão horizontal da arquitetura, o ciclo do projeto é representado por uma sequência de fases que vão do início ao fim de um projeto.

Já as iterações do processo representam sequências de atividades a serem realizadas em um período de tempo definido dentro de um projeto; onde em cada iteração é desenvolvida uma versão estável de um produto e os marcos de referência estão associados a objetivos definidos e alcançáveis, que podem ser analisados pelos clientes e desenvolvedores.

A dimensão vertical representa a estrutura estática do processo. Descreve como elementos do processo (atividades, disciplinas, artefatos e papéis) são agrupados em disciplinas de processo ou fluxos de trabalho [DANTAS, 2008]. Na estrutura estática um processo descreve "**quem**" está fazendo "**o quê**", "**como**" e "**quando**", conforme demonstrado na Figura 1.3. Os principais elementos presentes na estrutura estática são:

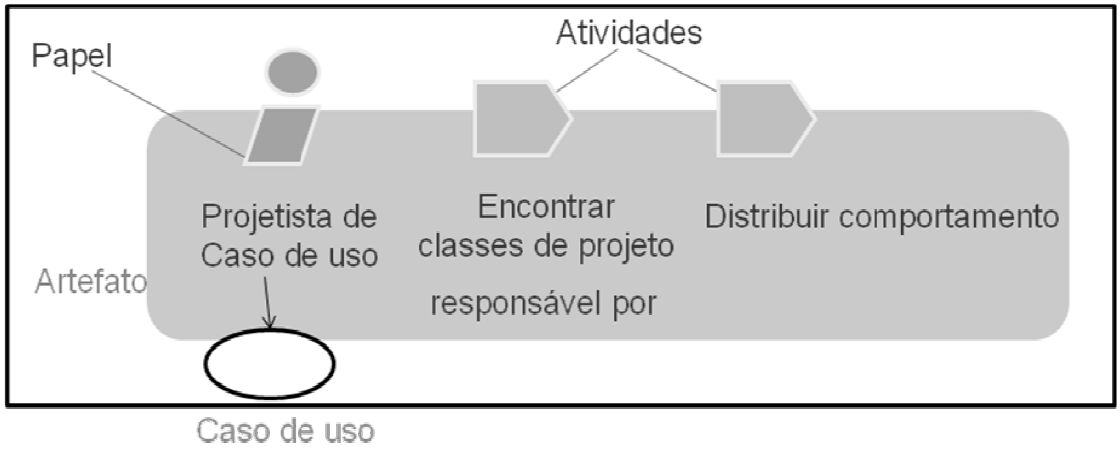


**Figura 1.3 Descrição do processo, adaptado de [http://www.wthreex.com/rup/portugues/index.htm.]**

- Papéis (*roles*) - representam quem será responsável por determinada função (tarefa);
- Atividades (*activities*)- representam unidades de trabalho que um indivíduo exercendo um papel pode ser solicitado a executar;
- Artefatos (*artifacts*) - representam os elementos tangíveis de um projeto. Podem assumir várias formas, como: modelo, documento, código fonte ou executável;
- Fluxos de trabalho (*workflows*) - Representam sequências de atividades que são executadas para a produção de um resultado. Os Fluxos de trabalho podem ser representados por diagramas de sequência, diagramas de colaboração e diagramas de atividades da linguagem UML.

### 1.2.3 Fases do RUP

Como dito anteriormente, o ciclo de desenvolvimento do RUP é dividido em 4 fases que são: Iniciação ou Concepção, Elaboração, Construção e Transição. O ciclo de vida termina com a entrega do produto final. Vale salientar que apesar das fases serem sequenciais, o RUP não é um processo que segue o modelo cascata, pois dentro de cada fase as disciplinas são executadas de maneira iterativa e incremental [DANTAS, 2008] & [MORAES, 2006].

Cada uma de suas quatro fases compreende um momento distinto dentro do ciclo de vida de um projeto de engenharia de software e, portanto, dão maior ou menor foco em algumas disciplinas, de acordo com a necessidade do projeto no decorrer de sua execução. São elas:

- Iniciação – foca na definição do escopo e análise da viabilidade econômica do projeto. Essa fase endereça riscos de requisitos e negócio antes de continuar com o projeto.

- Elaboração – foca nos riscos técnicos e arquiteturais. O escopo deve ser revisado e definido em detalhes. A maior parte dos requisitos funcionais e não-funcionais deve ser definida neste ponto. Os requisitos não funcionais podem ser encarados como fatores críticos para o sucesso, que descrevem o grau de risco envolvido no desenvolvimento do sistema.

- Construção – foca no tratamento dos riscos lógicos envolvidos na construção do produto. A fase de construção é de certa forma um processo de manufatura, em que a ênfase está no gerenciamento de recursos, pessoas e controle de operações para otimizar custos,codificações e qualidade.

- Transição – está associada com a entrega do produto ao usuário final, o qual pode requisitar treinamento, bem como encontrar *bugs* que precisem ser consertados. Muitas iterações podem ser necessárias antes que o usuário assine o aceite formal do sistema.

A seguir, cada uma das fases do RUP será descrita em detalhes.

**Iniciação**

Na fase de Iniciação o foco é em entender o escopo e os objetivos do projeto, e então obter informações suficientes para se decidir se vai seguir com o projeto ou abortá-lo. A Figura 1.4 ilustra o marco crucial desta fase.

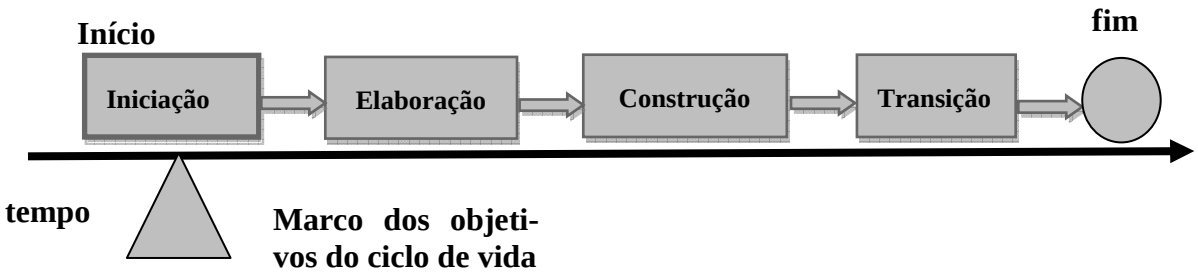


**Figura 1.4 Fase de iniciação, adaptado de [http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf]**

As atividades básicas da fase de iniciação são:

**1. Compreender o que se vai ser construído**

Para um entendimento comum por parte de todas as pessoas no projeto, algumas atividades são necessárias, tais como:

- Produção de um documento de visão: Esse documento deve apresentar claramente aos *stakeholders*[3] os benefícios do sistema, problemas que serão resolvidos pela aplicação, os principais usuários do sistema e os principais requisitos não funcionais. Ele pode ficar pronto na fase de Iniciação, mas pode ser refinado ao longo do projeto.
- Descrição do sistema: descrição do escopo do sistema sem muitos detalhes, através da identificação de casos de uso e seus atores.
- Detalhamento dos atores e casos de uso centrais: os principais casos de uso e atores são detalhados e em paralelo geram-se protótipos de interface com o usuário que permitirão a validação dos fluxos de eventos com os *stakeholders* e identificarão as funcionalidades centrais do sistema.

**2. Identificar as principais funcionalidades do sistema**

Identificar quais são os casos de uso essenciais e os que são arquiteturalmente significantes para que um maior tempo seja dedicado aos mesmos.

**3. Determinar uma possível solução**

O objetivo da fase de iniciação é identificar pelo menos uma possível solução para o desenvolvimento do projeto. Isso será possível com base na análise da tecnologia e arquitetura utilizadas em outros sistemas similares, bem como na análise de custos e riscos associados à utilização da arquitetura e de quais componentes de software serão necessários. Nesse momento poderá ser necessário implementar parte da arquitetura para assegurar-se que os riscos são aceitáveis.

