

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

Júlio César Henrique de Oliveira

**COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE
SOFTWARE: RUP, XP E ICONIX**

**NATAL/RN
DEZEMBRO/2007**

JÚLIO CÉSAR HENRIQUE DE OLIVEIRA

COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE
SOFTWARE: RUP, XP E ICONIX

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, como requisito obrigatório para obtenção do título de Especialista em Desenvolvimento de Sistemas Corporativos.

Orientador: Gleydson Lima, M.Sc.

Natal
2007

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Oliveira, Júlio César Henrique de
Comparativo entre processos de desenvolvimento de software: RUP,
XP e ICONIX / Júlio César Henrique de Oliveira. – Natal: FARN, 2007.
51f.

Orientador: Gleydson Lima (M.Sc.)
Monografia (Especialização em Desenvolvimento de Sistemas
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio
Grande do Norte.

1. Desenvolvimento de Softwares – Monografia. 2. Rational Unified
Process (RUP) – Monografia. 3. Extreme Programming (XP) – Monografia.
4. ICONIX – Monografia. I. Título.

RN/FARN/BC

CDU 004.4

RB. 30.704

FARN
BIBLIOTECA

Nº Reg

M000895

16/04/08

004.4

047e

2007

ex 01

JÚLIO CÉSAR HENRIQUE DE OLIVEIRA

COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE: RUP, XP E ICONIX

Monografia apresentada ao Curso de Especialização em Desenvolvimento de
Sistemas Corporativos da Faculdade Natalense para o desenvolvimento do Rio
Grande do Norte, aprovada em ____ de _____ de _____, pela
banca examinadora composta pelos seguintes membros:

Gleydson de Azevedo F. Lima

Gleydson Lima, M.Sc.
Orientador

RESUMO

Hoje as empresas de desenvolvimento de software buscam melhorias em seus processos, com o intuito de produzirem um produto com maior qualidade, em um menor tempo e custos reduzidos. Os processos de desenvolvimento de software vêm para ajudar as empresas a conseguirem estes objetivos. Este trabalho mostra os conceitos, as fases, as melhores práticas e os ciclos de vida que envolvem alguns dos melhores processos de desenvolvimento da atualidade, que são o *Rational Unified Process* (RUP), o *Extreme Programming* (XP) e o ICONIX. Neste trabalho será feito um comparativo entre estes processos, onde serão abordados aspectos como custos, responsabilidades, codificação, documentação, ferramentas, dentre outros, servindo de referência para as empresas que queiram comparar estes processos antes de sua implementação.

Palavras-Chave: Processos de Desenvolvimento de Software. Rational Unified Process. Extreme Programming. ICONIX.

ABSTRACT

Today businesses of developing software seek improvements in its processes, with the aim of producing a product with higher quality, in a shorter time and reduced costs. The process of developing software come to help companies achieve these goals. This work shows the concepts, the phases and best practices and the cycles of life involving some of the best development processes today's, which are the Rational Unified Process (RUP), the Extreme Programming (XP) and ICONIX. This work will be made a comparison between these process, which will be discussed aspects such as costs, responsibilities, coding, documentation, tools, among others, serving as a reference for companies wishing to compare these processes before implementation.

Keywords: Process of Developing Software. Rational Unified Process. Estreme Programming. ICONIX.

SUMÁRIO

1 INTRODUÇÃO	05
2 RATIONAL UNIFIED PROCESS (RUP)	07
2.1 MELHORES PRÁTICAS DO RUP	07
2.2 ESTRUTURA DO RUP	09
2.2.1 Elementos do RUP	10
2.2.1.1 Papéis	10
2.2.1.2 Atividades	11
2.2.1.3 Artefatos	12
2.2.1.4 Fases	13
2.2.1.4.1 Iniciação	14
2.2.1.4.2 Elaboração	15
2.2.1.4.3 Construção	15
2.2.1.4.4 Transição	16
2.2.1.5 Disciplinas	17
3 EXTREME PROGRAMMING (XP)	18
3.1 VALORES DO XP	18
3.1.1 Feedback	18
3.1.2 Comunicação	19
3.1.3 Simplicidade	20
3.1.4 Coragem	20
3.2 PRÁTICAS DO XP	21
3.2.1 Cliente Presente	21
3.2.2 Jogo do Planejamento	22
3.2.3 Reunião em Pé	25
3.2.4 Programação em Par	26
3.2.5 Refactoring	27
3.2.6 Desenvolvimento Guiado pelos Testes	27
3.2.7 Código Coletivo	28
3.2.8 Código Padronizado	28
3.2.9 Releases Curtos	29
3.3 CARACTERÍSTICAS DA EQUIPE	30

3.4	DOCUMENTAÇÃO DO PROJETO.....	31
4	ICONIX.....	33
4.1	MODELO DE DOMÍNIO	34
4.2	MODELO DE CASO DE USO.....	35
4.3	ANÁLISE ROBUSTA	36
4.4	DIAGRAMA DE SEQUÊNCIA.....	39
4.5	DIAGRAMA DE CLASSE.....	40
4.6	GERANDO CÓDIGO	41
5	COMPARATIVO DAS METODOLOGIAS	42
5.1	CUSTO DOS PROCESSOS.....	42
5.2	PAPÉIS.....	43
5.3	CODIFICAÇÃO	45
5.4	DOCUMENTAÇÃO	46
5.5	FERRAMENTAS.....	47
5.6	DESENVOLVIMENTO ORIENTADO POR CASO DE USO	47
5.7	PROCESSO MAIS ADEQUADO PARA UMA PEQUENA EMPRESA.....	48
5.8	PROCESSO MAIS ADEQUADO PARA O SETOR PÚBLICO	48
6	CONCLUSÃO	49
	REFERÊNCIAS	50

1 INTRODUÇÃO

Hoje as empresas produtoras de software estão procurando cada vez mais desenvolver seus produtos com menor custo, aliado a uma melhor qualidade e uma maior produtividade. Para que estes pontos sejam atingidos, é de fundamental importância que o processo de desenvolvimento seja definido e posto em prática.

Segundo Paula Filho (2001), um processo é um conjunto de passos parcialmente ordenados, constituídos por atividades, métodos, práticas e transformações, usado para atingir uma meta.

Na engenharia de software, um processo de desenvolvimento de software tem por finalidade obter um produto de software com qualidade, que atenda aos requisitos do sistema e que cumpra corretamente o contrato de desenvolvimento. Em um processo de desenvolvimento de software, o ponto de partida para a arquitetura de um processo é a escolha de um modelo de ciclo de vida.

Modelos de ciclo de vida descrevem os grupos de atividades e como elas se relacionam dentro do processo, alguns modelos descrevem ainda como e quando mover de uma atividade para outra bem como os resultados que podem ser produzidos de uma atividade para outra.

A engenharia de software produziu ao longo do tempo vários modelos de ciclo de vida, como o modelo de cascata, espiral e prototipagem evolutiva. Antes do modelo de cascata ser proposto, em 1970, não existia concordância de qual modelo adotar no desenvolvimento de software. Desde então muitos modelos têm sido propostos refletindo assim a quantidade de interpretações e caminhos que podem ser tomadas no desenvolvimento de software (PAULA FILHO, 2001).

No modelo de ciclo de vida em cascata, as principais atividades são executadas em seqüência, por exemplo, a análise de todos os requisitos do sistema deve ser finalizada antes de começar o processo desenho do sistema. É um processo rígido e burocrático, em que as atividades de requisitos, análise e desenho têm de ser muito bem dominadas, pois não são permitidos erros, já que um erro em umas dessas atividades poderá gerar problemas no sistema. O modelo em cascata é de pouca visibilidade para o cliente, já que este só recebe o produto no final do projeto.

O modelo em espiral baseia-se em uma série de iterações. Cada nova iteração corresponde a uma volta na espiral, que pode ser baseada num modelo

diferente e pode ter diferentes atividades. Este modelo permite que as idéias e o progresso sejam avaliados constantemente, bem como análise dos riscos e identificação de erros precocemente. Apesar de ser um modelo muito interessante, o modelo em espiral é muito complexo e requer atenção e conhecimento por parte dos gerentes de projeto. Alguns processos de desenvolvimento de software surgiram de uma variação entre o modelo em cascata e o modelo em espiral, surgindo modelos fundamentalmente iterativos e incrementais, como mostra a figura 1, estes modelos são: *Rational Unified Process* (RUP), o XP (*Extreme Programming*), ICONIX, dentre outros.

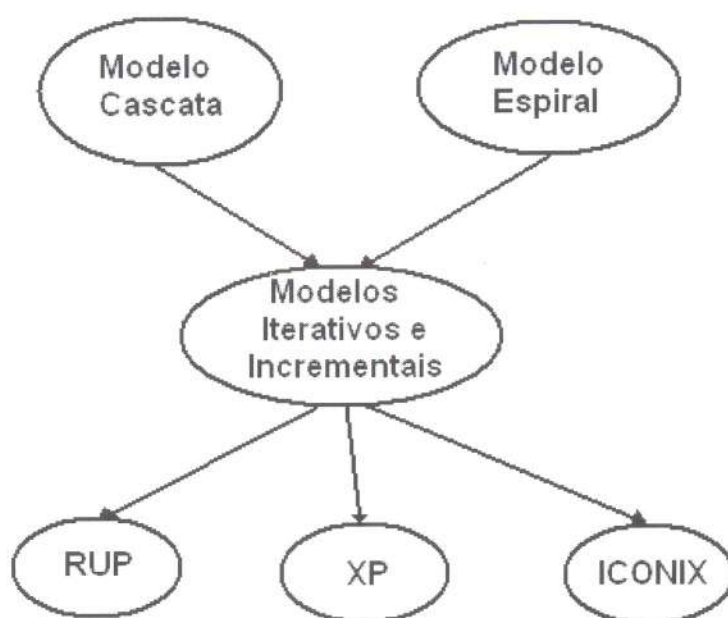


Figura 1 – Surgimento dos Modelos Iterativos e Incrementais
Fonte: (Dados da Pesquisa)

A proposta deste trabalho é fazer um comparativo entre três processos de desenvolvimento de software: RUP, XP e ICONIX. Do Capítulo 2 ao 4 serão apresentados os conceitos e características destes processos. No Capítulo 5 será apresentado o comparativo entre os processos, enfatizando os custos, pontos fortes e pontos fracos de cada processo. No Capítulo 6 serão as considerações finais sobre estes três processos.

2 RATIONAL UNIFIED PROCESS (RUP)

O *Rational Unified Process* (RUP) é um processo de desenvolvimento de software que fornece técnicas para a equipe de desenvolvimento aumentar a sua produtividade, como definir tarefas e responsabilidades para cada membro da equipe. Seu objetivo é assegurar a produção de software de alta qualidade que satisfaça as necessidades dos usuários finais dentro dos prazos estabelecidos e dos custos orçados.

O RUP pode ser adaptado e estendido para compor as necessidades de qualquer organização que o esteja adotando. Ele captura as melhores práticas do desenvolvimento de software, como: desenvolver software iterativamente; gerenciar os requisitos do software; utilizar uma arquitetura baseada em componentes; modelar o software visualmente, utilizando UML; verificar continuamente a qualidade do software; e controlar as mudanças do software, mantendo-o sincronizado com os requisitos e artefatos.

Segundo Kruchten (2003), mais de 1000 empresas usavam o RUP no final de 1999, dentre elas a Ericson, Alcatel, Intel, Oracle, Visa. Essas empresas estavam utilizando o RUP nos mais variados domínios de aplicações e em grandes e pequenos projetos de software, mostrando a versatilidade e ampla aplicabilidade que o RUP proporciona.

