UNIPAC
UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
DE BARBACENA

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Eduardo Luiz Almeida Damasceno

UMA ABORDAGEM DA APLICAÇÃO DA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE "EXTREME PROGRAMMING" NAS EMPRESAS

BARBACENA
NOVEMBRO DE 2004

EDUARDO LUIZ ALMEIDA DAMASCENO

UMA ABORDAGEM DA APLICAÇÃO DA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE "EXTREME PROGRAMMING" NAS EMPRESAS

Dissertação apresentada à Universidade Presidente Antônio Carlos, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Elio Lovisi Filho

BARBACENA
NOVEMBRO DE 2004

EDUARDO LUIZ ALMEIDA DAMASCENO

UMA ABORDAGEM DA APLICAÇÃO DA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE "EXTREME PROGRAMMING" NAS EMPRESAS

Dissertação apresentada à Universidade Presidente Antônio Carlos, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em _____/_____/_____

BANCA EXAMINADORA

_____
Prof. Elio Lovisi Filho (Orientador)
Universidade Presidente Antônio Carlos

_____
Prof. Lorena Campos
Universidade Presidente Antônio Carlos

_____
Prof. Reinaldo Fortes
Universidade Presidente Antônio Carlos

Dedico este trabalho aos meus pais,
meu irmão e principalmente a
todos os meus amigos

Agradeço a Deus, aos meus professores, em especial ao
meu orientador e a todos que direta ou indiretamente
colaboraram para a realização deste trabalho

LISTA DE ILUSTRAÇÕES

1. O ciclo de vida clássico....13
2. Ilustração de Classes e Objetos....15
3. Ilustração de Atributos e Valores....15
4. Classes segundo a UML....16
5. XP comparado a desenvolvimento em cascata e a desenvolvimento interativo....18
6. Custo de uma alteração no projeto de acordo com suas fases (visão tradicional)....22
7. Custo de uma alteração não cresce muito ao longo do tempo....22
8. O relacionamento entre as técnicas da XP....25
9. Práticas da XP....27
10. O ciclo de vida do feedback....34
11. Ciclo de atividades de um desenvolvedor em XP....41
12. Ambiente de trabalho do XP....42
13. Quadros e murais em um ambiente de trabalho utilizando XP....43
14. Espaço para lanche em ambiente de trabalho utilizando XP....43
15. Cartão de estória do usuário....47
16. O jogo do Planejamento....49
17. Ciclos de desenvolvimento em XP....51

SUMÁRIO

1. INTRODUÇÃO....8
1.1 Contextualização....8
1.2 Proposta....9
1.3 Objetivos....9
1.4 Estrutura....10

2. XP: UMA VISÃO GERAL....11
2.1 Engenharia de Software....11
2.1.2 Analise estruturada....12
2.1.2 Analise orientada a objeto....14
2.2 eXtreme Programming como metodologia ágil....17
2.3 Surgimento, conceito e promessas....20
2.4 Valores e práticas....23
2.4.1 Comunicação....23
2.4.2 Simplicidade....24
2.4.3 Feedback....24
2.4.4 Coragem....24
2.5 Episodio de Desenvolvimento....27

3. XP NA PRÁTICA....30
3.1 Valores da XP....30
3.1.1 Comunicação....30
3.1.2 Simplicidade....31
3.1.3 Feedback....32
3.1.4 Coragem....34
3.2 Formação da equipe....35
3.2.1 Gerente de Projeto....36
3.2.2 Treinador....37
3.2.3 Analista de Testes....38
3.2.4 Redator Técnico....39
3.2.5 Rastreador....39
3.2.6 Desenvolvedor....40
3.3 A organização do ambiente de trabalho....42
3.4 Práticas da XP....44
3.4.1 Cliente Presente (On-site Customer)....44
3.4.2 Jogo do planejamento (The Planning Game)....45
3.4.3 Pequenos Lançamentos (Small Releases)....50
3.4.4 Desenvolvimento guiado pelos testes (Test First Design)....53
3.4.4.1 Testes de unidade (Unit Test)....54
3.4.4.2 Testes de Aceitação (Acceptance Tests)....54
3.4.5 Integração Continua (Continuous Integration)....56
3.4.6 Projeto Simples (Simple Design)....59
3.4.7 Refatoração (Refactoring)....60
3.4.8 Programação em pares (Pair Programming)....62
3.4.9 Propriedade Coletiva (Collective Ownership)....64
3.4.10 Padrões de Codificação (Coding Standards)....65
3.4.11 Ritmo Sustentável (40 Hour Week)....66
3.4.12 Metáfora....67