**4. Entender o custo, cronograma e riscos do projeto**

Entender o que vai ser desenvolvido é importante, mas determinar os custos e prazos é crucial para um projeto. Outro fator que deve ser levado em conta no projeto é o entendimento dos

[3] *Stakeholders* são todas as pessoas que têm algum interesse nos resultados de um projeto.

riscos envolvidos no mesmo, pois caso venham a ocorrer, podem comprometer ou impedir a realização do projeto. A necessidade de entender os riscos decorre, principalmente, da constatação de que a quantidade e diversidade de riscos de um projeto excedem o montante de recursos alocados para neutralizar todos esses riscos durante a execução do projeto. Essa situação demanda que os riscos devam ser priorizados ou "gerenciados" adequadamente.

**5. Definir processos e ferramentas**

É importante que toda a equipe compartilhe uma visão de como o produto será desenvolvido, ou seja, qual processo será seguido. Dessa forma, deve ser definido qual IDE (*Integrated Development Environment*), ferramenta de gerenciamento de requisitos, modelagem visual, configuração e gestão de mudança, etc. deverão ser utilizadas.

## Elaboração

Nessa fase, as diferenças do modelo cascata em relação ao modelo iterativo ficam evidentes. Por que o modelo iterativo divide o desenvolvimento do produto em ciclos, nos quais a equipe identifica e especifica os requisitos relevantes e as previsões em relação aos riscos são feitas. Já o modelo em cascata é sequencial e presume que todo o processo terá uma sequência pré-definida, dando pouca importância ao gerenciamento de riscos de projetos. A Figura 1.5 ilustra o marco da fase de elaboração.

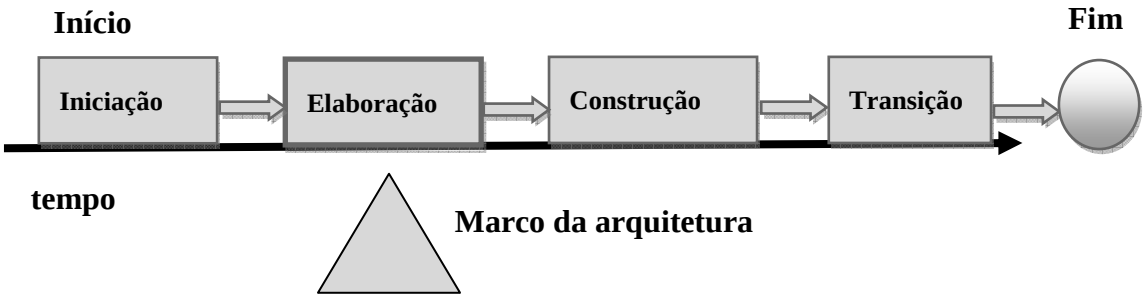


**Figura 1.5 Fase de elaboração, adaptado de [http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf]**

O objetivo dessa fase é definir e consolidar uma arquitetura para o sistema, provendo uma base estável para as atividades de projeto e implementação. Nessa fase são os endereçados os riscos associados com requisitos, arquitetura, custo, cronograma, processo e ferramentas, como explicados a seguir:

- Requisitos – deve-se analisar se a aplicação está sendo construída corretamente;
- Arquitetura – deve-se analisar se a solução correta está sendo construída;
- Custos e cronogramas – deve-se analisar se o projeto está atendendo os custos e prazos estimados;
- Processo e ferramentas – deve-se analisar se o processo e as ferramentas são adequados para a realização do trabalho.

A fase de elaboração é onde os requisitos que mais impactam na arquitetura do software são capturados em forma de casos de uso, bem como onde os cronogramas e plano de construção

do sistema são elaborados. Está é a fase mais critica, porque é nela que a engenharia é considerada completa, e os custos para modificação do sistema aumentam à medida que o projeto avança. Nessa etapa os riscos e custos aumentam.

### Construção

Essa fase é a etapa de construção do produto, onde a maior parte do trabalho é realizada. A Figura 1.6 ilustra o marco da fase de construção.

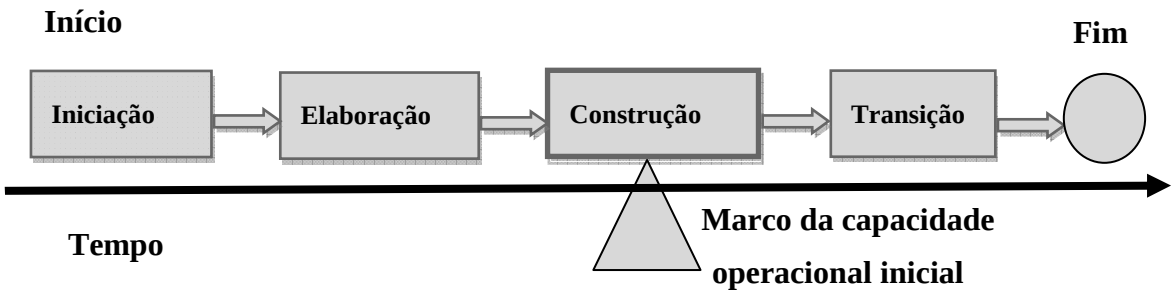


**Figura 1.6 Fase de Construção, adaptado de [http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf]**

No marco da capacidade operacional inicial, o produto está pronto para ser entregue para a fase de Transição. Todas as funcionalidades foram desenvolvidas e todos os testes alfa[4], se houver, foram concluídos. Os objetivos principais dessa fase são:

- Controlar os custos de desenvolvimento, otimizar os recursos e evitar retrabalho desnecessário;
- Concluir a análise, o projeto, o desenvolvimento e o teste de todas as funcionalidades necessárias;
- Desenvolver o produto completo de modo que esteja pronto para a fase de transição para a comunidade de usuários. Implica também em descrever os casos de uso restantes e outros requisitos, concluir a implementação e testar o software;
- Decidir se o software, o ambiente de trabalho e os usuários estão prontos para que o aplicativo seja implementado;
- Atingir certo paralelismo entre o trabalho das equipes de desenvolvimento.

Para atender a estes objetivos, as atividades básicas da fase de construção são:

- Gerenciamento de recursos, otimização do processo e controle do projeto;
- Desenvolvimento completo do componente e definição dos critérios de teste de aceitação;
- A avaliação dos *releases* do produto de acordo com os critérios de aceitação.

[4] Testes conduzidos pelo cliente nas instalações do desenvolvedor.

Os critérios de avaliação para a etapa de construção envolvem responder às seguintes perguntas:

- O *release* do produto é estável o suficiente para ser implantado na comunidade de usuários?
- Todos os envolvidos estão prontos para a transição?
- As despesas reais com recursos ainda são aceitáveis se comparadas com as planejadas?

**Transição**

Nessa fase, o sistema está pronto. O foco da fase é garantir que o software esteja disponível para os usuários finais. A Figura 1.7 ilustra o marco da fase de transição.

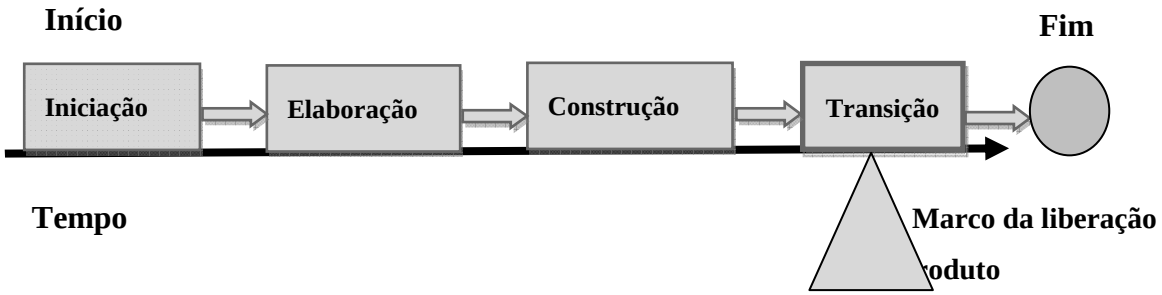


**Figura 1.7 Fase de Transição, adaptado de [http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf]**

No final da fase de transição, os objetivos do projeto devem ter sido atendidos e o projeto deve estar concluído. Em alguns casos, o ciclo de vida atual pode combinar com o início de outro ciclo de vida, levando à próxima geração do mesmo produto.

O ciclo de desenvolvimento ocorre quando o processo passa pelas 4 fases do RUP e os ciclos sucessivos são chamados de evoluções de ciclos, como pode ser visto na Figura 1.8.