2.1 MELHORES PRÁTICAS DO RUP

O RUP incorpora as melhores práticas de desenvolvimento de software como:

- **Desenvolvimento Iterativo**

O desenvolvimento iterativo permite suavizar os riscos com antecedência, porque a integração é geralmente o único momento em que os riscos são descobertos. Como expomos as iterações mais cedo, passamos por todos os componentes de processo, exercitando muitos dos aspectos do projeto, como ferramentas, habilidades das pessoas, aceitação do produto pelo cliente, dentre outros.

Os desenvolvedores podem aprender no caminho, e suas várias habilidades e especialidades são empregadas mais completamente durante todo o ciclo de vida.

Testadores começam a testar mais cedo, os responsáveis pela documentação começam a documentar mais cedo também, e assim por diante.

O próprio processo de desenvolvimento pode ser melhorado e refinado a cada iteração. A avaliação ao término de uma iteração não só olha para o estado do projeto de uma perspectiva de produto/prazo como também analisa o que deveria ser mudado no processo, para fazer como que a execução seja melhor na próxima iteração.

- Gerenciamento de Requisitos

O gerenciamento de requisitos é fundamental para uma boa metodologia de desenvolvimento de software e pré-requisito para o sucesso de um software, uma vez que a falta de compreensão do comportamento do sistema planejado e deslize de requisitos são fatores comuns em projetos mal sucedidos.

Corrigir erros de requisitos mal definidos é muito mais caro, e reduzir estes erros com antecedência, pelo ciclo de desenvolvimento, corta custos de projeto e previne demoras.

O gerenciamento de requisitos proporciona um envolvimento precoce do cliente no processo, ajudando a assegurar que a aplicação satisfaça suas necessidades. Requisitos bem definidos e bem gerenciados constroem uma compreensão comum das necessidades e compromissos do projeto entre os interessados.

- Arquitetura Baseada em Componentes

Segundo Kruchten (2003), um componente de software pode ser definido como um pedaço não específico de software, um módulo, um pacote ou um subsistema que cumpre uma função clara, tem um limite bem definido e pode ser integrado numa arquitetura bem definida.

Definindo uma arquitetura baseada em componentes conseguimos a identificação, o isolamento, o projeto, o desenvolvimento, o teste dos componentes individualmente ou integrados, formando todo o sistema.

Além disso, alguns desses componentes podem ser desenvolvidos para reutilização, especialmente os que fornecem soluções comuns a uma gama extensiva de problemas comuns. Estes componentes reutilizáveis são geralmente bibliotecas de classes, e formam a base de reutilização dentro de uma organização, aumentando assim a qualidade do produto, a produtividade global dos desenvolvedores, além de permitir redução nos custos.

2.2.1 A abordagem iterativa permite aos desenvolvedores identificar os componentes progressivamente e decidir quais deles desenvolver, reutilizar ou comprar.

2.2.2 - Desenvolvimento Orientado por Caso de Uso

O RUP é uma abordagem dirigida a caso de uso, que significa que os casos de uso definidos para o sistema são a fundamentação para o resto do processo de desenvolvimento. Casos de uso exercem um papel principal em vários fluxos de processo, especialmente requisitos, projeto, teste e gerenciamento. Os casos de uso são fundamentais para a modelagem do domínio de negócio.

2.2 ESTRUTURA DO RUP

A estrutura do processo RUP é representada por 2 dimensões, como mostra a figura 2:

- o eixo horizontal, que representa o tempo e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve. Este eixo representa o aspecto dinâmico do processo e é expresso em termos de fases, iterações e marcos.

- o eixo vertical, que representa o aspecto estático do processo, é descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis do processo.

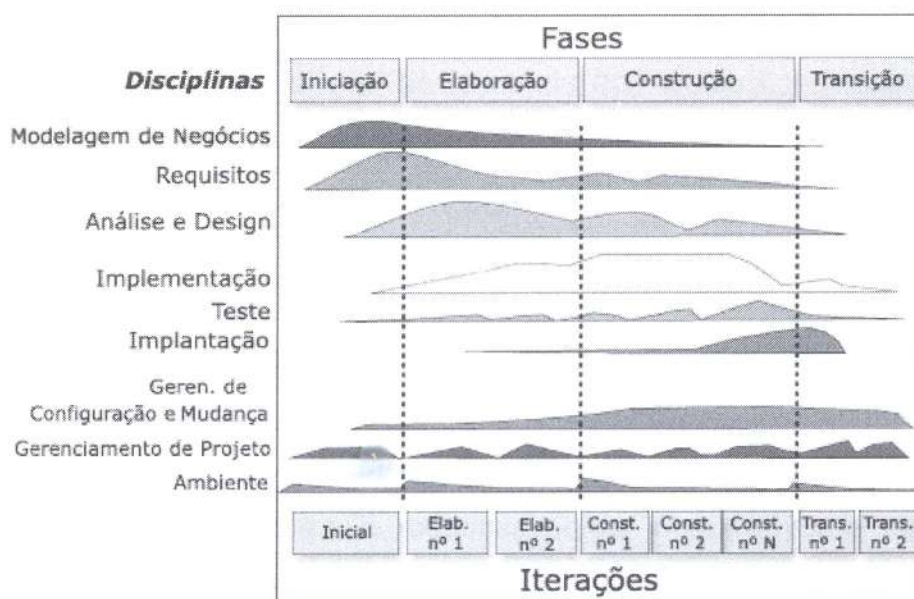


Figura 2 – Visão geral do RUP
Fonte: (RATIONAL..., 2007)

2.2.1 Elementos do RUP

O RUP possui cinco elementos principais que compõem toda sua estrutura: papéis, artefatos, atividades, fluxo de trabalho e disciplinas.

2.2.1.1 Papéis

Um papel no RUP representa um comportamento e as responsabilidades de um indivíduo ou um grupo de indivíduo dentro de uma equipe. O comportamento é expresso em termos de atividades que o indivíduo desempenha em determinada fase do projeto, e cada indivíduo é associado a um conjunto de atividades.

Os papéis não são pessoas, eles apenas descrevem o comportamento das pessoas no negócio e quais são suas responsabilidades.

Segue alguns exemplos de papéis no RUP:

- Analista de Sistema

Um indivíduo atuando como analista de sistema, coordenando o surgimento de requisitos e modelagem de casos de uso.

- Analista de Teste

Um indivíduo atuando como analista de teste, que identifica e posteriormente define os testes necessários, monitora a abrangência dos testes e avalia a qualidade geral obtida ao testar os Itens de Teste-alvo. Este papel também envolve a especificação dos Dados de Teste necessários e a avaliação do resultado dos testes conduzidos em cada ciclo de teste. Às vezes, este papel também é denominado *Designer* de Teste ou considerado parte do papel Testador.

- Implementador

Um indivíduo que é responsável por desenvolver e testar componentes de acordo com os padrões adotados para o projeto, para fins de integração com subsistemas maiores. Quando é necessário criar componentes de teste, como *drivers* ou *stubs*, para possibilitar a realização dos testes, o implementador também é responsável por desenvolver e testar esses componentes e os subsistemas correspondentes.

- Gerente de Projeto

Um indivíduo atuando como gerente de projeto que aloca recursos, redefine as prioridades, coordena interações com clientes e usuários e geralmente mantém a equipe do projeto concentrada na meta certa. O gerente de projeto também estabelece um conjunto de práticas que garantem a integridade e a qualidade dos artefatos do projeto.

- Projetista

Um indivíduo atuando como desenhista, que define as responsabilidades, operações, atributos e relações de uma ou mais classes e determina como elas deveriam ser ajustadas ao ambiente de desenvolvimento.

- Projetista de Teste

Um indivíduo atuando como projetista de teste, responsável pelo planejamento, projeto, desenvolvimento e avaliação de testes, incluindo geração do plano de teste e modelo de teste, desenvolvimento dos procedimentos de teste e avaliação da cobertura de teste, resultados e efetividade.

2.2.1.2 Atividades

Papéis têm atividades que definem o trabalho que eles desempenham. Uma atividade é uma unidade do trabalho que um indivíduo, naquele papel, pode ser pedido a desempenhar, e isso produz um resultado significativo no contexto do projeto.

A atividade tem um propósito claro, normalmente expresso em termos de criar ou atualizar artefatos, como um modelo, uma classe ou um plano de teste. Toda atividade é atribuída a um papel específico.

A figura 3 representa uma visão geral de algumas atividades, o indivíduo que assume o papel de Analista do Processo de Negócio é responsável pelas atividades de Avaliar Organização-Alvo, Definir e Ajustar Metas, Manter papéis de Negócio, Definir a Arquitetura de Negócio, dentre outras atividades. Enquanto o Designer de Negócios é responsável por Detalhar um Caso de Uso de Negócio, Localizar Entidades e Trabalhadores de Negócio, Definir Requisitos de Automatização, dentre outras.



Figura 3 – Visão geral de atividades
Fonte: (RATIONAL..., 2007)

2.2.1.3 Artefatos

As atividades possuem artefatos de entrada e de saída. Um artefato é um produto de trabalho final ou inicial que é produzido, modificado e usado durante o projeto. Os artefatos são usados como entrada pelos trabalhadores para desempenhar uma atividade e são o resultado ou a saída de tais atividades.

Segue alguns exemplos de artefatos do RUP:

- Um documento, como um caso de uso de negócio
- Um modelo, como um modelo de caso de uso ou modelo de projeto
- Código-Fonte
- Executáveis
- Componente
- Plano de Teste
- Classe de Teste

Na figura 4 vemos os artefatos gerados pelo Gerente de Testes, Testador, Analista de Teste, Designer de Teste, Designer e Implementador.

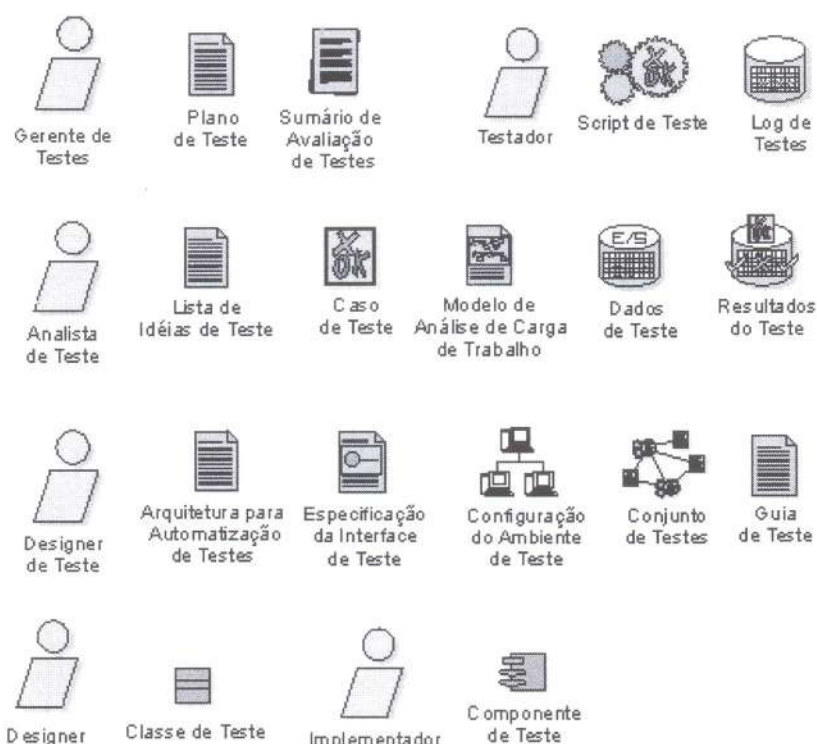


Figura 4 – Visão geral de artefatos
Fonte: (RATIONAL..., 2007)

Os artefatos podem ser compostos por outros artefatos. Por exemplo, o artefato Modelo de Projeto contém muitas classes; o Plano de Desenvolvimento de Software contém vários outros planos: Plano de Pessoal, Plano de Iteração, etc.