3.5 Métricas e Gerenciamento.....68

4. SITUAÇÃO ATUAL DA UTILIZAÇÃO DA XP NAS EMPRESAS.....71
4.1 APOENA Software Livre.....71
4.2 Improve It.....73
4.3 Naphta Informática.....74
4.4 Estudos realizados através de pesquisa.....75
4.4.1 Secretaria da Fazenda do Estado de São Paulo.....76
4.4.2 TIM.....77
4.5 Resultados.....78

5. CONCLUSÃO.....79
5.1 Revisão.....79
5.2 Conclusões.....80
5.3 Propostas.....84

6. BIBLIOGRAFIA.....85

1. INTRODUÇÃO

"A engenharia de software é uma disciplina que integra métodos, ferramentas e procedimentos para o desenvolvimento de software de computadores." (PRESSMAN, 1995). Todo projeto que envolva o desenvolvimento de um software deve ser feito utilizando-se alguma técnica de engenharia de software para que tenha uma melhor qualidade no produto final.

1.1 – Contextualização

Hoje em dia o início do desenvolvimento de um software é muito conturbado, pois passa necessariamente por uma fase de estimativas: de custo, tempo de desenvolvimento, recursos alocados, etc. Quanto mais cedo tivermos estas estimativas e quanto mais precisas elas forem, mais chances de o projeto ser bem sucedido.

Porém, são muitas as alternativas oferecidas atualmente – RUP ou XP, Java ou .Net, sistemas abertos ou proprietários, etc.– e muitas também são as novas propostas. Cabe a cada equipe conhecer suas potencialidades e necessidades, e encontrar dentre o que é fornecido pela Engenharia de Software o que poderá melhorar sua produtividade.

Atualmente RUP e XP são as duas principais metodologias de desenvolvimento de sistemas sendo adotadas ou cogitadas por equipes de desenvolvimento de software. Ambos têm muitas características comuns, como requerer seqüências de atividades ou fases onde pessoas desempenham papéis bem definidos para gerar artefatos que são encaminhados ao cliente e/ou realimentam o ciclo (SMITH, 2001). Entretanto, distinguem-se bastante em uma análise mais detalhada, o que torna cada uma mais indicada para um determinado perfil de projeto.

A eXtreme Programming (XP) vem despertando a atenção de vários autores consagrados da área de engenharia de software. Prova disto, é que referências a XP podem ser encontradas nas home pages de Edward Yourdon, Peter Coad e Roger Pressman. Além disto,

publicações especializadas importantes, tais como as da IEEE Society, tratam do assunto. A edição completa da revista IEEE Computer de Junho de 2003 (COMPUTER, 2003) foi dedicada às metodologias ágeis de engenharia de software, nas quais a XP pode ser enquadrada.

A eXtreme Programming surgiu recentemente como uma alternativa para equipes de desenvolvimento menores que precisam de agilidade para lidar com constantes mudanças em requisitos. XP baseia-se em iterações curtas e orientadas a funcionalidades com releases freqüentes para o cliente, programação em pares, design simples e emergente do próprio processo de programação e ausência de "propriedade" individual do código por parte de cada desenvolvedor (JEFRIES, 2001). XP é mais adequado para projetos com até 10 desenvolvedores e duração máxima de 15 meses (SMITH, 2001).

Este trabalho trata de eXtreme Programming (XP), uma metodologia ágil de engenharia e desenvolvimento de sistemas, focada principalmente na implementação e na comunicação entre os membros da equipe, esta metodologia enfatiza o desenvolvimento de software com foco no resultado finalque é o software. Na XP a codificação é a principal tarefa, com menor ênfase em processos formais de desenvolvimento e maior em disciplina rigorosa.

1.2 – Proposta

O que procura-se realizar é um estudo da metodologia de desenvolvimento "eXtreme Programming" através de uma abordagem de como ela é utilizada nas empresas.

Estudaremos esta metodologia, pois é uma metodologia nova no mercado e ágil para equipes pequenas e médias, a maioria hoje em dia, desenvolvendo softwares com requisitos vagos e em constante mudança. Além de ser uma metodologia transparente, produtiva, e divertida ela prima pela satisfação do cliente e pela qualidade do software final.