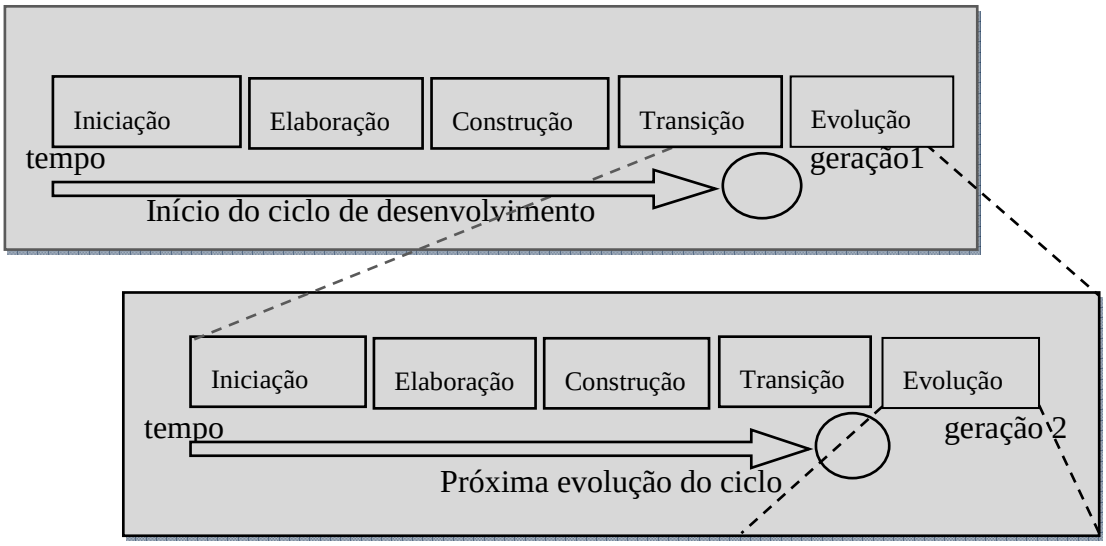


**Figura 1.8 Ciclo de desenvolvimento, adaptado de [http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf]**

A fase de transição pode ser mais simples ou mais complexa, depende do tipo do produto. Por exemplo, um *release* de um produto de prateleira pode ser algo simples, enquanto que a substituição de um sistema de controle aéreo é algo complexo.

As atividades que serão realizadas nas iterações desta fase dependem muito da meta que se deseja. Por exemplo, para corrigir erros do sistema, normalmente são utilizadas as atividades de implementação e teste. As atividades básicas dessa fase são: realizar planos de implantação, testar o produto final, criar *release* do produto, conseguir *feedback* do usuário, construir o produto com base no *feedback* e disponibilizar o produto para o usuário final.

### 1.2.4 Disciplinas

Uma disciplina no RUP expressa todas as atividades que devem ser realizadas para produzir um conjunto de artefatos. O RUP organiza suas disciplinas da seguinte forma [PERRELLI, 2009]:

- Fluxos de processo correspondem às atividades de desenvolvimento: modelagem de negócios, requisitos, análise e projeto, implementação, testes e implantação.
- Fluxos de suporte correspondem às atividades de gerenciamento e infraestrutura: gerenciamento de configuração e mudanças, gerenciamento de projeto e ambiente.

As disciplinas dos fluxos de processo são descritas a seguir:

- **Modelagem do negócio**: Os objetivos desta disciplina são compreender a estrutura e a dinâmica da organização-alvo; entender os problemas da organização; fazer com que os clientes e desenvolvedores tenham um entendimento comum em relação à organização e obter os requisitos necessários do sistema para sustentar a organização.

  A disciplina de Modelagem e Negócios é baseada em modelos de casos de uso de negócio e modelo de objeto de negócio. O modelo de caso de uso de negócio descreve uma sequência de atividades necessárias para fornecer um resultado para o ator de negócio e o modelo de objetos de negócio descreve a realização dos casos de uso de negócio.

- **Requisitos:** Na disciplina de requisitos o foco é manter concordância entre os clientes e as partes interessadas no que diz respeito àquilo que o sistema deve fazer; fornecer aos desenvolvedores uma melhor compreensão dos requisitos do sistema; ter uma base para estimar custos e prazos de desenvolvimento do sistema e definir uma interface para o usuário ter um entendimento de como o sistema funcionará de acordo com as suas necessidades.

  Os modelos de caso de uso de negócio e objetos de negócio produzidos na disciplina de Modelagem e Negócio servirão como informação importante para esse fluxo. Além desses documentos, serão criados um documento de visão, um modelo de casos de uso, e a especificação suplementar. O documento de visão descreve qual o entendimento que as partes envolvidas do sistema têm dele. O Modelo de casos de uso descreve as funções fornecidas pelo sistema, onde cada caso de uso representa uma sequência de ações que resultam em valor para um ator especifico. A Especificação suplementar captura os requisitos que não são capturados pelos casos de uso no modelo de casos de uso. Exemplos de especificação suplementar são: definição de atributos de quali-

dade, confiabilidade e usabilidade do sistema, bem como a definição da tecnologia necessária para colocar o sistema em uso.

- **Análise e Projeto:** Os objetivos desse fluxo são: derivar o projeto do sistema a partir dos requisitos; obter uma arquitetura robusta do sistema e adaptar o projeto de acordo com o ambiente de implementação. O modelo de análise e projeto é o principal produto dessa disciplina. O modelo de análise e projeto possui a realização dos casos de uso. A realização de casos de uso expressa o modelo de projeto associado a cada caso de uso, um exemplo seria os diagramas de classes associados às classes e subsistemas participantes da implementação de cada caso de uso.
- **Implementação:** O objetivo é construir um sistema, produzindo o código em termos de subsistemas de implementação organizados em camadas; implementar classes e objetos em termos de componentes (arquivo fonte, executável, binários, etc.). A disciplina de implementação limita o seu escopo de maneira que as classes individuais possam ser testadas em unidades.
- **Teste:** A disciplina de teste atua em vários aspectos como uma provedora de serviços a outras disciplinas. O teste foca principalmente na avaliação da qualidade do produto, realizada através de algumas práticas, tais como: encontrar e documentar erros no software, avisar sobre a qualidade geral observada no software, validar as funções da maneira que foram projetadas, verificar se os requisitos foram implementados de forma correta, entre outras.
- **Implantação:** Descreve as atividades que possibilitam que o software seja disponibilizado para o usuário final. A ênfase da disciplina de implantação é testar o produto no ambiente de implantação, onde são realizados testes beta antes do produto ser entregue ao usuário final.

As disciplinas dos fluxos de suporte são descritas a seguir:

- **Ambiente:** Esta disciplina centraliza-se nas atividades necessárias à configuração do processo para um projeto. Ao fazer isso, ela também serve de suporte a todas as outras disciplinas. A meta das atividades dessa disciplina é organizar o ambiente de desenvolvimento de software (processos e ferramentas) que dará suporte à equipe de desenvolvimento.
- **Gerência de configuração e mudanças:** O gerenciamento de configuração e mudanças envolve a identificação de itens de configuração, restrições a mudanças nestes itens, verificação de mudanças feitas nos itens e definição e gerenciamento das configurações dos mesmos durante o processo de desenvolvimento. Os métodos, processos e ferramentas que são usados para fazer esse controle em uma organização podem ser considerados como o sistema de gerência de configuração (SGC). O SGC manuseia informações importantes sobre o processo de desenvolvimento, sobre a implantação e manutenção do software, além de conservar o acervo de artefatos potencialmente reusáveis que resultam da execução destes processos.

  O SGC é essencial para o controle dos artefatos produzidos por várias pessoas que trabalham em um projeto. O controle ajuda a evitar desordens custosas e garante que os artefatos resultantes não sejam conflitantes em relação a questões como: atualização simultânea, notificação de mudanças nos artefatos compartilhados pelos desenvolvedores, e múltiplas versões.

- **Gerência de projetos:** O gerenciamento de projetos de software é a arte de balancear os objetivos, o gerenciamento de riscos e as restrições para entregar, com sucesso, um produto que esteja de acordo com as necessidades dos clientes e usuários. O propósito do fluxo de gerenciamento de projetos é fornecer um processo para o gerenciamento de projetos de software; bem como providenciar guias para o planejamento, recrutamento de pessoal, execução e monitoração dos projetos.