2.2.1.4 Fases

O ciclo de vida do software no RUP está dividido em quatro fases sequenciais: iniciação, elaboração, construção e transição. Cada fase é concluída por um marco principal, ou seja, cada fase é basicamente um intervalo de tempo entre dois marcos principais. No final de cada fase é executada uma avaliação para checar se os objetivos da fase foram alcançados, como mostra a figura 5. Agora veremos em mais detalhes sobre cada fase do ciclo de vida do RUP.

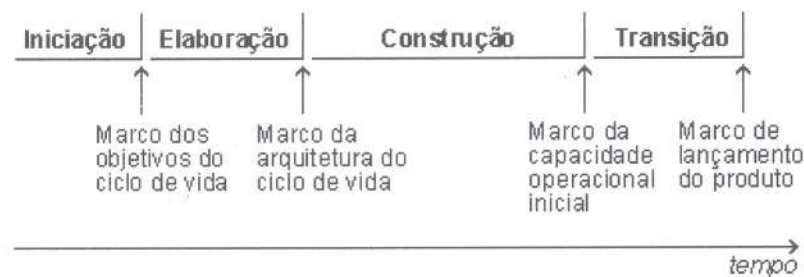


Figura 5 – Fases e Marcos de um projeto RUP
Fonte: (RATIONAL..., 2007)

2.2.1.4.1 Iniciação

A fase de iniciação especifica a visão do produto final e seu domínio de negócio, e define a extensão do projeto. Esta fase é concluída pelo marco objetivo do ciclo de vida.

A iniciação tem os seguintes objetivos a serem alcançados:

- Estabelecer o escopo principal do *software*, ou seja, devem-se definir os critérios de aceitação do sistema e o que deve ser feito ou não no *software*.
- Discriminar os casos de uso críticos do sistema, ou seja, aqueles que serão desenvolvidos primeiro e que servirão como base para outros casos de uso.
- Propor pelo menos uma opção de arquitetura para alguns cenários básicos.
- Estimar o custo geral do projeto
- Estimar os riscos do projeto
- Preparar o Ambiente de suporte para o projeto

As atividades essenciais desta fase são:

- Formular a extensão do projeto, ou seja, captar o contexto e os requisitos mais importantes e restrições de forma que derivemos critérios de aceitação para o sistema.
- Planejar e preparar um caso de uso de negócio e avaliar alternativas para o gerenciamento de riscos, fornecendo pessoal, plano de projeto e intercâmbios entre custo, prazo e rentabilidade.

- Sintetizar uma arquitetura candidata e avaliar decisões de fazer/comprar/reutilizar, de forma que os custos, prazos e recursos possam ser calculados, ou seja, fazer um esboço da arquitetura e para que decisões a cerca da construção/compra/reutilização de componentes sejam tomadas.

2.2.1.4.2 Elaboração

A fase de elaboração planeja as atividades necessárias e recursos exigidos, especifica as características e projeta a arquitetura do sistema. Esta fase é concluída pelo marco arquitetura do ciclo de vida.

Na elaboração é criada a *baseline* para a arquitetura do sistema, fornecendo uma base estável para a fase de construção. Os principais objetivos desta fase são:

- Definir, validar e delinear a arquitetura do sistema.
- Tratar todos os riscos do ponto de vista da arquitetura do projeto.
- Delinear a visão de um todo do projeto

As atividades essenciais da fase de elaboração são:

- A visão do projeto é elaborada, e é estabelecida uma compreensão sólida dos casos de uso mais críticos, que dirigem as decisões arquitetônicas e de planejamento.
- São elaborados os processos, a infra-estrutura e o ambiente de desenvolvimento.
- A arquitetura é elaborada e os componentes são selecionados. São avaliados componentes potenciais, e as decisões de compra, construção e reutilização determinam o custo da fase de construção.

2.2.1.4.3 Construção

Durante a fase de construção, todos os componentes e características da aplicação são desenvolvidos, integrados ao produto e conseqüentemente testados. Esta fase é, de certo modo, um processo industrial no qual é colocada ênfase em gerenciar recursos e controlar operações para aperfeiçoar custos, prazos e qualidade.

Os objetivos primários da fase de construção são:

- Minimizar custos de desenvolvimento, aperfeiçoando recursos e evitando retrabalhos;
- Alcançar a qualidade desejada o quanto antes;
- Produzir versões alfa, beta e outros lançamentos para testes;

As atividades essenciais da fase de construção são:

- Administração e controle dos recursos e otimização dos processos;
- Desenvolvimento completo de componentes e testes das funcionalidades do sistema frente aos requisitos;

O resultado da fase de construção é um produto pronto para seus usuários finais. No mínimo, consiste em:

- O produto de software (executáveis, programas de instalação, etc.);
- Os manuais de usuário;
- Descrição das funcionalidades que estão saindo no release atual;

2.2.1.4.4 Transição

O propósito da fase de transição é levar o produto de software à comunidade de usuários. Após os usuários começarem a utilizar o sistema, normalmente surgem necessidades de desenvolvimento de novas funcionalidades ou que sejam corrigidos alguns problemas. Esta fase inclui:

- Teste beta para validar o sistema novo contra as expectativas dos usuários;
- Operações paralelas com o sistema legado que o projeto está substituindo;
- Conversão de bancos de dados;
- Treinamento de usuários e equipe de suporte;
- Saída do produto para a equipe de marketing e vendas;

A fase de transição focaliza as atividades exigidas para colocar o software nas mãos dos usuários. Geralmente, esta fase inclui várias iterações, inclusive lançamentos beta, lançamentos de novos recursos, correção de *bugs* e lançamentos de atualizações. Muito esforço é empregado desenvolvendo a documentação do usuário e treinamento de suporte. Porém nesta fase os questionamentos dos

usuários deveriam ser limitados a configuração do sistema, instalação e questões de utilidade.

2.2.1.5 Disciplinas

Uma disciplina é uma coleção de atividades relacionadas que fazem parte de um contexto comum em um projeto. As disciplinas proporcionam um melhor entendimento do projeto sob o ponto de vista tradicional de um processo em cascata. A separação das atividades em disciplinas torna a compreensão das atividades mais fácil, porém dificulta mais o planejamento das atividades. O RUP possui nove disciplinas, divididas em disciplinas do processo e de suporte. As disciplinas de processo são: modelagem de negócios, requisitos, análise e *design*, implementação, teste e implantação. As de suporte são: configuração e gerenciamento de mudanças, gerenciamento de projeto, e ambiente.

Neste capítulo vimos os conceitos que envolvem o RUP, sua importância para o desenvolvimento de software em projetos de grande porte, onde a equipe de desenvolvimento é grande e haja a disponibilidade de recursos. Agora passaremos a mostrar os conceitos que envolvem o *Extreme Programming* (XP).

3 EXTREME PROGRAMMING (XP)

O Extreme Programming (XP), é um processo de desenvolvimento de software voltado para projetos com equipes pequenas, com até 12 desenvolvedores, cujos requisitos são vagos e mudam com frequência, o desenvolvimento é orientado a objetos e incremental, onde o sistema começa a ser desenvolvido logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

Segundo [TELES], o XP está enquadrado na categoria de processos de desenvolvimento chamada como *Processos Ágeis de Desenvolvimento*. Estes processos compartilham a premissa que o cliente aprende sobre suas necessidades, na medida em que é capaz de manipular o sistema que está sendo produzido. Com base no *feedback* do sistema ele re-avalia as suas necessidades e prioridades, gerando mudanças que devem ser incorporadas ao software. O aprendizado é importante, porque permite que o cliente direcione o desenvolvimento de modo que a equipe produza sempre aquilo que tem maior valor para o seu negócio.

3.1 VALORES DO XP

O XP baseia-se em quatro valores fundamentais: *feedback*, comunicação, simplicidade e coragem.

3.1.1 Feedback

O *feedback* é o retorno que o cliente fornece à equipe de desenvolvimento quando surge alguma necessidade em relação ao sistema, quando aprende mais sobre os requisitos ou quando aprende mais a forma de como foram desenvolvidos.

Assim como os retornos que os clientes passam para a equipe de desenvolvimento, existe o caminho inverso onde a equipe de desenvolvimento fornece ao cliente estimativas, apontam riscos técnicos, sugerem alternativas de design, entre outras. Este retorno é a manifestação do aprendizado da equipe de desenvolvimento sobre aquilo que precisa ser feito.

O cliente aprende como o sistema pode contribuir com ele da melhor forma possível e realimenta a equipe de desenvolvimento no decorrer de dias ou semanas, ao invés de meses ou anos. A equipe de desenvolvimento aprende a melhor forma

de projetar o sistema e o realimenta com este aprendizado em segundos ou minutos, ao invés de dias, semanas ou meses (TELES, 2004).

3.1.2 Comunicação

Para que o *feedback* exista em um projeto de software, é necessário que a comunicação esteja presente de forma muito clara. A grande maioria dos sistemas é desenvolvida em equipe, sendo assim, os componentes da equipe e o cliente precisam trocar idéias e informações para que o software ganhe forma e atinja os objetivos desejados.

Podemos transmitir uma idéia ou pensamento de diversas formas, como face-a-face, por telefone, por email, mensagens instantâneas, entre outras. A comunicação face-a-face é o meio de comunicação mais completo que existe, uma vez que o nosso interlocutor pode interpretá-la levando em conta o conteúdo da fala, os nossos gestos, a nossa expressão facial, o tom de voz, entre outros. É importante notar que, em uma comunicação face-a-face, existe grande riqueza de elementos que facilitam a compreensão da nossa mensagem, além de uma dinâmica que viabiliza questionamentos e respostas. Os outros meios de comunicação não apresentam todas as vantagens da comunicação face-a-face, como por exemplo, o meio escrito não apresenta recursos como tom de voz, gestos, expressão facial, possibilidade de tirar dúvidas no momento do recebimento da informação. Sendo assim, torna-se mais difícil a compreensão uma vez que o cliente ou a equipe precisam se esforçar para entender o que está escrito na mensagem.

Tradicionalmente, as equipes de *software* se utilizam extensivamente do meio escrito para comunicar idéias. Devido às deficiências deste meio de comunicação, as idéias são freqüentemente interpretadas de forma incorreta. O XP procura evitar este problema utilizando-se fortemente do meio de comunicação mais rico que existe, a conversa face-a-face. Ou seja, procura explorar tanto quanto possível a interação direta entre as pessoas envolvidas no projeto, de modo a diminuir as falhas de comunicação e evitar re-trabalhos desnecessários, frutos destas falhas.

3.1.3 Simplicidade

A filosofia do desenvolvimento ágil, dedica atenção especial ao *feedback*. Ou seja, considera fundamental obter *feedback* de todas as partes envolvidas no projeto o quanto antes, de modo que seja possível corrigir os seus rumos com pequenas e freqüentes intervenções ao longo do seu ciclo de vida.

É importante que as ações de cada membro da equipe sejam simples e restritas. Permitindo que, logo após a execução de uma ação, possa-se obter *feedback* sobre ela rapidamente e fazer pequenos ajustes caso estes sejam necessários.

Quando um cliente solicita uma funcionalidade, esta deve ser desenvolvida apenas com o suficiente para que o cliente tenha seu pedido atendido e que possa validar se foi atendido da forma como gostaria. Os desenvolvedores devem criar a funcionalidade da forma mais simples possível, evitando fazer trabalho especulativo.

Segundo Teles (2004), trabalho especulativo é aquele executado utilizando premissas sobre as quais não se tem total certeza. Ou seja, o desenvolvedor ao programar a funcionalidade, se depara com uma dúvida cuja resposta está fora de seu alcance imediato. Neste caso, ele assume uma resposta que lhe parece razoável e desenvolve a funcionalidade levando-a em conta. Mas, aquilo que ele assumiu como sendo a resposta verdadeira freqüentemente está incorreto. A única maneira de descobrir a resposta certa seria levar a dúvida até o cliente, que nem sempre está por perto.