1.3 – Objetivos

O objetivo principal deste trabalho é pesquisar os princípios fundamentais e as características da metodologia XP. Buscando evidenciar os pontos positivos da metodologia.

Faremos isto através de uma abordagem de como esta metodologia está sendo utilizada nas empresas atualmente.

1.4 – Estrutura

O segundo capítulo apresenta uma visão geral da XP explicitando o seu surgimento, conceitos e promessas, além de seus valores e práticas de forma geral e alguns conceitos de engenharia de software. Evidenciamos também a XP como metodologia ágil e ilustramos um episodio de desenvolvimento de software utilizando a XP como metodologia.

No terceiro capítulo é apresenta uma análise detalhada da XP na prática, mostrando todas as suas características, como a organização de uma equipe típica, a organização do ambiente de trabalho e algumas métricas e gerenciamento.

No quarto capítulo são ilustrados casos reais de empresas que utilizam XP na prática do dia a dia, e com isso vem obtendo ótimos resultados.

No quinto capítulo é apresentada uma conclusão onde se busca evidenciar maneiras melhores de se desenvolver softwares utilizando esta nova metodologia.

2. XP: UMA VISÃO GERAL

"XP é o mais importante movimento em nosso campo hoje em dia. Eu estou prevendo que ele será tão essencial para a geração atual quanto o SEI e o Capability Maturity Model (CMM) foram para a passada". (DEMARCO, no prefácio de BECK, 2000).

Segundo Beck, eXtreme Programming, ou XP, mais do que um método, é um conjunto de práticas, não necessariamente novas, mas que quando aplicadas em conjunto se complementam de tal forma a auxiliar uma pequena equipe de programadores a entregarem software de forma rápida e com qualidade em um ambiente turbulento. (BECK, 2000).

2.1 – Engenharia de Software

Em meados da década de 60 houve uma intensa crise no desenvolvimento de softwares que desencadeou uma serie de discussões a respeito da qualidade dos softwares produzidos, uma vez que eram produzidos fora de controle.

Uma vez que o hardware não era mais o fator limitante para o sucesso de sistemas baseados em computador os softwares tomaram este papel. Em uma busca desenfreada por soluções computacionais a maior parte dos softwares era desenvolvida pelo próprio usuário ou pela própria organização, o que ocasiona em um processo implícito realizado no cérebro do desenvolvedor e geralmente sem nenhuma documentação.

Com o crescimento do desenvolvimento de software, que era tido como uma arte, surgiram as "software houses" que desenvolviam softwares em larga escala para todo o mercado interdisciplinar. Elas disponibilizaram centenas ou até milhares de soluções para mainframes ou microcomputadores.

Devido a este avanço no campo de software, os custos de manutenção começaram a consumir índices muito preocupantes e os problemas associados ao software continuaram a se

intensificar. A preocupação relativa ao software e a maneira pela qual ele é desenvolvido levou a adoção de práticas de engenharia de software.

A Engenharia de Software surgiu com o intuito de identificar e analisar as causas dos problemas envolvidos com o desenvolvimento de softwares, além disso, veio propor soluções economicamente viáveis para a resolução destes problemas e organizar o conhecimento sobre técnicas disponíveis para o desenvolvimento de softwares utilizando a aplicação prática do conhecimento científico no projeto e construção de programas e da documentação requerida para desenvolver, operar e manter esses programas.

Dentre os paradigmas da engenharia de software destacamos a analise estruturada e a analise orientada a objetos.

2.1.2 – Analise estruturada

A abordagem em cascata é uma forma de organizar as fases do paradigma estruturado que "requer uma abordagem sistemática, seqüencial ao desenvolvimento do software, que se inicia no nível do sistema e avança ao longo da analise, projeto, codificação teste e manutenção." (PRESSMAN, 1995).

Este modelo, também conhecido por abordagem "top-down", foi proposto por Winston W. Royce em 1970. Até meados da década de 1980 foi o único modelo com aceitação geral. Este modelo foi derivado de modelos de atividade de engenharia com o fim de estabelecer ordem no desenvolvimento de grandes produtos de software. Comparado com outros modelos de desenvolvimento de software, este é mais rígido e menos administrativo.

O modelo cascata é um dos mais importantes modelos, e é referência para muitos outros modelos, servindo de base para muitos projetos modernos. A versão original deste modelo foi melhorada e retocada ao longo do tempo e continua sendo muito utilizado hoje em dia.