## 1.3 OpenUp

Originalmente, o OpenUp (*Open Unified Process*) era conhecido como BUP (*Basic Unified Process*). Em 2005, a IBM liberou o BUP para a fundação Eclipse e este foi renomeado para OpenUp [OPENUP, 2007].

O OpenUp é considerado um processo híbrido, pois foi desenvolvido pela IBM com base nos processos RUP e XP, tendo como principal objetivo reunir as melhores características de cada uma desses processos. Apesar dessa característica híbrida, o OpenUp está sendo citado aqui como um processo tradicional, devido à sua rigidez na documentação de um produto e de suas práticas seguirem o modelo RUP de desenvolvimento.

### 1.3.1 O OpenUp e suas características

O OpenUP é um processo de desenvolvimento de software *open source*, projetado para equipes pequenas e centralizadas, atualmente mantido pelo Projeto Eclipse. Este processo segue uma abordagem iterativa e incremental de desenvolvimento de software. Contudo, abraça uma filosofia pragmática e ágil que foca na natureza colaborativa do desenvolvimento de *software* [MONTEIRO e YBANEZ, 2009]. Além disso, é um processo independente de ferramenta que pode ser estendido para direcionar uma grande variedade de tipos de projeto.

O OpenUP é um processo **mínimo**, pois apenas conteúdos fundamentais do processo são definidos; **completo**, pois abrange todas as fases do ciclo de vida de desenvolvimento; e **extensível**, pois permite que novos conteúdos de processos sejam acrescentados para atender de forma mais completa as características de um determinado projeto [CUNHA & VASCONCELOS, 2007].

#### 1.3.1.1 Princípios básicos do OpenUp

O OpenUP é baseado em 4 princípios básicos que são:

**a) Equilíbrar as prioridades concorrentes para maximizar o valor para os *stakeholders***

Para que se alcance o sucesso, *stakeholders* e participantes do projeto devem convergir para um claro entendimento e concordar com três fatores: O problema a ser resolvido, Restrições impostas à equipe de desenvolvimento (custo, cronograma, recursos, regulamentos) e Restrições impostas à solução. Coletivamente, estes três itens representam os requisitos para o desenvolvimento do sistema. O desafio de todos os participantes do projeto é criar uma solução que maximize o seu valor para os *stakeholders*, mesmo que sujeito a restrições. O equilíbrio está em fazer uma análise crítica do custo-benefício entre características desejadas e as decisões de projeto subsequentes que definem a arquitetura do sistema.

**b) Focar na evidenciação da arquitetura**

Sem uma base arquitetural, um sistema não evoluirá de forma eficiente, será difícil organizar a equipe ou comunicar ideais sem um foco técnico comum que a arquitetura fornece. Dessa forma, a arquitetura deve ser utilizada como um ponto focal, para que desenvolvedores possam alinhar seus interesses e idéias ao se tornarem evidentes as decisões técnicas essenciais tomadas em cada evolução arquitetural.

**c) Colaborar para alinhar os interesses e compartilhar o entendimento**

Em uma equipe, cada membro tem seus próprios conhecimentos, habilidades e maneiras de fazer as coisas. Este princípio tem a finalidade de alinhar essas diferenças de forma que o projeto seja beneficiado bem como que todos os membros da equipe tenham um entendimento comum sobre o projeto. O contínuo aprendizado também é estimulado por este principio fazendo com que cada membro da equipe desenvolva mais habilidades e incremente seus conhecimentos.

**d) Evoluir para continuamente obter *feedback* e promover melhorias**

O objetivo deste princípio é fazer com que através de *feedbacks*, obtenham-se maneiras de melhorar o produto e também o processo da equipe envolvida. Através de *feedbacks,* podem-se identificar potenciais riscos e tratá-los mais cedo durante o projeto.

É importante ter uma visão clara do objetivo do projeto, para que seja possível medir o progresso e identificar possíveis melhorias no processo.

### 1.3.1 Visão Geral do OpenUp

O processo pode ser facilmente entendido através das 3 camadas apresentadas na Figura 1.9.

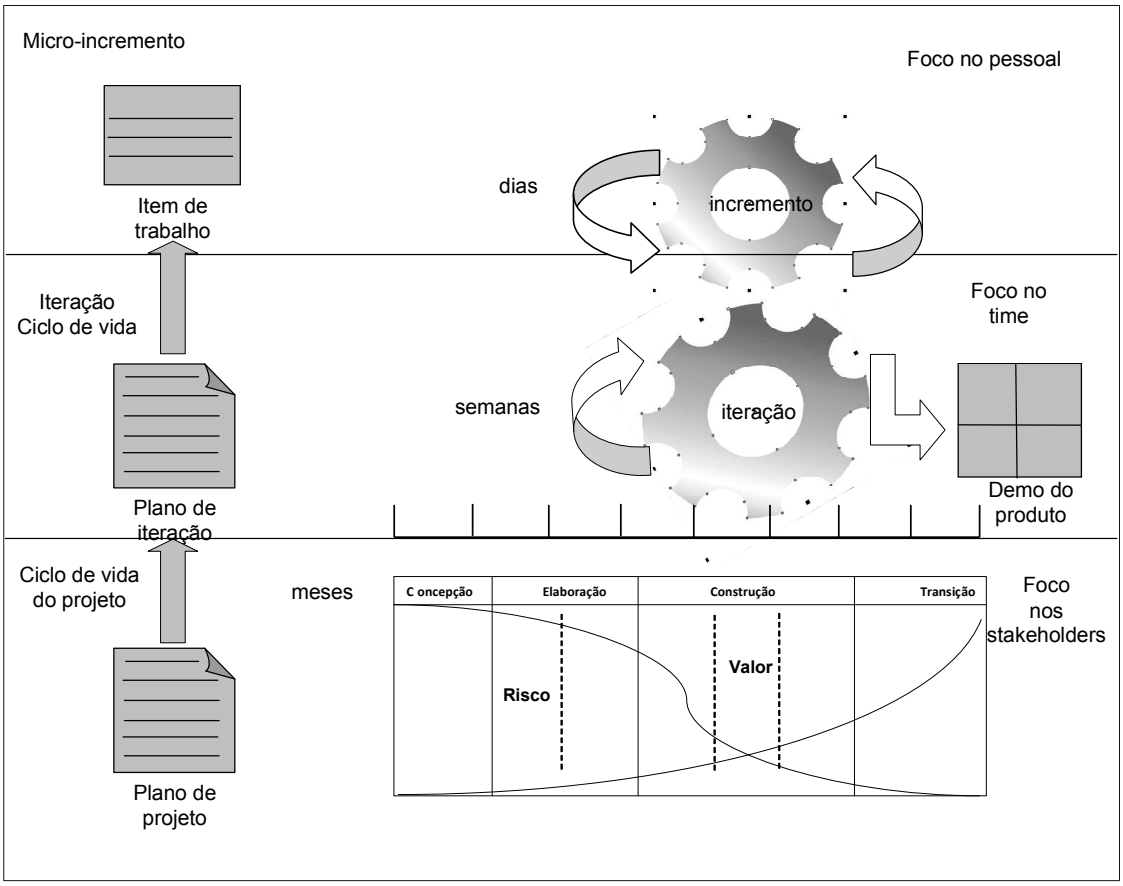


**Figura 1.9 Visão Geral do OpenUp, adaptado de [OPENUP, 2007]**

**Ciclo de Vida do Projeto** – Fases com foco nas necessidades dos s*takeholders*.

O OpenUP apresenta a mesma distribuição de fases já conhecidas no RUP, onde o critério de saída de cada fase é no mínimo responder às seguintes perguntas:

- **Concepção:** Todos os *stakeholders* concordam com o escopo e objetivos do projeto?
- **Elaboração:** Todos concordam com a arquitetura proposta e o valor entregue ao cliente considerando os riscos levantados?
- **Construção:** Existe uma aplicação que está bem próxima a ser finalizada?
- **Transição:** A aplicação está finalizada e o cliente satisfeito?