3.1.4 Coragem

A adoção do XP pela equipe de desenvolvimento exige muita coragem, principalmente em pontos que contrariam os tradicionais processos de desenvolvimento, como:

- *desenvolver o software de forma incremental* utilizando um ciclo de vida em espiral, onde partes que já haviam sido implementadas são revisitadas várias vezes para acomodar novas funcionalidades ou alterar as já existente de acordo com o *feedback* do cliente;

- *manter o sistema simples*, é o conceito de simplicidade que vimos anteriormente onde apenas deve ser implementado aquilo solicitado pelo cliente, uma vez que feito um trabalho especulativo este irá gerar custos e perdas de produtividades desnecessárias;
- *permitir que o cliente priorize as funcionalidades*, permitindo que o cliente defina a ordem de desenvolvimento das funcionalidades que vão lhe dar maior retorno;
- *desenvolvimento em par*, uma das maiores quebras de paradigma em relação aos processos tradicionais, onde os desenvolvedores formam pares;
- *refactoring* é uma prática essencial do XP que é utilizada para melhorar a qualidade do código do sistema. Significa fazer melhorias no código sem alterar a funcionalidade que ele implementa;
- *testes automatizados*, o XP recomenda que a equipe de desenvolvimento crie testes unitários para as classes do sistema;
- *expor o código a todos os membros da equipe*, a prática da programação em par e código coletivo fazem com que o código escrito por um desenvolvedor seja visto e manipulado por outras pessoas da equipe de desenvolvimento. Portanto, os desenvolvedores ficam mais expostos a eventuais críticas e avaliações por parte dos seus colegas;

3.2 PRÁTICAS DO XP

3.2.1 Cliente Presente

O XP sugere que o cliente esteja presente durante o dia-a-dia do projeto. A sua participação contribui para o sucesso do projeto, enquanto a sua ausência é um sério fator de risco podendo levar o projeto ao fracasso.

A participação do cliente viabiliza a simplicidade do processo (TELES, 2004). Uma vez que o projeto passa a ser conduzido através de uma série de pequenos ajustes e não através de mudanças bruscas.

As funcionalidades do sistema são descritas brevemente em cartões e recebem o nome de estórias. Estórias são convites ao diálogo, onde o

desenvolvedor, ao começar a implementar uma estória, chama o cliente para um diálogo para entender melhor a funcionalidade. Essa é uma das razões pelas quais é essencial que o cliente esteja presente no dia-a-dia do desenvolvimento.

Quando os desenvolvedores começam a implementar as funcionalidades, novas dúvidas podem surgir. A presença do cliente permite que as dúvidas sejam respondidas rapidamente, contribuindo com a agilidade do processo e evitando o trabalho especulativo. A presença do cliente ao longo do desenvolvimento viabiliza o ciclo contínuo de *feedback* entre ele e a equipe de desenvolvimento.

3.2.2 Jogo do Planejamento

O jogo do planejamento é umas das principais práticas do XP, que consiste no planejamento para assegurar que a equipe de desenvolvimento esteja sempre trabalhando naquilo que irá gerar maior valor para o cliente. Este planejamento é feito diversas vezes ao longo do projeto, para que todos tenham a oportunidade de revisar as prioridades.

Para assegurar todo este planejamento o XP assume algumas premissas que devem ser implementadas, como:

- Dividir as responsabilidades

Segundo Beck (2004), a chave para a gestão de um projeto de software é o equilíbrio de poderes entre o cliente e a equipe de desenvolvimento. Por esta razão, o XP procura separar claramente o papel do cliente e da equipe de desenvolvimento, para assegurar que o cliente tome todas as decisões de negócio, enquanto a equipe de desenvolvimento cuida das decisões técnicas.

- Escrever Estórias

Todas as funcionalidades do sistema devem ser descritas em pequenos cartões e estas estórias devem ser escritas pelo próprio cliente, com suas próprias palavras. Uma vez que ao escrever uma estória, o cliente é forçado a pensar melhor na funcionalidade, pois ele formaliza o pensamento e analisa melhor o assunto sobre o qual irá tratar.

A equipe de desenvolvimento utiliza os cartões para implementar as funcionalidades. Quando uma história é um pouco mais longa, podendo levar semana para ser implementada, esta é dividida em vários outros cartões e redistribuída para o resto da equipe.

- Estimar os custos das Estórias

Para estimar uma história, a equipe de desenvolvimento precisa adotar uma unidade de medida que será usada para todas as histórias.

Os desenvolvedores costumam estimar indicando a quantidade de horas que uma funcionalidade irá consumir. As estimativas em XP não se baseiam em horas desenvolvimento porque isso pode gerar distorções que prejudicam a qualidade das estimativas.

O XP utiliza o conceito de dia ideal de desenvolvimento, que representa um dia no qual o desenvolvedor só trabalha na implementação de funcionalidades, sem ter que se preocupar com nenhuma outra atividade. A equipe refere a este dia como sendo um ponto. Um ponto é uma unidade de medida utilizada para estimar as histórias.

Em cada cartão (estórias), são colocados os pontos estimados para implementação no canto superior esquerdo, e após a implementação são colocados os pontos consumidos no canto superior direito. Permitindo assim um comparativo entre os pontos estimados e os realmente consumidos, servindo como base de conhecimento para outras estimativas.

- Planejar os *releases*

O XP trabalha com o objetivo de gerar valor para o cliente ao longo do projeto. Para fazer isso, é importante que o software seja entregue de forma incremental, de modo que após cada entrega o cliente possa começar a utilizar o sistema e obter os benefícios que ele oferece. Esta entrega recebe o nome de *release*.

Os projetos em XP costumam ser divididos em *release*, de modo que cada release implemente um conjunto de funcionalidades que possui um valor bem definido para o cliente.

Projetos em XP procuram trabalhar com o conceito de *releases* pequenos, isto é, os *releases* têm curta duração, devem ter em torno de 2 meses. Trabalhar com *releases* pequenos é importante porque permite que a equipe entregue algo que tem valor para o cliente rapidamente. Com isso, ele pode logo começar a usufruir os benefícios do software, o que serve como um incentivo para que ele dê prosseguimento ao projeto. A equipe de desenvolvimento, por sua vez, passa a ter a oportunidade de receber *feedback* dos usuários finais do sistema, o que permite que ela faça ajustes, de modo a aprimorar a qualidade dos *releases* posteriores.

- Planejar as iterações

No XP, cada *release* é dividido em um determinado número de iterações. Uma iteração é basicamente um pequeno espaço de tempo dedicado para a implementação de um conjunto de estórias. Normalmente, uma iteração pode variar de uma a três semanas dependendo das características do projeto. O XP sugere iterações de duas semanas, que representa um meio termo, não são nem tão curtas e nem tão longas para a maioria dos projetos. Uma vez definido o tempo das iterações, estas devem manter-se inalteradas ao longo de todo projeto.

O início de cada iteração é caracterizado por uma reunião que recebe o nome de reunião de planejamento da iteração. Nesta reunião, o cliente e os desenvolvedores definem as estórias que serão implementadas na iteração que se inicia. Para que esta reunião ocorra, eles precisam saber quantos pontos espera-se que a equipe será capaz de implementar. Esta quantia é conhecida como velocidade de iteração. Não se deve considerar nem o primeiro dia da iteração nem o último dia, pois como visto o primeiro dia é para a reunião de planejamento da iteração e o último dia é para o encerramento da iteração.

- Encerrar as iterações

O último dia da iteração é reservado para uma reunião de encerramento. Nela, o cliente executa os testes de aceitação que foram escritos para as histórias da iteração. O objetivo é que ele valide todo o sistema e verifique se o resultado da iteração está satisfatório. Utilizando o sistema ele poderá detectar eventuais erros, bem como aprender mais sobre os requisitos e, se houver necessidade, fazer mudanças nos mesmos.

- Encerrar os *releases*

Um *release* se encerra ao final da última iteração prevista para ele. Ao final do *release*, a equipe coloca o sistema em produção para que todos os usuários passem a ter acesso a ele. Este processo pode envolver inúmeras atividades para que o sistema entre no ambiente de produção do cliente. Dependendo da empresa cliente, isso pode ser muito rápido ou muito lento.

3.2.3 Reunião em Pé

O XP propõe que no início de cada dia seja realizada uma reunião em pé, uma reunião rápida que envolve todos os membros da equipe de desenvolvimento. Recomenda-se que esta reunião seja rápida e consuma em torno de dez minutos.

A reunião em pé serve para que todos os membros da equipe comentem rapidamente o trabalho que executaram no dia anterior. Desta forma, toda a equipe fica autorizada sobre o andamento do projeto naquele dia. A reunião é uma oportunidade que todos têm de apresentar aos demais os desafios que enfrentaram e a solução que criaram para eles. Desta forma, quando problemas semelhantes ocorrerem, todos saberão o que fazer ou quem procurar.

Esta reunião não tem apenas o objetivo de avaliar o passado. Ela também é utilizada para decidir o que será feito no dia que se iniciará. Ela serve para priorizar as atividades dos membros da equipe ao longo do dia.

3.2.4 Programação em Par

A programação em par é uma técnica onde dois programadores trabalham em um mesmo problema ao mesmo tempo e em um mesmo computador. Enquanto um programador (o condutor) assume o teclado e digita os comandos que farão parte do programa, o outro (o navegador) o acompanha fazendo um trabalho de estratégia.

Esta técnica congrega diversas técnicas em uma só. Ela é utilizada para fazer a revisão do código em par. Isto é, enquanto o condutor digita, o navegador está permanentemente revisando o código e tentando evitar que eventuais erros do condutor passem despercebidos.

Segundo Teles (2004), a programação em par praticamente elimina erros na medida em que eles são quase sempre detectados pelo parceiro que não está com a mão no teclado.

O uso da programação em par gera idéias mais eficazes e, ao mesmo tempo, mais simples. A questão da simplicidade é bastante nítida. Se um parceiro sugere algo que é muito complicado, o outro rapidamente consegue notar a complexidade, na medida em que surgirem as dúvidas. Nestes casos, o parceiro que está com dúvidas pode sugerir modificação para simplificar a solução.

A programação em par corrige os eventuais desvios de atenção quando o programador está só, através da pressão do par. Quando o programador está acompanhado de outra pessoa, ele deixa de ter um compromisso apenas consigo mesmo. O seu compromisso se expande e passa a englobar também o seu colega. Ou seja, sua responsabilidade aumenta já que passa a envolver mais alguém. O efeito disso é que diversos focos de distração são eliminados.

A condução da programação em par é uma atividade que deve ser realizada sempre por ambos os programadores que formam um par. É fundamental que eles se revezem diversas vezes ao longo de um dia de trabalho.

O revezamento não se aplica apenas a quem está conduzindo o teclado. Em um projeto trocamos de par com frequência e esta troca de par é importante para a disseminação do conhecimento.

3.2.5 Refactoring

Refactoring é uma técnica da engenharia de *software* que permite a reformulação de um código mal escrito, tornando-o mais claro de se entender e mais fácil de manter-se.

Um código mal formulado cria dificuldades sérias para quem um dia tiver a necessidade de modificá-lo ou compreendê-lo por alguma razão. Quando o código é limpo e legível, é possível alterá-lo rapidamente. Por outro lado, se estiver uma bagunça, pode ser necessário gastar várias horas, ou até dias, para entendê-lo. A pessoa que tiver de manter este código, poderá ser alguém de outra equipe ou alguém da mesma equipe. De fato, ao longo do desenvolvimento, os desenvolvedores podem se deparar várias vezes com códigos mal formulados.

O XP recomenda que os desenvolvedores não fiquem parados diante desta situação. Se um desenvolvedor tem uma dúvida em uma parte do código ele deve procurar compreendê-lo e, em seguida, reescrevê-lo de modo a ficar mais legível.