Grande parte do sucesso do modelo cascata está no fato dele ser orientado para documentação. No entanto deve salientar-se que a documentação abrange mais do que arquivo de texto, abrange representações gráficas ou mesmo simulação.

A idéia principal que o dirige é que as diferentes etapas de desenvolvimento seguem uma seqüência. O resultado da etapa anterior é usado na etapa seguinte. As atividades a executar são agrupadas em tarefas, executadas sequencialmente, de forma que uma tarefa só poderá ter início quando a anterior tiver terminado.

O modelo pressupõe que o cliente participa ativamente no projeto e que sabe muito bem o que quer.

O ciclo de vida clássico abrange as seguintes atividades:

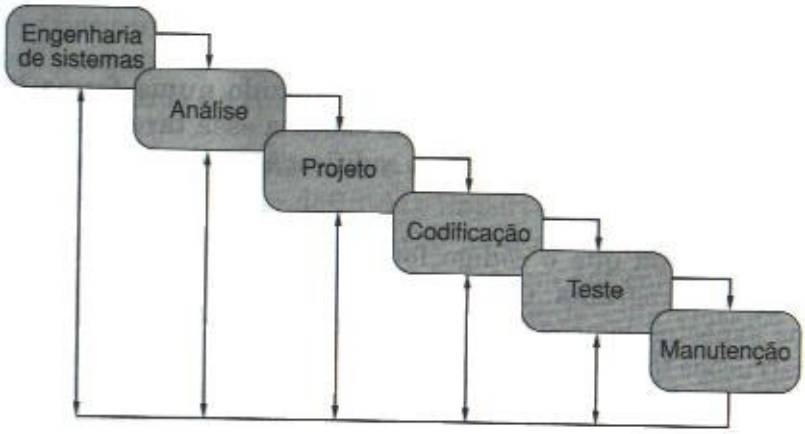

Figura 2.1 – O ciclo de vida clássico. Fonte: PRESSMAN, 1995.

Para melhor compreender as etapas deste ciclo vamos, resumidamente, explicar cada uma delas:

Engenharia de Sistemas: “A analise e engenharia de sistemas envolve a coleta dos requisitos em nível do sistema, com uma pequena quantidade de projeto e análise de alto nível.” (PRESSMAN, 1995).

Analise: Nesta etapa, são estabelecidos os requisitos do produto que se deseja desenvolver, o que consiste usualmente nos serviços que se devem fornecer, limitações e objetivos do software.

Projeto: “O projeto de software é, de fato, um processo de múltiplos passos que se concentra em quatro atributos distintos do programa: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interface.” (PRESSMAN, 1995).

Codificação: Esta é a etapa em que são criados os programas. Se o projeto possui um nível de detalhe elevado, a etapa de codificação pode implementar-se automaticamente.

Teste: O processo de teste centraliza-se em dois pontos principais: as lógicas internas do software e as funcionalidades externas. Esta fase decide se foram solucionados erros de “comportamento” do software e assegura que as entradas definidas produzam resultados reais que coincidam com os requisitos especificados.

Manutenção: Essa etapa consiste na correção de erros que não foram previamente detectados, em melhorias funcionais e de preferência e outros tipos de suporte.

“O Ciclo de Vida Clássico é o paradigma mais antigo e o mais amplamente usado na engenharia de software. Porém, no decorrer da última década, críticas ao paradigma fizeram

com que até mesmo seus antigos defensores questionassem sua aplicabilidade em todas as situações." (PRESSMAN, 1995).

"Alem disso, embora o desenvolvimento em cascata seja reconhecidamente ineficaz, ainda é o processo mais utilizado para o desenvolvimento de sistemas." (TELES, 2004).

Os projetos reais dificilmente seguem um fluxo seqüencial, este é um dos maiores problemas em relação ao modelo clássico, outro ponto a ponderar é a especificação feita pelo cliente, normalmente ele só tem o conhecimento de alguns aspectos do software que deseja, com isso muitos outros pontos só ficam claros quando ele tem a oportunidade de utilizar o sistema. "Portanto, o cliente não especifica estes detalhes no começo do projeto por uma razão muito simples: ele não os conhece." (TELES, 2004).

Com este modelo o cliente só terá acesso ao sistema em um ponto bastante tardio do cronograma, podendo correr o risco de não ter nenhuma funcionalidade a disposição caso venha a ocorrer o cancelamento do projeto.