### Ciclo de Vida da iteração com foco no time

Além da divisão por fases, o OpenUP divide o projeto em iterações planejadas que podem variar de alguns dias a algumas semanas (a média recomendada é de 4 semanas, podendo ser reduzida ou aumentada para até aproximadamente 6 semanas). Ao final de cada iteração deve ser gerado um incremento ao Produto (*Build* executável ou demo), bem como geralmente é realizada uma retrospectiva e avaliação onde são discutidas as lições aprendidas e a saúde do projeto. Vale mencionar que o principal objetivo da retrospectiva é aprender com erros e acertos e não apontar culpados.

**Micro incrementos com foco individual**

Um micro incremento é a execução de um pequeno passo que deve ser mensurável para alcançar os objetivos de uma iteração. Este pode representar o resultado de alguns dias ou horas de trabalho de uma pessoa ou grupo determinado.

### 1.3.2 Fases do OpenUp

Como visto anteriormente, o OpenUp é dividido em quatro fases: concepção, elaboração, construção e transição. Cada fase pode ter quantas iterações forem necessárias, dependendo do grau de incerteza do domínio de negócio, da tecnologia a ser utilizada, da complexidade arquitetural, do tamanho do projeto, etc. O processo foi projetado para suportar equipes pequenas e co-localizadas, com 3 a 6 membros e que trabalhe em um projeto que irá durar de 3 e 6 meses.

**Concepção**

A fase de concepção no OpenUp é responsável por conseguir que se tenha um entendimento simultâneo entre todos os *stakeholders* dos objetivos do projeto. Assim, os principais objetivos dessa fase são: Entender o que construir a partir da visão dos *stakeholders* a respeito do produto a ser desenvolvido, definindo quem são os *stakeholders* do sistema e por que; Definir o escopo do sistema e seus limites; Identificar quais os requisitos mais críticos; Definir pelo menos uma arquitetura candidata e sua aplicação prática; e Entender o custo, cronograma e os riscos associados ao projeto.

**Elaboração**

A fase de elaboração tem o propósito de estabelecer uma *baseline* da arquitetura do sistema para o esforço de desenvolvimento que será desempenhado na próxima fase. Algumas finalidades para a fase de elaboração são: Obter um entendimento mais detalhado dos requisitos, o que permite ao usuário criar um plano mais detalhado e obter comprometimento dos *stakeholders;* Projetar, implementar e testar um esqueleto da estrutura do sistema, ou seja, definir uma arquitetura executável; Mitigar os riscos essenciais e produzir um cronograma e uma estimativa de custos precisos; e Refinar e detalhar o plano de projeto de alto nível.

**Construção**

A finalidade da fase de construção é terminar o desenvolvimento do sistema baseado na arquitetura estabelecida na fase de Elaboração. Existem objetivos para a fase de Construção que ajudam no processo de desenvolvimento com baixo custo para um produto completo, como exemplo:

- **Desenvolver de forma iterativa** um produto completo com a descrição do restante dos requisitos, preencher os detalhes do projeto, terminar a implementação e testar o software; Liberar a primeira versão beta do sistema e determinar se os usuários já estão prontos para que a aplicação possa ser implantada;
- **Minimizar os custos de desenvolvimento e conseguir algum grau de paralelismo** entre os desenvolvedores ou as equipes de desenvolvimento, como por exemplo, atri-

buindo os componentes que podem ser desenvolvidos independentemente para desenvolvedores distintos.

**Transição**

A fase de Transição possui alguns objetivos que ajudam a fazer um ajuste na funcionalidade, desempenho e na qualidade total do produto beta proveniente da fase anterior, como por exemplo:

- **Executar o teste Beta** para validar se as expectativas dos usuários foram atendidas**;**
- **Realizar algumas atividades de ajuste fino**, tais como reparação de erros e melhorias no desempenho e na usabilidade;
- **Obter a concordância dos *stakeholders*** de que a distribuição está completa, incluindo vários níveis de testes para aceitação do produto, como testes formais, informais e beta;
- **Melhorar o desempenho de projetos futuros com documentação das lições aprendidas;** e
- **Melhorar o ambiente** de processos e ferramentas para o projeto.

### 1.3.3 Disciplinas

As disciplinas do OpenUP são detalhadas a seguir:

**a) Requisitos**

A disciplina de requisitos tem como atividades: entender o problema a ser resolvido; entender as necessidades dos *stakeholders*; definir os requisitos para a solução; definir o escopo do sistema; identificar interfaces externas ao sistema; identificar restrições técnicas na solução; fornecer a base para o planejamento das iterações e fornecer a base inicial para a estimativa de custo e cronograma. Esta disciplina está relacionada a outras disciplinas da seguinte forma: a disciplina de Análise e Projeto obtém suas entradas primárias a partir da disciplina de Requisitos; a de Teste valida o sistema de acordo com os requisitos; a de Gerência de Configuração e Mudança fornece os mecanismos para controlar as mudanças nos requisitos; e a de Gerência de Projeto planeja o projeto e atribui os requisitos a cada iteração analisando os requisitos priorizados e atribuindo o trabalho.

**b)Análise e Projeto**

Os propósitos da Análise e Projeto são: transformar os requisitos em um projeto do que será o sistema, desenvolver uma arquitetura robusta para o sistema e adaptar o projeto para corresponder ao ambiente de implementação. A disciplina de Análise e Projeto está relacionada a outras disciplinas da seguinte forma: a disciplina de Requisitos provê a primeira entrada para a Análise e Projeto; a disciplina de Implementação implementa o que foi projetado; a disciplina de Teste testa a implementação do que foi projetado durante a Análise e Projeto e a disciplina de Geência de Projetos planeja as atividades de análise e projeto.

**c) Gerência de Configuração e Mudança**

Esta disciplina está relacionada ao controle de mudanças de artefatos e à habilidade de manter versões e configurações consistentes dos artefatos. A finalidade desta disciplina é: manter um conjunto de produtos de trabalho consistente à medida que o desenvolvimento do sistema evolui; manter construções de software consistentes; fornecer meios eficientes para se adaptar às mudanças, re-planejando o trabalho adequadamente; e fornecer dados para a medição do progresso das atividades relacionadas a mudanças.

**d) Implementação**

A disciplina de implementação tem como propósitos: construir o sistema de forma incremental e verificar que as unidades técnicas usadas para construir o sistema funcionam como especificadas. Esta disciplina se relaciona com as outras disciplinas da seguinte forma: a disciplina de Requisitos define o que será implementado; a de Análise e Projeto organiza e define o escopo da implementação; a de Teste valida se a construção do sistema atende aos requisitos; a de Gerência de Configuração e Mudança fornece mecanismos para controlar mudanças no sistema em construção e a disciplina de Gerência de Projeto planeja quais funcionalidades serão implementadas em cada iteração.

**e) Gerência de Projetos**

A disciplina de Gerência de Projetos tem o objetivo de: manter a equipe focada na entrega contínua do produto de software testado para a avaliação dos *stakeholders;* ajudar a priorizar a sequência de trabalho; ajudar a criar um ambiente de trabalho eficaz para maximizar a produtividade da equipe; manter os *stakeholders* e a equipe informados sobre o progresso do projeto e fornecer uma estrutura para controlar o risco do projeto e para adaptar-se continuamente às mudanças

O gerenciamento de projeto age como um elo entre os *stakeholders* e a equipe de desenvolvimento. É interessante que as atividades de gerenciamento de projeto adicionem valor ao criar um ambiente de trabalho de elevado desempenho, onde os clientes tenham a confiança na equipe do projeto, possam conhecer as capacidades e restrições da plataforma técnica e que os membros da equipe de projeto entendam as necessidades reais dos clientes, produzindo continuamente um produto de software de qualidade.

**f) Teste**

O propósito desta disciplina é: encontrar e documentar defeitos; validar e provar as suposições feitas no projeto e especificar os requisitos através de demonstrações concretas; validar se o produto de software foi feito como projetado e validar se os requisitos estão apropriadamente implementados. A disciplina de Teste está relacionada as outras disciplinas da seguinte maneira: a disciplina de Requisitos captura requisitos para o produto de software e é um das contribuições primárias para identificar que testes executar; a de Análise e Projeto determina o projeto apropriado para o produto de software, que é outra contribuição importante para identificar que testes executar; a de Implementação produz construções do produto de software que é validado pela disciplina de Teste; a de Gerência de Projeto planeja o projeto e o trabalho necessário em cada iteração e a de Gerência de Configuração e

Mudança controla mudanças dentro do projeto. O esforço de teste verifica se cada mudança foi completada adequadamente. Ativos de teste são mantidos sob gerência de configuração.