3.2.6 Desenvolvimento Guiado pelos Testes

Desenvolver software é uma atividade que envolve manipular uma grande quantidade de elementos abstratos. As interações entre estes elementos podem gerar uma infinidade de resultados, alguns dos quais indesejáveis e imprevisíveis.

Quando os testes fazem parte do desenvolvimento, os programadores recebem *feedback* mais rápido sobre aquilo que estão implementando. Desta forma, aprendem mais, resolvem os defeitos de forma mais rápida e têm mais tempo para desenvolver funcionalidades e gerar valor para o cliente.

O XP trabalha com testes de unidade e testes de aceitação. O teste de unidade é aquele realizado sobre as classes do sistema para verificar se os resultados gerados são corretos. Estes testes são escritos pelos desenvolvedores enquanto codificam o sistema. Já os testes de aceitação são efetuados sobre as funcionalidades, ou histórias do sistema. Este teste procura simular a interação do usuário com o sistema e verificar se o seu comportamento está correto.

Uma iteração envolve sempre o desenvolvimento de um conjunto de histórias. Mas, além de implementá-las, a equipe também deve se preocupar com os

testes de aceitação. Eles devem ser sempre escritos antes que a equipe comece a implementar as estórias.

3.2.7 Código Coletivo

É comum encontrarmos “ilhas de conhecimento” nos projetos de software. Uma “ilha de conhecimento” pode ser entendida como sendo uma pessoa que possui um domínio sobre uma área do projeto. Somente esta pessoa conhece com detalhes aquela parte ou aquele código.

Diversos problemas podem surgir com o advento das “ilhas de conhecimento”. A pessoa que atua como uma “ilha de conhecimento” pode se transformar em um gargalo dentro do projeto caso a parte do código que está sob sua responsabilidade precise ser modificada inúmeras vezes. Neste caso, apenas uma pessoa é capaz de mexer no código.

Outro problema diz respeito à qualidade e legibilidade do código. Como apenas uma pessoa é responsável pelo código, não é possível saber se ele está legível ou não. Às vezes, ele pode estar absolutamente claro para aquela pessoa que lida com ele diariamente, mas totalmente incompreensível para outros membros da equipe.

No XP não existe uma pessoa responsável por um subconjunto do código. Cada desenvolvedor tem acesso a todas as partes do código e total liberdade para alterar o que necessitar, ou seja, o código é coletivo.

O código sendo coletivo, cada parte dele pode eventualmente sofrer a visita de diferentes desenvolvedores ao longo de sua vida. A consequência direta é que o código estará sempre sendo revisitado. Como diversas pessoas tentarão compreendê-lo, sempre que alguma coisa não estiver clara, ela será substituída, ou será feito um refactoring.

3.2.8 Código Padronizado

O XP sugere padrões de codificação para que a equipe de desenvolvimento possa se comunicar de forma clara através do código fonte.

Se analisarmos o código de diferentes programadores, provavelmente notaremos que existem diferentes estilos. Por exemplo, um programador prefere

colocar “{” no final da declaração de um método, enquanto outro prefere colocar na linha posterior, segue:

Programador 1 – posiciona { no final da declaração

```
Public int soma (int a, int b) {  
    Return a + b;  
}
```

Programador 2 – posiciona { na linha posterior

```
Public int subtrai (int a, int b)  
{  
    Return a - b;  
}
```

Essas diferenças de estilo costumam estar presentes entre os programadores de qualquer equipe de desenvolvimento. Tais diferenças dificultam a compreensão do código por todos os membros da equipe. Isso pode ser evitado se todos adotarem um mesmo estilo de codificação.

O XP recomenda que a equipe adote um padrão no início do projeto. Ela pode construir este padrão em conjunto ou pode adotar um que já exista e esteja documentado.

3.2.9 Releases Curtos

No desenvolvimento de software tradicional, o projeto é dividido em fases de modo que comece na análise, passe para o design, em seguida para a implementação e os testes e, finalmente, a implantação. Quando o sistema é implementado o cliente começa a obter os benefícios de sua utilização em seu negócio. Ou seja, no desenvolvimento tradicional, o valor gerado pelo software só começa a ser recebido pelo cliente quando o sistema está finalizado.

O XP sugere que a equipe de desenvolvimento implemente as funcionalidades mais importantes o quanto antes. Ou seja, o cliente deve selecionar o conjunto de funcionalidades que poderá gerar um retorno mais alto e mais rápido. Desta forma, os primeiros releases deverão fornecer o máximo de valor, de modo que o cliente consiga obter o máximo de retorno para o seu negócio.

Releases curtos permitem que os usuários do sistema tenham acesso a novas funcionalidades mais rapidamente e com maior frequência. Isso tem um impacto positivo no aprendizado da equipe de desenvolvimento e do próprio cliente.

3.3 CARACTERÍSTICAS DA EQUIPE

O XP define alguns papéis que devem existir em uma equipe a utilize, como:

- Gerente de Projeto

O gerente de projeto é responsável principalmente pelo relacionamento da equipe de desenvolvimento com o cliente. É a pessoa que deve reportar ao cliente o andamento da equipe, os problemas que vêm sendo encontrados e os riscos que são identificados. O gerente deve estar presente na maioria das reuniões que forem convocadas pelo cliente.

- *Coach*

O papel do *coach* é assegurar que a equipe respeite e utilize diariamente os valores e práticas do XP. É uma pessoa que possui bastante conhecimento técnico e conhece profundamente o processo de desenvolvimento. Seu papel no projeto é fundamental porque o sucesso do XP vem do fato do processo atuar fortemente sobre o comportamento das pessoas que formam a equipe.

- Analista de Teste

O analista de teste é a pessoa responsável por testar e garantir a qualidade do sistema. Ele deve ajudar o cliente a escrever os testes de aceitação e deve assegurar que o sistema esteja respeitando todos eles. Ao final de cada iteração, o analista de teste deve verificar todo o software com base nos teste de aceitação.

- Redator Técnico

O redator técnico é responsável por manter as documentações do sistema atualizadas. Fazendo isso, ele evita que os desenvolvedores se envolvam fortemente no trabalho de

documentação, o que permite que eles dediquem mais tempo ao trabalho de codificação.

- Desenvolvedor

O desenvolvedor é a pessoa que analisa, projeta e codifica o sistema. Em resumo, é a pessoa que efetivamente constrói o software. Dentro do XP, não existem divisões entre analista, projetista, programador e etc. Cada desenvolvedor exerce estes diferentes papéis em diversos momentos do projeto.

3.4 DOCUMENTAÇÃO DO PROJETO

O XP define que a documentação das funcionalidades do sistema seja feita apenas o mínimo necessário antes de chegar ao código. Ao mesmo tempo, utiliza a comunicação face-a-face como fonte de informação para a codificação. A documentação mais detalhada só ocorre efetivamente quando o código já está pronto.

Este é a grande diferença da documentação do XP em relação à documentação tradicional. No processo tradicional, documentamos bastante antes de chegar até o código e fazemos as alterações à medida que o código é construído. No XP, documentamos quando a funcionalidade já está pronta. Portanto, estamos menos expostos ao re-trabalho que surge quando a codificação nos força a alterarmos aquilo que havia sido documentado.

O problema de documentar após a codificação é que isso pode deixar para depois um trabalho bastante grande. Portanto, existe o risco da documentação ser deixada de lado face às pressões do projeto. Para evitar este problema é importante que a documentação do código de uma iteração seja feita dentro da iteração. Ou seja, ao final da iteração, deve-se documentar todas as funcionalidades para a iteração. Este trabalho não pode ser deixado para o final do projeto, nem para a iteração seguinte.

O XP não define um conjunto de documentos necessários para o projeto. Cada projeto tem uma característica específica e pode necessitar de mais ou menos documentos. No entanto, o XP sugere alguns documentos que podem ser interessantes na maioria dos projetos, como: Estória, Testes de Aceitação, Testes

de Unidade, *Javadoc*, Modelo de Classes, Modelo de Dados, Modelo de Negócio, Manual do Usuário, Acompanhamento Diário, Acompanhamento do Projeto e Fotos.

Neste capítulo vimos os conceitos que envolvem o XP, sua importância para o desenvolvimento de software em projetos de pequeno e médio porte, onde a equipe de desenvolvimento é reduzida e exista a necessidade da presença do cliente como forma de aumentar o *feedback* com a equipe. Agora passaremos a mostrar os conceitos que envolvem o ICONIX.

4 ICONIX

O ICONIX é um processo de desenvolvimento de software desenvolvido pela *ICONIX Software Engineering*. Este processo é considerado puro, prático e simples, mas também poderoso e com um componente de análise e representação dos problemas sólido e eficaz.

O ICONIX é um processo não tão burocrático como o RUP, ou seja, não gera tanta documentação. E apesar de ser um processo simples como o XP, não deixa a desejar na Análise de *Design*, e se destaca com um poderoso processo de análise de software.

Este processo também faz uso da linguagem de modelagem UML e possui uma característica exclusiva chamada "Rastreabilidade dos Requisitos" (*Traceability of Requirements*), mais precisamente, ele nos permite "obrigatoriamente" através de seus mecanismos, verificar em todas as fases, se os requisitos estão sendo atendidos.

A abordagem do ICONIX é flexível e aberta, isto é, pode-se fazer uso de outros recursos da UML para complementar os recursos usados nas fases do ICONIX, caso necessário.

O ICONIX é composto por 5 fases principais:

- Modelo de Domínio
- Modelo de Caso de Uso
- Análise Robusta
- Diagrama de Seqüência
- Diagrama de Classe

Este processo está dividido em duas partes, o modelo estático e o modelo dinâmico. Como mostra a figura 6.

No modelo estático encontram-se os modelos de domínio e os diagramas de classe, neste modelo não há interação com o usuário. Enquanto que no modelo dinâmico encontram-se os modelos de casos de uso, diagramas robustos e diagramas de seqüência, neste modelo há dinamismo, ou seja, o usuário está sempre interagindo com o sistema através de ações onde o sistema apresenta alguma resposta em tempo de execução.

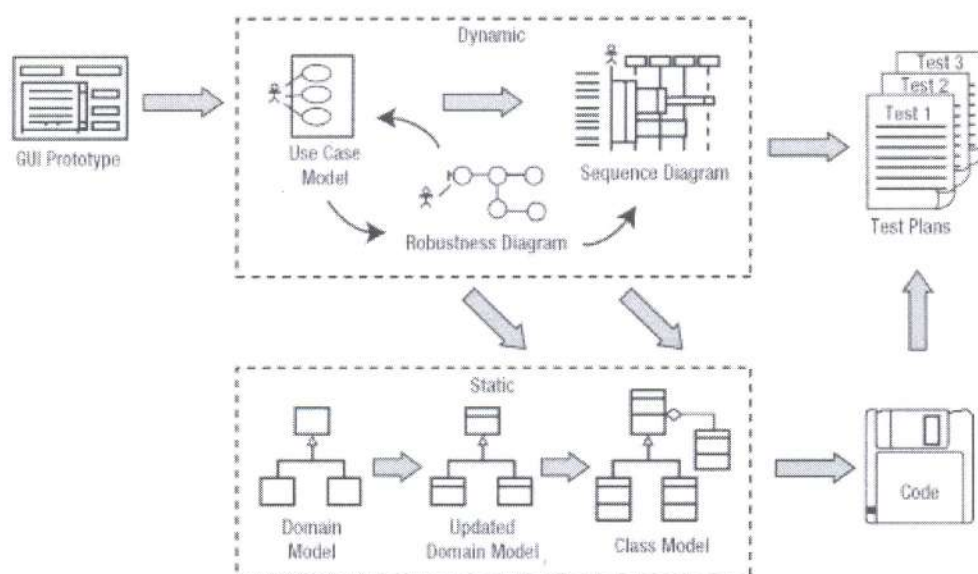


Figura 6 – Visão Geral do ICONIX
Fonte: (STEPHENS, 2007)

4.1 MODELO DE DOMÍNIO

O Modelo de Domínio é uma parte essencial do processo de ICONIX. Ele constrói uma porção estática inicial de um modelo que é essencial para dirigir a fase de design a partir dos casos de uso.