## 1.4 MSF

O Microsoft Solutions Framework (MSF) surgiu em 1994, a partir da análise de como a Microsoft desenvolvia seus produtos. Basicamente o MSF é uma compilação das boas práticas utilizadas pela empresa, que foi criado tanto para uso interno como para uso de seus clientes.

Apesar de ter sido criado pela Microsoft, o MSF aborda basicamente o processo de construção de soluções, não se prendendo ao uso de produtos desta empresa [CARDIM, 2006 e VASQUES, 2007]. Inicialmente, surgiu o MSF 3.0 que era composto por quatro elementos básicos: princípios fundamentais, modelo de processo, modelo de equipe e disciplinas. Com a popularização dos processos ágeis, a versão 4.0 do MSF foi lançada com duas variantes: o MSF Agile Software Development (MSF4ASD), abordando processos ágeis; e o MSF for CMMI Process Improvement (MSF4CMMI), mais focado em processos tradicionais.

Ao longo do tempo o MSF já se chamou: MDF (Microsoft Development Framework) e PCM (Product Cycle Model), consolidando-se como MSF após o lançamento do livro Microsoft Secrets, de Michael Cusumano (1995).

### 1.4.1 MSF e suas características

As principais características do MSF são: permite uma fácil compreensão tanto por parte da equipe como do cliente, além de ser bem flexível em sua aplicação; É um processo iterativo e incremental, assim como o RUP e OpenUp; permite acompanhar a autonomia técnica dos colaboradores no projeto; e possui uma teoria de controle, onde pequenas iterações permitem calibrar com mais precisão e agilidade estimativas e reduzir as margens de erro.

#### 1.4.1.1 Princípios básicos MSF

Nessa seção, serão apresentados os princípios básicos do MSF para as equipes que utilizam os seus conceitos para o desenvolvimento de projetos de software. [ARAÚJO, 2005]

**a) Parceria com o cliente**

A idéia é ter o cliente como membro do time para fazer validações constantes de forma que ele esteja comprometido com o projeto e possa entender o que está sendo feito. Assim, os riscos do projeto podem ser mitigados.

**b) Qualidade é trabalho de todos**

O termo qualidade é muito subjetivo. Promover qualidade significa investir em pessoas, ferramentas e processos. Requer tanto prevenção de "*bugs*/problemas" quanto verificação de possíveis soluções.

**c) Trabalho em direção a uma visão compartilhada**

Através da visão do projeto, os membros do time podem definir prioridades, tomar decisões e garantir que os esforços estejam alinhados aos resultados que se esperam.

**d) Manter-se ágil, adaptar-se às mudanças**

Quanto mais uma organização procura maximizar o impacto no negócio de um investimento em tecnologia, mais ela descobre novos ambientes e desafios.

Os projetos de tecnologia têm uma característica relevante: mudanças constantes. Alterações no projeto devem ser esperadas e é impossível isolar a entrega do projeto. Com isso, o MSF foi desenvolvido para gerenciar e antecipar mudanças. Todas as alterações são aprovadas pela equipe, dessa forma mitiga-se os impactos negativos das mudanças.

**e) Encorajar comunicação aberta**

Para desenvolver um bom trabalho, é necessário que todos os membros da equipe tenham conhecimento prévio do que está sendo feito. Isso minimiza o desconhecimento, diminui as incertezas e o retrabalho.

**f) Autorização dos membros da equipe**

Dar poder aos membros da equipe é um grande diferencial do MSF, pelo fato de pregar um modelo em rede (não hierárquico), um modelo de auto-gestão individual; onde cada membro é responsável pela entrega do produto final.

**g) Estabelecer a responsabilidade desobstruída e responsabilidade compartilhada**

Definição clara dos papéis e das responsabilidades de cada integrante da equipe é um dos principais fatores de sucesso. Se não forem estabelecidas responsabilidades individuais, poderá implicar em um retrabalho e certa insegurança por parte de algum *stakeholder* sobre a função que lhe foi imposta dentro do projeto.

**h) Foco em entregar um valor de negócio**

Os projetos de tecnologia não devem focar em “entregas de tecnologia”, mas em “entregas com valor tangível ao negócio”. O projeto tem que possuir uma ligação íntima com o negócio, se não existir essa ligação pode resultar em entregas com atraso e projetos cancelados.

**i) Aprender com todas as experiências**

O MSF recomenda revisões e coleta de lições aprendidas no projeto para que erros não se repitam.

**j) Criar sempre a possibilidade de serem entregues produtos**

O time deve se comprometer a sempre estar criando um produto de excelência, mesmo enquanto realiza mudanças no mesmo. Os desenvolvedores devem está sempre executando o produto e o cliente testando, para que possíveis problemas sejam solucionados antes do prazo de término do projeto.

### 1.4.2 Visão Geral do MSF

Um projeto MSF é regido por iterações ou ciclos. Em um ciclo, cada componente da equipe executa suas funções e atualiza o resultado do seu trabalho conforme a necessidade. Os ciclos se repetem até que o projeto seja concluído ou cada versão seja lançada. Cada componente da equipe será responsável por um ou mais papéis, dependendo do tamanho ou da complexidade do projeto. O MSF é dividido em dois modelos: Modelo de equipe (Team Model) e Modelo de processo (Process Model).

O modelo de equipe tem por objetivo definir os papéis e responsabilidades das pessoas que estarão envolvidas numa equipe de um projeto. Neste modelo são definidos seis papéis: Gerenciamento de programa, Gerenciamento de produto, Experiência do usuário, Gerenciamento de *release*, Teste, Arquitetura e Desenvolvimento, conforme ilustrado na Figura 1.10.

O papel de **Gerenciamento de programa** foca em atender o objetivo de concluir e entregar o projeto dentro dos prazos e custos estimados, gerando valor de negócio para o cliente.

O papel de **Gerenciamento do produto** satisfaz os clientes com valor de negócio, marketing, advocacia do cliente e planejamento do produto;

O papel de **Experiência do usuário** atua nas áreas de acessibilidade, internacionalização, produção de material de treinamento, suporte, pesquisa de usabilidade e teste, advocacia do usuário e *design* de interface;

O papel de **Gerenciamento de *release*** está relacionado aos processos de infraestrutura, suporte, operações, logística e gerenciamento comercial das publicações;

O papel de **Teste** atua nas áreas de planejamento de teste, engenharia de teste e relatório de testes;

O papel de **Desenvolvimento** atua nas áreas de consultoria tecnológica, implementação da arquitetura, desenvolvimento da aplicação e desenvolvimento da infraestrutura; e

O papel de **Arquitetura** é responsável pelas soluções de arquitetura e por garantir que as mesmas satisfazem todas as necessidades e expectativas.