O modelo de domínio consiste em identificar os objetos de um problema do mundo real e transferi-los para um diagrama. Para realizar o modelo de domínio, é preciso tentar descobrir o maior número possível de classes existentes no problema para qual se pretende desenvolver o software. Nem todas as classes que retratam o cenário do problema serão identificadas no início do projeto, portanto outras serão encontradas, modificadas ou excluídas com o andamento do projeto.

Para se construir um modelo de domínio a primeira coisa que temos a fazer é achar as classes apropriadas que representam as reais abstrações do domínio do problema. Ao executar bem esta atividade, teremos uma base sólida para construir o sistema.

O ICONIX sugere algumas práticas que devem ser seguidas para a correta construção de um modelo de domínio:

- Destacar todos os substantivos e frases que contenham substantivos, analisar e refinar esta lista gradualmente, pois substantivos e frases de substantivo se tornarão objetos e

atributos, enquanto que verbos e frases de verbo se tornarão associações. Casos possessivos como (“seu”, “nosso” e “seus”) tendem a indicar serem substantivos, mas, devem ser atributos no lugar de objetos.

- Revisar a lista de classes em busca de ambigüidades, irrelevância, classes incorretas ou vagas e itens desnecessários, algumas vezes mesmo que sejam substantivos é possível que estejam fora do escopo do problema.
- Fazer generalização quando necessário, tomar cuidado com generalização, verificar se realmente existe uma relação de generalização e garantir que esteja de acordo com o mundo real. Verifique se for o caso de existência de agregação ou composição e aplique no modelo.
- Revisar o diagrama, com atenção em associações e generalizações, pois, este diagrama será a base para o diagrama de classe.

4.2 MODELO DE CASO DE USO

O modelo de caso de uso representa as exigências dos clientes, ele deve detalhar de forma clara e legível, todos os cenários que os usuários executarão para realizar alguma tarefa. Ou seja, deve especificar as funcionalidades ou requisitos de todo o sistema.

Este modelo é criado a partir de um protótipo de interface validado pelo cliente de acordo com os requisitos do sistema. O objetivo da interface não é codificação do sistema e pode ser feito usando qualquer ferramenta ou tecnologia mesmo que esteja longe da realidade do sistema proposto pelo cliente. No ICONIX, os sistemas são concebidos a partir da visão do usuário para com o sistema, por isso o uso de um protótipo construído “na presença do cliente” de acordo com suas necessidades. O uso de protótipos de interface possibilita ter maiores garantias que os requisitos estão sendo atendidos e uma visão maior do sistema e conseqüentemente maior controle.

Para construir um modelo de caso de uso seguiremos algumas etapas:

- Identificar e descrever cada caso de uso detalhadamente com clareza e sem ambigüidades, manter um ciclo de refinamento de texto contínuo.
- Para cada caso de uso, refinar o texto tendo certeza que orações estão claras e bem descritas, o formato básico do texto é substantivo-verbo-substantivo, e os potenciais atores e objetos de domínio são fáceis identificar.
- Atualizar o modelo de domínio sempre que novos objetos forem identificados na especificação.
- Durante a descrição dos casos de uso procurar definir claramente os cursos de ação básicos (realização da ação pelo usuário) e os cursos alternativos (algo que impeça o curso de ação básico).
- Verificar se todos os casos de uso atendem as necessidades funcionais desejadas no sistema.
- Verificar se para cada caso de uso foi descrito o curso básico de ação e o(s) curso(s) alternativo(s).

4.3 ANÁLISE ROBUSTA

A análise robusta envolve um trabalho sobre a descrição de um caso de uso, fazendo uma simulação de como seria o desenho de um software antes da implementação deste mesmo caso de uso, através do uso de objetos montaremos este diagrama, como mostrado na figura 7.

Esta fase tem como objetivo, conectar a parte de análise com a parte de projeto assegurando que a descrição dos casos de uso estão corretas, além de descobrir novos objetos através do fluxo de ação. A Análise Robusta focaliza construir um modelo analisando as narrativas de texto de caso de uso, identificando um conjunto de objetos que participarão de cada caso de uso.

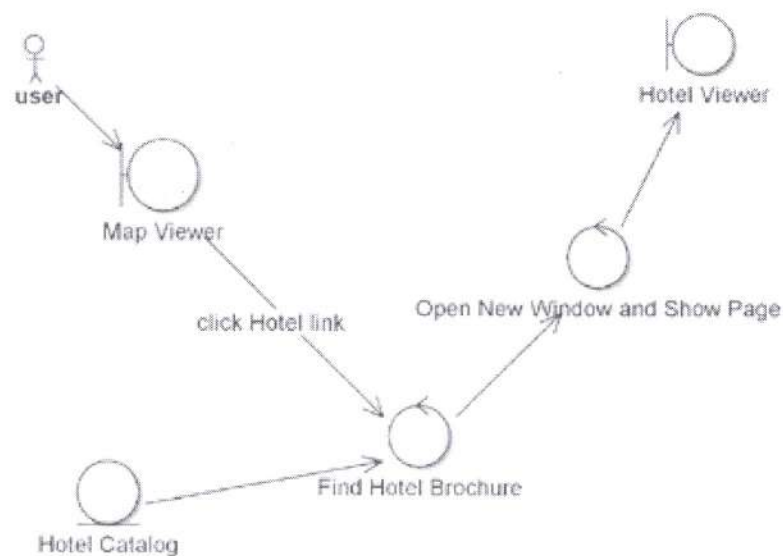


Figura 7 – Exemplo de Diagrama de Análise Robusta
Fonte: (STEPHENS, 2007)

A análise robusta é dividida nas seguintes etapas:

Etapas 1: Desenhar o diagrama para cada caso de uso

- a. Identificar os objetos que fazem parte do cenário, utilizando estereótipos UML, ver figura 8.

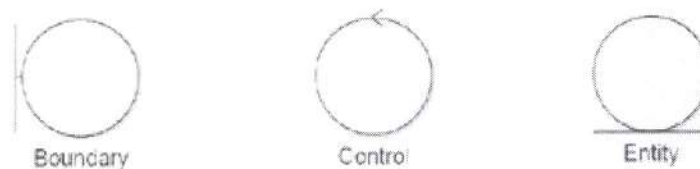


Figura 8 – Estereótipos da Análise Robusta
Fonte: (STEPHENS, 2007)

- b. Atualizar o modelo de domínio com os objetos e atributos descobertos.
- c. Atualizar a descrição do caso de uso, removendo as ambigüidades.

Etapas 2: Atualizar o diagrama de classe que reflete a complementação da fase de análise do projeto.

Objetos da Análise Robusta:

- Objetos Limite ou interface (*Boundary Objects*) – é usado pelos atores para se comunicarem com o sistema.

- Objetos Controladores (*Control Objects*) - são objetos que controlam a lógica de negócio, eles fazem a conexão entre os objetos interface e objetos entidade.
- Objetos entidade (*Entite Objects*) são responsáveis por realizar algum tipo de persistência. Geralmente eles vêm do modelo de domínio.

A análise robusta é um modelo simples, mas, é uma técnica extremamente útil para análise, é uma fase primordial e têm vários papéis dentro do ICONIX.

Para construirmos um modelo de análise robusta seguiremos algumas dicas interessantes, como:

- Caso encontremos problemas na descrição dos casos de uso, devemos corrigi-los. Ao final desta fase, teremos o modelo de caso de uso refinado. O mesmo acontecerá para o diagrama de classe;
- Ter certeza que os casos de uso foram descritos corretamente e estão de acordo com as funcionalidades do sistema;
- Se necessário, perder bastante tempo neste modelo, pois, ele servirá para construção do diagrama de seqüência. Caso este modelo não esteja de acordo com a realidade ou mal feito, não será possível prosseguir para o diagrama de seqüência;
- Durante a construção do modelo robusto, será possível encontrar objetos novos não identificados anteriormente no modelo de domínio;
- A análise robusta é a ponte entre a fase de análise e a fase de projeto, portanto, ao concluir este modelo, nós estaremos partindo para os detalhes de projeto;
- Qualquer pessoa deve ser capaz de rastrear um diagrama robusto de acordo com a descrição do texto de caso de uso;
- Refinar o modelo estático (diagrama de classe) continuamente com os novos objetos encontrados e se alguns atributos importantes foram descobertos é hora de ir povoando nossas classes mais importantes com eles;

Regras para uso dos objetos na análise Robusta:

- Atores somente podem se comunicar com objetos interface;
- Objetos interface somente podem se comunicar com atores e controladores;
- Objetos entidade somente podem se comunicar com objetos controladores;
- Objetos controladores somente podem se comunicar com objetos entidade e interface, também com outros controladores, mas não com atores;

4.4 DIAGRAMA DE SEQÜÊNCIA

Esta é a fase onde o design do projeto começa. O ICONIX utiliza o diagrama de seqüência, mas nada impede que outros diagramas sejam utilizados nesta fase.

O Diagrama de Seqüência tem como objetivo construir um modelo dinâmico entre o usuário e o sistema. Para tal, utilizaremos os objetos e suas interações identificadas na análise robusta, só que agora, temos por obrigação o detalhamento de cada fluxo de ação. A figura 9 mostra um diagrama de seqüência.

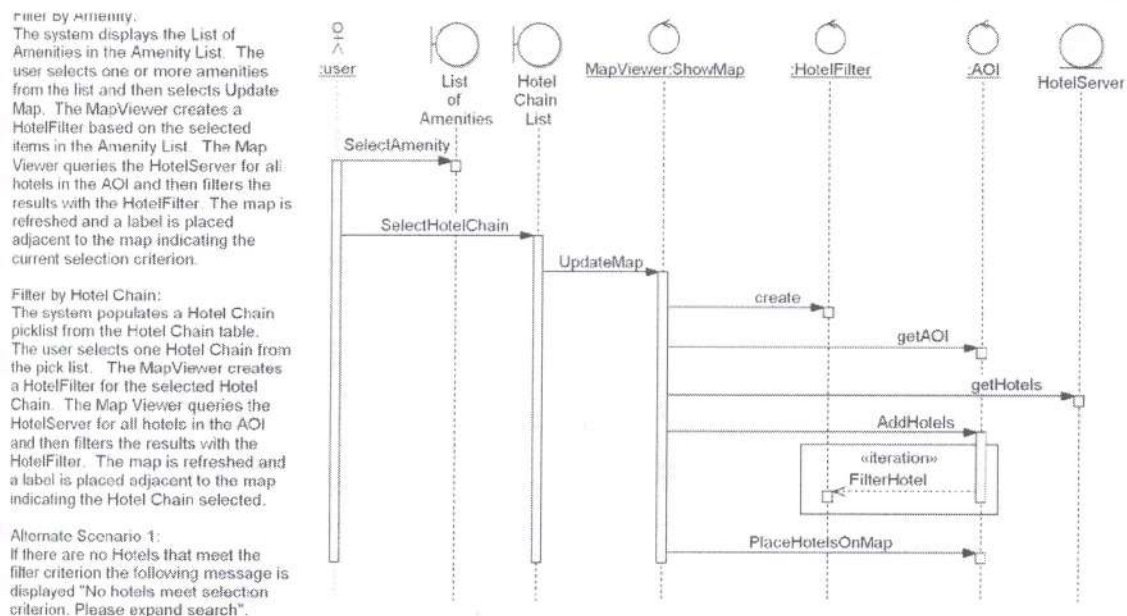


Figura 9 – Exemplo de Diagrama de Seqüência
Fonte: (STEPHENS, 2007)

Um Diagrama de seqüência é composto por quatro componentes:

- O texto que descreve os fluxos de ação para os casos de uso – o texto deve estar do lado esquerdo das linhas de ação dos objetos, é bom separar o texto com uma linha em branco entre os fluxos de ação, assim, fica fácil ver cada linha de ação e sua descrição.
- Os objetos que interagem entre si – os objetos são extraídos da análise robusta e são formados por dois componentes: o nome do objeto (opcional) e a classe a qual ele pertence, todos ficam no topo do diagrama dentro de uma caixa da qual desce uma linha pontilhada.
- Mensagens - mensagens são setas entre objetos. Uma seta de mensagem pode ir diretamente entre duas linhas pontilhadas, entre uma linha e um método de um retângulo, ou entre dois métodos de um retângulo.
- Métodos ou operações – são mostrados em cima das linhas a qual pertencem os objetos.

Para que construirmos um diagrama de seqüência, seguiremos as seguintes etapas:

- Identificar as mensagens passadas entre os objetos e os métodos associados para serem invocados.
- Desenhar o diagrama de seqüência com a descrição do caso de uso aberto no lado esquerdo do diagrama.
- Atualizar o diagrama de classe com os atributos e métodos encontrados no diagrama de seqüência.

4.5 DIAGRAMA DE CLASSE

O diagrama de classe é o modelo de domínio que foi atualizado ao longo das fases do ICONIX e representa as funcionalidades do sistema de modo estático sem a interação do usuário com o sistema. A figura 10 mostra um exemplo de um diagrama de classe.

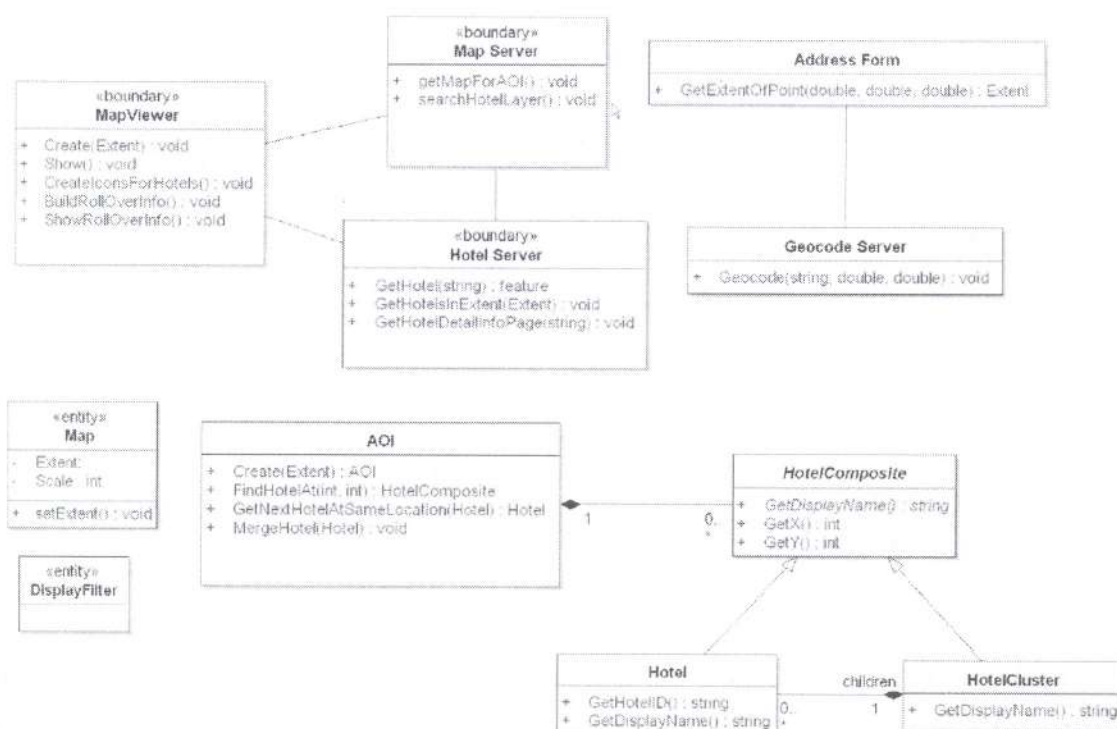


Figura 10 – Exemplo de Diagrama de Classe
Fonte: (STEPHENS, 2007)

4.6 GERANDO CÓDIGO

Esta é a fase do ICONIX onde os desenvolvedores começam a implementar as funcionalidades do sistema. Os desenvolvedores podem estar envolvidos em todas as etapas de desenho do software, o ideal é que os desenvolvedores sejam também designers.

As etapas que envolvem esta fase são: Escrever/Gerar código e Integração com os testes.

Neste capítulo apresentamos o ICONIX e suas principais fases. O ICONIX é um processo simples, porém possui fases de análise que superam o XP e que não deixam nada a desejar ao RUP. Uma vez que utilizam a notação UML para modelagem dos requisitos e processos. No próximo capítulo apresentaremos um comparativo entre os três processos de desenvolvimento de software vistos anteriormente.

5 COMPARATIVO ENTRE OS PROCESSOS

Neste capítulo faremos um comparativo entre as metodologias RUP, XP e ICONIX. Enfatizando o tempo do projeto, custo, artefatos, documentação, codificação e modelagem

Com base nos capítulos anteriores, notamos que estas metodologias funcionam com base em iterações, têm integração com o cliente e possuem papéis bem definidos.

5.1 CUSTO DOS PROCESSOS

O RUP apresenta quatro fases em seu ciclo de vida, onde em cada fase existem iterações que irão percorrer as disciplinas que dão suporte para o processo de desenvolvimento. Este processo torna-se um tanto quanto burocrático pela quantidade de atividade executadas e pelos inúmeros artefatos produzidos. A maioria dos esforços é reservado para a fase de construção onde será desenvolvido e testado a aplicação com maior intensidade. Porém, o RUP pode ser adaptado a qualquer tipo de projeto de software, desde pequenos projeto a grandes projetos. Apesar de sua flexibilização, este processo poderá tornar-se custoso para projetos de pequeno porte, e em projetos de grande porte os benefícios compensarão frente aos custos.

O XP, por sua vez, é um processo menos custoso, comparado ao RUP, no que se refere às fases de análise, design e documentação. A fase de análise e design é substituída pelas histórias, que são as descrições das funcionalidades do sistema, escritas pelos clientes, servindo como base para o desenvolvimento desta funcionalidade. Porém, o XP direciona seus esforços para a fase de codificação, onde são empregadas técnicas que viabilizam a disseminação de conhecimento do código entre os desenvolvedores através da programação em par, a reconstrução de códigos mal escritos ou de difícil entendimento através do refactoring, *feedback* diário entre os membros da equipe, onde estes expõem suas experiências do dia anterior através de reuniões em pé, bem como técnicas que definem um padrão de codificação para todos os desenvolvedores que fazem parte da equipe. Na fase inicial do processo são identificadas as histórias, onde o cliente descreve as funcionalidades do sistema. O uso destas técnicas em projetos de grande porte

pode-se tornar um gargalo na fase de desenvolvimento, podendo comprometer o custo e o prazo do projeto. Já em projetos de pequeno e médio porte, estas técnicas, podem tornar-se aliadas no que se refere à produtividade da equipe.

Já no ICONIX, grande parte do custo é gasto nas fases que antecedem a codificação do sistema, onde são produzidos os modelos de domínio, a partir dos modelos de caso de uso, permitindo a identificação dos objetos do domínio. Também são produzidos protótipos de interface, que servirão de base para o desenvolvimento das funcionalidades do sistema. Em projetos de pequeno e médio porte, o ICONIX mostra-se bastante viável, pois dá maior ênfase as fases de análise e design, dando um suporte bem modelado das funcionalidades do sistema, uma vez que utiliza a UML em seus modelos. Porém, em projetos de grande porte não se mostra uma opção muito viável, pois deixa a desejar na padronização de artefatos de documentação e atividades que envolvem os testes do sistema.

5.2 PAPÉIS

Um papel é uma definição abstrata de um conjunto de atividades executadas e dos respectivos artefatos. A definição destes papéis é de suma importância para um processo de desenvolvimento de software, uma vez que define as responsabilidades de cada indivíduo, ou grupo de indivíduos, de um projeto. O RUP é o processo que mais possui papéis, são 33 distribuídos pelas diversas fases e disciplinas do processo. Estes papéis no RUP estão divididos em grupos, segue:

- Analistas

Analista de Sistemas, Designer de Negócios, Revisor do Modelo de Negócios, Analista do Processo de Negócios, Revisor de Requisitos, Especificador de Requisitos, Analista de Teste e Designer de Interface de Usuário.

- Desenvolvedores

Designer de Cápsula, Revisor de Código, Designer de Banco de Dados, Implementador, Integrador, Arquiteto de Software, Revisor de Arquitetura, Revisor de Design, Designer e Designer de Teste.

- Testadores

Testador.

- Gerentes

Engenheiro de Processo, Gerente de Projeto, Gerente de Controle de Mudança, Gerente de Configuração, Gerente de Implantação, Revisor do Projeto e Gerente de Testes.

- Outros Papéis

Envolvidos, Todos os Papéis, Desenvolvedor do Curso, Artista Gráfico, Especialista em Ferramentas, Administrador de Sistema e Redator Técnico.

Em projetos de grande porte ou sistemas de missão crítica o RUP, mostra-se um processo mais maduro, uma vez que aborda com mais segurança todas as fases do desenvolvimento do software e especifica em detalhes as atribuições e responsabilidades de cada papel.

O XP, no entanto, apresenta apenas 5 papéis, que são: o gerente de projeto – responsável pela coordenação e acompanhamento do projeto, o *coach* – responsável por garantir que todas as fases, valores e práticas do XP estão sendo respeitadas, o analista de teste – responsável por testar e garantir a qualidade do sistema, o redator técnico – responsável por manter as documentações do sistema atualizadas, e o desenvolvedor – responsável por analisar, projetar e programar. O que torna o XP mais viável para projetos onde a equipe de desenvolvimento é reduzida. Isso não significa que teremos uma pessoa para cada papel do RUP, uma pessoa pode desempenhar diversos papéis durante o projeto, ou vários membros da equipe podem desempenhar um mesmo papel.

Enquanto que ICONIX, não existe uma definição exata dos papéis assumidos pelos membros da equipe, nas documentações encontramos referências a desenvolvedores, analistas, *testers* e designers, mas nenhuma atividade é descrita para um desses papéis. Isso pode ser um fator negativo na hora da escolha deste processo como sendo o processo de desenvolvimento escolhido. O fato de não especificar seus papéis, torna o ICONIX um processo a ser utilizado em projetos de pequeno ou médio porte, onde a equipe não seja grande, para que não ocorra a possibilidade de desencontros de atividades. Onde teremos muitas pessoas desempenhando um único papel, deixando outras atividades de serem executadas, necessitando de um maior controle por parte dos gerentes de projeto.

5.3 CODIFICAÇÃO

Todos os processos dão ênfase à fase de codificação do sistema, no RUP a fase de construção é a que demanda maior esforço por parte da equipe de desenvolvimento, nesta fase são gerados os artefatos que irão compor o executável final, como classes, bibliotecas, arquivos de configuração, bem como o lançamento de releases betas, onde são disponibilizados para a fase de transição. O RUP divide o código em subsistemas, ou seja, uma arquitetura baseada em componentes. Apesar de o RUP definir e especificar os artefatos gerados pela fase de construção, ele não define boas práticas para a codificação do sistema, nem possui técnicas que aumentam a produtividade da equipe, como as técnicas encontradas no XP.