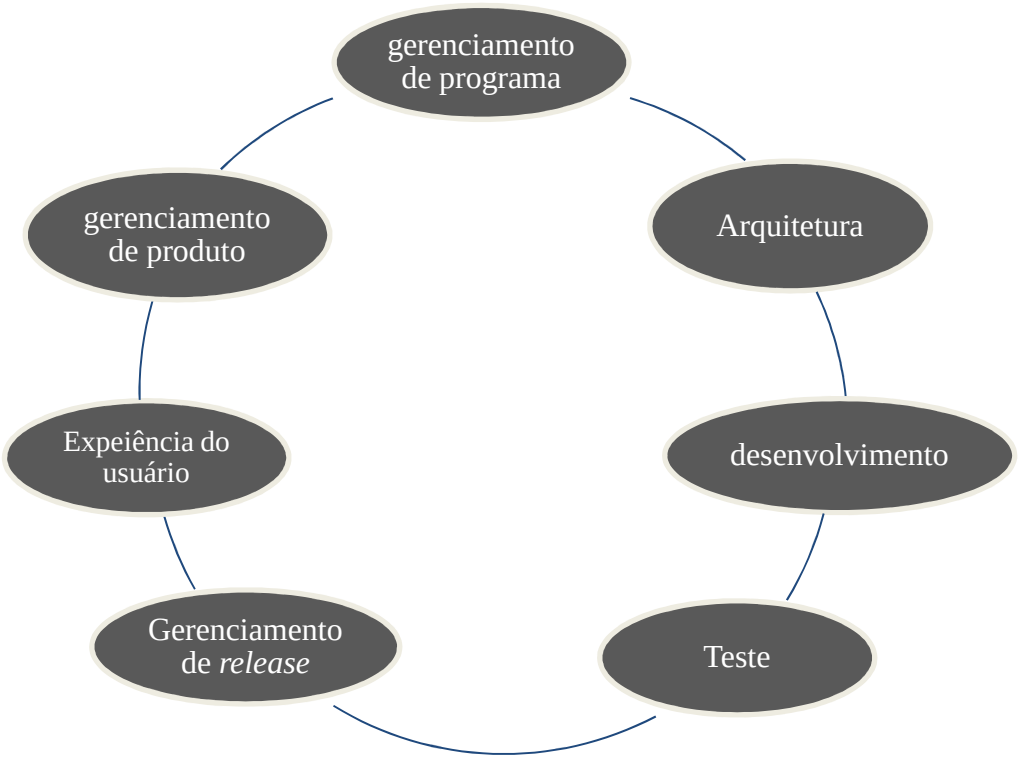


**Figura 1.10 Modelo de equipes, adaptado de [ARAÚJO, 2005]**

O modelo de processo trabalha em conjunto com o modelo de equipe organizando o processo em fases distintas, as quais serão explicadas na próxima seção.

### 1.4.3 Fases do Modelo de Processos do MSF

O modelo de processos do MSF é dividido em 5 fases: Visão, Planejamento, Desenvolvimento, Estabilização e Implantação. Cada fase descreve um conjunto de subprodutos que devem ser entregues, assim como marcos que devem ser atingidos e os respectivos critérios de aceitação. Essas fases serão descritas a seguir e mostradas na Figura 1.11.

**a) Visão:** Esta fase tem como foco fazer com que a equipe tenha uma visão comum do projeto. As atividades realizadas nessa fase são: formação da equipe, elaboração do projeto (definição da visão mais ampla do que o projeto deve fazer) e a definição do escopo do projeto.

**b) Planejamento:** Durante essa fase a equipe estima custos, prepara plano de trabalho e programa os *deliverables*[5]. É nesta fase que a equipe lista os requisitos do projeto (requisitos de negócio, do usuário, operacionais e de sistema).

**c) Desenvolvimento:** nessa fase a equipe implementa a maioria dos componentes da solução. Os principais artefatos gerados nessa fase são: código fonte e executável, *scripts* de instalação e configuração, especificação funcional e especificação dos casos de teste.

**d) Estabilização:** nessa fase, a solução é testada em um ambiente o mais próximo possível do de produção até atingir a estabilidade. A equipe se preocupa em resolver erros encontrados nos testes e prepara a solução para a implantação. Esta fase só termina quando todos os erros são corrigidos. No entanto, as atividades de estabilização do produto podem continuar enquanto os componentes do projeto são transferidos do ambiente de teste para o ambiente de produção.

**e) Implantação:** esta fase tem por objetivo transferir a solução para o ambiente de produção, onde a mesma deverá passar a gerar o valor de negócio esperado pelo cliente. O reconhecimento, por parte do cliente, de que a equipe atingiu os objetivos a que se propôs, finaliza esta fase.

[5] Produtos que serão entregues pelo projeto.

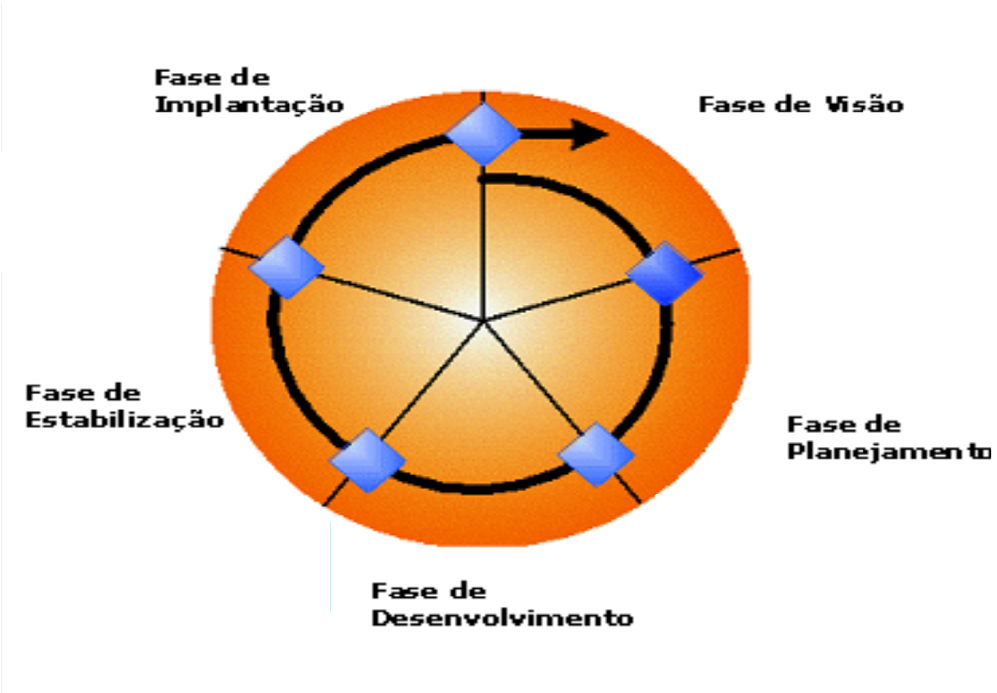


**Figura 1.11 Fases do Modelo de Processos do MSF**

### 1.4.4 Disciplinas do MSF

O MSF possui três disciplinas de gerenciamento que são:

- **Gerenciamento de projeto**: é uma disciplina que incorpora atividades de diversas áreas de conhecimento; onde a maioria das responsabilidades é atribuída ao gerente de projeto. A disciplina de gerenciamento de projeto ajuda o time a obter sucesso sem desperdiçar recursos adicionais que não fornecem valor suficiente aos recursos investidos.
- **Gerenciamento de risco:** é o gerenciamento pró-ativo, compreensivo, visando o sucesso e diminuindo fatores negativos que impactariam no fracasso do projeto. A gerência de riscos é uma resposta à incerteza intrínseca em projetos de tecnologia.
- **Gerenciamento de Conhecimento:** A disciplina de gerenciamento de conhecimento identifica habilidades exigidas pelo time, alocando desse modo, recursos que o projeto necessita e criando oportunidades de aprendizado e crescimento.

Além das disciplinas de gerenciamento, o MSF na sua versão para CMMI (MSF4CMMI), possui também disciplinas (chamadas atividades) relacionadas à engenharia [MICROSOFT, 2010], são elas:

- **Desenvolvimento de requisitos (*Developing Requirements*)**
  Esta disciplina foca na análise e especificação de requisitos do cliente, do produto e de componentes do produto, baseados em planos de desenvolvimento. Essa disciplina preocupa-se com o desenvolvimento dos requisitos do ciclo de vida do produto, criação do produto inicial, define o ambiente técnico (ex.: arquitetura da computação) para

que o produto seja implantado e define métodos de elicitação de requisitos (ex.: entrevistas, reuniões diárias).

- **Planejamento de requisitos (*Arranging Requirements into a Product Plan*)**
  Com base nos requisitos especificados, esta atividade foca no planejamento da implementação destes requisitos, bem como no planejamento de mudanças nos requisitos.
- **Criação de uma Solução Arqitetural (*Creating a Solution Architecture*)**
  Esta disciplina foca na avaliação e seleção de soluções que satisfaçam a um determinado conjunto de requisitos; e no desenvolvimento de um projeto detalhado para que se possa implementar as soluções no produto ou componente de produto.
- **Implementação da Solução (*Implementing Development Tasks*)**
  Esta disciplina está relacionada à implementação da solução, realização de testes unitários, revisão de código e inclusão dos componentes implementados na base de código.
- **Construção do Produto (Building *a Product*)**
  Esta disciplina está relacionada à construção do código integrado (*build*).
- **Verificação dos Requisitos (Verifying *Requirements*)**
  Engloba as atividades de teste dos componentes da solução em relação aos seus requisitos.
- **Registro de Bugs (*Working with Bugs*)**
  Engloba as atividades de registro e monitoração dos *bugs* encontrados durante os testes.