O XP dá maior ênfase à fase de codificação, no sentido de permitir maior qualidade, clareza e padronização do código. Além do *feedback* contínuo com o cliente, este processo define várias premissas e práticas que envolvem o desenvolvimento do sistema, como tratar o código de forma coletiva, permitindo que o código seja de conhecimento de todos, uma vez que todos os desenvolvedores terão acesso e liberdade para alterar qualquer código, a programação em par, que a médio e longo prazo pode aumentar a produtividade da equipe, uma vez que quando duas pessoas estão resolvendo um determinado problema é natural que este seja resolvido de maneira mais eficiente e mais rápida do que se fosse resolvido por uma única pessoa, além de evitar as famosas distrações que os desenvolvedores passam durante todo o dia. Outro ponto importante é o refactoring de código, onde os desenvolvedores refazem códigos mal escritos ou não claros. Estes pontos do XP podem ser decisivos em um projeto que busca uma maior produtividade da equipe.

Já o ICONIX não define nenhum padrão de codificação, nem boas práticas de desenvolvimento, nem os artefatos produzidos pela equipe de desenvolvimento, enquanto o código-fonte é a parte importante do XP, o ICONIX presa mais pelas fases que antecedem a codificação do sistema, como a modelagem do domínio, a modelagem dos casos de uso, a análise robusta, o diagrama de seqüência e o diagrama de classe. Em projetos que de médio e grande porte, onde a equipe é relativamente grande, faz-se necessário que sejam definidos padrões de codificação, documentação do código-fonte e outras práticas com o intuito de prover qualidade ao desenvolvimento das funcionalidades. Apesar de o ICONIX ser um

processo guiado pelos casos de uso, diagramas de seqüência e diagramas de classe, isso por si só não viabiliza a padronização de código.

5.4 DOCUMENTAÇÃO

Em relação à documentação do projeto o RUP pode ser considerado o processo mais burocrático neste quesito, uma vez que em cada iteração são gerados/atualizados vários artefatos e documentos, a vantagem desta abordagem é que temos um projeto muito bem documentado, porém faz-se necessário um acompanhamento forte de toda documentação, uma vez que as alterações nos requisitos levam a alterações nos artefatos e/ou documentos relacionados. Este processo é mais bem adequado a projetos de grande porte, onde existe a necessidade de um nível alto de documentação de todas as fases e artefatos do sistema, bem como a existência de recursos suficientes para manter toda a documentação do projeto atualizada.

O XP por sua vez não especifica nenhum tipo de documentação, apenas define que as histórias sejam bem descritas, uma vez que servirão como base para o restante do projeto. Porém, o XP sugere alguns modelos de documentos que podem ser necessários em alguns projetos, como Histórias, Testes de Aceitação, Testes de Unidade, Javadoc, Modelo de Classes, Modelo de Dados, Modelo de Negócio, Manual do Usuário, Acompanhamento Diário, Acompanhamento do Projeto e Fotos. Sendo que para cada projeto verificaremos os documentos necessários a cada um. O XP, no quesito documentação, é mais adequado a projetos de pequeno porte onde não há a necessidade de tanta documentação, sendo descartado para projetos de grande porte que necessitam de documentação.

O ICONIX é um meio termo entre o RUP e o XP, no quesito documentação, uma vez que o RUP especifica muitos artefatos de documentação, além dos modelos UML, e o XP deixa a desejar na documentação, utilizando-se apenas das histórias para documentação das funcionalidades. No ICONIX são utilizados diagramas UML para documentação dos requisitos do sistema, estes diagramas servirão como base para todo o desenvolvimento do sistema. Para projetos de pequeno e médio o ICONIX leva vantagem em relação ao XP, uma vez que cobre a deficiência gerada por este, porém em projetos de grande porte leva desvantagem,

uma vez que não define artefatos para documentação de fonte, manual do usuário, material de suporte, dentre outros.

5.5 FERRAMENTAS

Para ser efetivo e produtivo, um processo de desenvolvimento de software deve ser suportado através de ferramentas que viabilizem o correto andamento do projeto. O RUP utiliza softwares, como o *Rational Rose*, que devem ser adquiridos da própria IBM, juntamente com a documentação do produto.

O XP não especifica nenhuma ferramenta em específico para o processo, embora haja ferramentas livres como o *XPlanner* e o *Junit*.

No ICONIX também não é especificado nenhuma ferramenta, porém pode-se fazer uso de ferramentas como *System Architect*, *Rational Rose* e o *Enterprise Architect*, uma vez que o ICONIX utiliza-se de diagramas da UML.

O custo das ferramentas pode ser um fator decisivo no momento da escolha de um processo de desenvolvimento de software. Outros fatores devem ser vistos, como usabilidade, portabilidade e documentação.

5.6 DESENVOLVIMENTO ORIENTADO POR CASO DE USO

Casos de uso fornecem um caminho para o melhor entendimento dos requisitos do sistema, através de uma perspectiva gráfica, definindo o comportamento executado por um sistema ou funcionalidade do sistema. Os casos de uso fornecem ligações importantes entre requisitos de sistema e outros artefatos de desenvolvimento, como projeto e testes. Outros processos de desenvolvimento fornecem caso de uso como representação, mas usam nomes diferentes para isto, como cenários ou caminhos.

O RUP e o ICONIX são processos orientados por casos de uso, isso significa que os casos de uso definidos para o sistema servirão como base para o resto do processo de desenvolvimento. Casos de uso exercem um papel principal em vários fluxos de processo, especialmente requisitos, projeto, teste e gerenciamento. Casos de uso também são fundamentais para a modelagem do domínio do negócio.

Já o XP não é um processo orientado por casos de uso, ele utiliza Estórias para descrição dos requisitos do sistema, que são escritas pelo próprio cliente em cartões. O XP não utiliza nenhum dos diagramas UML, o que pode ser um ponto negativo no momento da escolha deste processo, já que diagramas UML facilitam o entendimento do problema e fornecem uma ferramenta excelente para modelagem dos negócios e requisitos do sistema.

5.7 PROCESSO MAIS ADEQUADO PARA UMA PEQUENA EMPRESA

O processo de desenvolvimento de software é muito dinâmico, portanto é difícil encontrarmos uma metodologia ideal, ou que se encaixe perfeitamente para um determinado projeto de software. Ao escolhermos um processo de desenvolvimento para um determinado projeto, levaremos em consideração o tamanho da equipe, a quantidade de solicitações, as prioridades, o prazo a ser entregue, os recursos disponíveis, ou seja, diversos fatores devem ser avaliados.

Para a realidade de uma pequena empresa, onde os recursos são escassos, a equipe reduzida, muitas solicitações sugerimos o uso do ICONIX com algumas técnicas do XP. O ICONIX, pois ele utiliza alguns modelos UML para modelagem do domínio, modelagem das classes e modelagem dos requisitos (casos de uso). As técnicas de refactoring, programação em par e padronização da codificação, podem ser de grande valia para a produtividade desta equipe.

5.8 PROCESSO MAIS ADEQUADO PARA O SETOR PÚBLICO

Projetos deste setor costumam utilizar uma equipe de desenvolvimento com muitos membros, conseqüentemente demandam mais custos para o projeto, pois geralmente são sistemas de grande porte. E em sistemas de grande porte, onde são desenvolvidos inúmeros casos de uso, por diversas pessoas diferentes, já que são sistemas que duram décadas, faz-se necessário um processo de desenvolvimento robusto, que especifique de forma clara todas as fases do desenvolvimento, defina os responsáveis por cada atividade, tenha processos de teste e qualidade bem definidos e a documentação seja um quesito prioritário. Com todos esses pontos não nos resta dúvidas que o RUP é a melhor opção para o setor público.

6 CONCLUSÃO

Em uma análise generalista, identificamos que os processos de desenvolvimento estudados ao longo deste trabalho, são semelhantes do quesito da dinâmica de desenvolvimento de software, ou seja, procuram uma abordagem intermediária entre o modelo em cascata e o modelo em espiral, que interaja entre as diversas fases do projeto durante várias vezes ao longo do tempo.

Estes processos têm objetivos semelhantes, como a interação com o cliente, desenvolvimento iterativo, modelagem dos requisitos do sistema e integração com testes, buscando também a redução e o gerenciamento dos riscos dos projetos bem como um produto que atenda as necessidades dos clientes. Porém utilizam estratégias diferentes para alcançar tais objetivos.

O RUP organiza o projeto fazendo uma divisão das atividades a serem realizadas (pelos papéis responsáveis) e padroniza os artefatos que são usados como entrada e saída para as atividades. Toda essa organização viabiliza o RUP para projetos de grande porte, onde a equipe de desenvolvedores é grande e o investimento do projeto seja alto.

No XP, apesar de existirem algumas divisões de responsabilidades, elas são mais reduzidas que o RUP, uma vez que o XP não dá ênfase a documentação do projeto e sim a comunicação face-a-face com o cliente, e enfatiza a codificação do projeto. Sendo assim, esta metodologia mostra-se viável para projetos de pequeno e médio porte, onde a equipe de desenvolvimento é reduzida e os requisitos mudam com frequência.

Enquanto que o ICONIX apresenta-se como uma metodologia que enfatiza as fases que antecedem o desenvolvimento do sistema, tratando a documentação do projeto através dos modelos de negócio, modelos de casos de uso, diagramas de seqüência, diagramas de classe e diagramas robustos. Não se atendo a documentos como manual do usuário, dentre outros. Mostra-se uma metodologia viável para projetos de pequeno e médio porte, onde a equipe é relativamente pequena, onde exista necessidade de documentações dos requisitos de forma gráfica, utilizando diagramas UML.

REFERÊNCIAS

BECK, Kent. **Programação extrema explicada**: acolha as mudanças. Porto Alegre: Bookman, 2004.

DON, Wells. **Extreme Programming. Extreme Programming**: A gentle introduction. Disponível em: <<http://www.extremeprogramming.org/>>. Acesso em: 05 dez. 2007.

ICONIX. Disponível em: <<http://pt.wikipedia.org/wiki/Iconix>>. Acesso em: 21 out. 2007.

ICONIX PROCESS. Disponível em: <<http://iconixprocess.com>>. Acesso em: 23 nov. 2007.

ICONIX SOFTWARE ENGINEERING. Disponível em: <<http://www.iconixsw.com/>>. Acesso em: 24 nov. 2007.

KRUCHTEN, Philippe. **Introdução ao RUP – Rational Unified Process**. Rio de Janeiro: Ciência Moderna, 2003.

LUIZ, Ronaldo Rezende Vilela. **Obtendo Qualidade de Software com o RUP**. Disponível em: <<http://www.javafree.org/content/view.jf?idContent=7>>. Acesso em: 21 out. 2007.

MAIA, José Anízio. **Construindo Softwares com Qualidade e Rapidez Usando ICONIX**. Disponível em: <www.guj.com.br/content/articles/patterns/iconix_guj.pdf>. Acesso em: 24 nov. 2007.

MODELOS Ciclo de Vida. Disponível em: <http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida#Modelos_em_Engenharia_de_Software_e_Requisitos>. Acesso em: 22 out. 2007.

OAT SOLUTIONS. **Comparativo entre Ferramentas Case**. Disponível em: <http://oatsolutions.com.br/artigos/Comparativo_CASES_v3.pdf>. Acesso em: 30 nov. 2007.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software**: Fundamentos, Métodos e Padrões. Rio de Janeiro: Livros Técnicos e Científicos, 2001.

RATIONAL SOFTWARE CORPORATION. Disponível em:
<<http://www.wthreex.com/rup/>>. Acesso em: 18 out. 2007.

ROSENBERG, Doug; STEPHENS, Matt; COLLINS-COPE, Mark. **Agile Development with ICONIX Process: People, Process, and Pragmatism**. Apress, 2005.

STEPHENS, Matt. **Agile Development and ICONIX**. Disponível em:
<<http://www.simple-talk.com/opinion/opinion-pieces/agile-development-and-iconix/>>.
Acesso em: 27 nov. 2007.

TELES, Vinícius Manhães. **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. São Paulo: Novatec, 2004.