## 1.5 Considerações finais

A partir do detalhamento dos processos descritos no capítulo, esperamos que o leitor tenha compreendido a filosofia por trás dos processos tradicionais, ajudando-o desta forma a desenvolver um senso critico para a escolha do processo que mais se adéque à sua empresa/organização na busca pela qualidade do software.

## 1.6 Tópicos de Pesquisa

Com o objetivo de se ter um processo de desenvolvimento de software bem definido, no contexto de Processos Tradicionais de desenvolvimento de software, alguns trabalhos de pesquisa podem ser desenvolvidos:

- **Implantação de modelos de qualidade de software com base em processos tradicionais**

  Empresas de software têm buscado fortemente a garantia da qualidade de seus processos de desenvolvimento, através de certificações e/ou avaliações formais com base em modelos de qualidade de software (ex: CMMI, MPS.BR), como forma de “garantir” a qualidade dos produtos de software gerados a partir desses processos. Neste contexto, os processos de desenvolvimento tradicionais apresentam grande aderência aos requisitos desses modelos de qualidade. Estudos experimentais podem ser realizados para verificar as contribuições positivas e negativas da implantação desses modelos com base nestes processos.

- **Comparação entre processos tradicionais e ágeis para o desenvolvimento de software**

  Estudos experimentais podem ser realizados para analisar as vantagens e desvantagens do uso de processos tradicionais versus processos ágeis no desenvolvimento de software.

- **Estudo sobre relações entre papéis funcionais dos processos tradicionais e o comportamento pessoal no trabalho em equipes de desenvolvimento de software**

  As equipes de software têm buscado modelos consistentes para a montagem de equipes com melhor desempenho. A união correta entre o comportamento do indivíduo no trabalho em grupo e sua função no processo de desenvolvimento tem tido uma atenção especial nas fábricas de software. Neste contexto, estudos experimentais podem ser realizados para analisar quais os perfis comportamentais mais adequados de profissionais para assumir determinados papéis em processos tradicionais.

## 1.7 Sugestão de Leitura

Para o leitor que queira conhecer o processo RUP, é indicado ler os seguintes livros:

- **Introdução ao RUP: Rational Unified Process**, de Phillippe Kruchten; e
- **Conheça o Rational Unified Process**, de Mauro Viana.

Para aprofundar-se nos aspectos gerenciais do RUP e o seu uso na implantação de normas, leia as seguintes monografias:

- **Rational Unified Process: uma abordagem gerencial.** Disponível em: http://www.de9.ime.eb.br/~tssouza/eng_soft/Trabalho%20RUP/Mono_RUP.pdf. Último Acesso em 15/11/2009.
- **Uso do Processo RUP na Implantação da ISO9000-3.** Disponível em: http://inf.unisul.br/~vera/egs/Rup_iso9000.pdf. Último Acesso em 15/11/2009

Para um melhor entendimento sobre o processo OpenUp, é indicado ler o artigo:

- **OpenUP – the best of two worlds**, de Bjorn Gustafsson. Disponível em: http://www.projectkoach.com/papers/OpenUP_2_worlds.pdf. Último acesso em 22/11/2009.

Para ter um entendimento sobre os modelos de processos, MSF *agile* e MSF CMMI, leia:

- O artigo **Simplifique projetos de equipe com modelos de processos,** de Brian A. Randell, 2008. Disponível em: http://msdn.microsoft.com/pt-br/magazine/dd221363.aspx, Último acesso em 22/11/2009; e
- O livro **Microsoft Solutions Framework (MSF),** de Marley Keeton Poderes, Jeff Carter, Geof Lory e Andrew McMurray.

## 1.8 Exercícios

1. Por que o RUP é considerado um processo tradicional? Justifique a sua afirmação.
2. Por que o RUP pode ser considerado um processo aderente ao modelo iterativo e incremental?

3. As fases do RUP são quatro. Fale de forma resumida sobre os objetivos primordiais de cada uma.
4. Quais são as disciplinas do RUP? Qual o papel de cada uma?
5. Quais são as diferenças relevantes entre o RUP e o OpenUp?
6. Fale sobre os princípios básicos do OpenUp.
7. Quais as disciplinas existentes no RUP que não se encontram no OpenUp? Fale sobre elas.
8. Quais são as fases do MSF? Detalhe cada uma.
9. Fale um pouco sobre o modelo de equipe do MSF.
10. Explique os princípios do MSF. Selecione os que você acha mais importantes e explique o porquê.

## 1.9 Referências Bibliográficas


Ambler, S. W (2010) "History of the Unified Process". Disponível em http://www.enterpriseunifiedprocess.com/essays/history.html. Último acesso em: 13/04/2010.

Araújo, M. L.. C. L de (2005) " Avaliando a Metodologia Pro.NET em relação ao MSF 4.0", Disponível em http://www.cin.ufpe.br/~tg/2005-1/mscla.pdf. Último acesso em: 12/10/2009.

[Boehm 1988] Boehm, B. (1988) A Spiral Model of Software Development and Enhancement Computer, Vol. 21, 5 (5), May 1988, pp. 61-72.

Cardim, I. C. (2006) "*Avaliando o Microsoft Solutions Framework for Agile Software Development em relação ao Extreme Programming"*. Disponível em http://www.cin.ufpe.br/~tg/2005-2/icc2.pdf. Último acesso em 05/09/2009.

Cunha, C. E. A da e Vasconcelos A. M. L de (2007) "Utilizando OpenUP/Basic para Desenvolvimento de Aplicações WEB". Disponível em http://www.cin.ufpe.br/~tg/2006-2/ceac.pdf. Último acesso em 31/05/2010.

Dantas, Fernando (2008) "Study Guide IBM Rational Unified Process – RUP 2003". Disponível em http://si.uniminas.br/~marcio/rup/Resumo_Livro_RUP_Made_Easy.pdf. Último acesso em 25/10/2010

[Humphrey 1995] Humphrey, W. S (1995) A Discipline for Software Engineering. Addison Wesley, Longman. pp. 242 .

Monteiro, J. M e Ybanez, M (2009) "*Integrando Metodologias Ágeis e Modelos de Maturidade de Software: Um Estudo de Caso"*. Disponível em http://www.infobrasil.inf.br/iConstructor/Custom/anais2009/Integrando%20Metodologias%20%C3%81geis%20e%20Modelos%20de%20Maturidade%20de%20Software%20Um%20Estudo%20de%20Caso.pdf. Último acesso em 05/11/2009.

Moraes, T (2006) *"Aplicação de padrões ao processo de desenvolvimento de software RUP"*, Disponível em http://dsc.upe.br/~tcc/20062/Monografia_TiagoMoraes.pdf. Último acesso em 08/11/2009.

Microsoft (2010) "The Engineering section of the MSF for CMMI Process Improvement". Disponível em http://msdn.microsoft.com/en-us/library/ee461541(v=VS.100).aspx. Último acesso em: 29/04/2010.

OpenUP (2007) Introduction to OpenUP (Open Unified Process). Disponível em http://www.eclipse.org/epf/general/OpenUP.pdf. Último acesso em 02/06/2010

Perrelli, H. (2009) “Visão Geral do RUP”. Disponível em http://www.cin.ufpe.br/~if717/slides/3-visao-geral-do-rup.pdf. Último acesso em 14/08/2009.

Piske, O. T (2003) “RUP-Rational Unified Process”. Disponível em http://www.angusyoung.org/arquivos/artigos/trabalho_rup.pdf. Último acesso em 16/11/2009.

Rational Software Corporation. (2001) “Rational Unified Process:Visão Geral”, Disponível em http://www.wthreex.com/rup/portugues/index.htm. Último acesso em 09/08/2009.

Vasques, R.C e Gomes, I. M (2007) “CMMI, MSF e Tecnologias Microsoft – uma visão de integração”. Disponível em http://www.isdbrasil.com.br/bco_conhecimento/artigo_cmmi_msf.pdf. Último acesso em 29/04/2010.