Portanto podemos dizer que "a falácia básica do modelo em cascata é que ele assume que o projeto irá passar pelo processo apenas uma vez, que a arquitetura é excelente e fácil de utilizar, o design é correto e a implementação pode ser corrigida enquanto os testes estiverem sendo executados. Outra forma de dizer isto é que o modelo em cascata assume que todos os erros irão aparecer na implementação e, portanto, os reparos podem ser feitos sem maiores problemas restando os componentes e o sistema". (TELES, 2004).

2.1.2 – Analise orientada a objeto

O segundo paradigma a ser destacado é o orientado a objeto. Este paradigma começou a amadurecer em uma abordagem poderosa e prática no final da década de 1980 e foi desenvolvido por James Rumbaugh e colaboradores na GE (General Electric). A orientação a objetos baseia-se em notações bastante difundidas na área de desenvolvimento de software e representa uma mudança incremental às metodologias convencionais, gerando pouco impacto à migração para metodologias orientadas a objetos.

Existem muitas definições para o que se chama, em desenvolvimento de software, de Orientação a Objetos. Estudando a bibliografia da área, observa-se que cada autor apresenta a sua visão do que entende por esta abordagem. Citaremos alguns conceitos:

- A orientação a objeto pode ser vista como a abordagem de modelagem e desenvolvimento que facilita a construção de sistemas complexos a partir de componentes individuais;
- O desenvolvimento orientado a objetos é a técnica de construção de software na forma de uma coleção estruturada de implementações de tipos abstratos de dados;
- Desenvolvimento de sistemas orientado a objetos é um estilo de desenvolvimento de aplicações onde a encapsulação potencial e real de processos e dados é reconhecida num estagio inicial de desenvolvimento e num alto nível de abstração, com vistas a construir de forma econômica o que imita o mundo real mais fielmente;
- A orientação a objetos é uma forma de organizar o software como uma coleção de objetos discretos que incorporam estrutura de dados e comportamento. (POHREN, 2004).

Veremos alguns conceitos deste paradigma:

Objeto: Um objeto é algo distinguível que contém atributos (ou propriedades) e possui um comportamento. Cada objeto tem uma identidade e é distinguível de outro mesmo que seus atributos sejam idênticos. O estado de um objeto compreende o conjunto de suas propriedades, associadas as seus valores correntes. O estado de um objeto diz respeito aos seus atributos e aos valores a eles associados. Já o comportamento descreve as mudanças do estado do objeto interagindo com o seu mundo externo, através das operações realizadas pelo objeto.

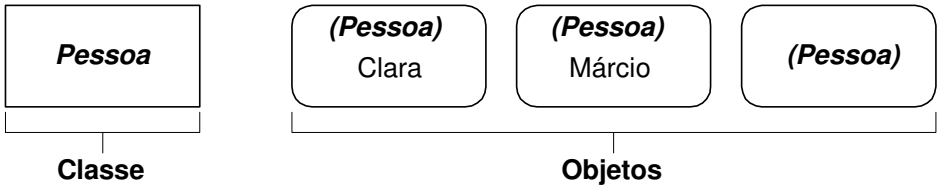

Figura 2.2 – Ilustração de Classes e Objetos.

A figura 2.2 representa classes e objetos e tem a função de ser um esquema, um padrão ou um "template" para os dados.

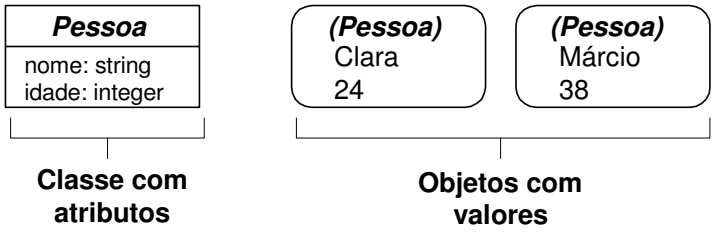

Figura 2.3 – Ilustração de Atributos e Valores.

A figura 2.3 ilustra uma classe com atributos e objetos instanciados a partir desta classe.

Classe: “Uma classe é o agrupamento de objetos com a mesma estrutura de dados (definida pelos atributos ou propriedades) e comportamento (operações).” (RUMBAUGH, 1991). Uma classe é uma abstração que descreve as propriedades importantes para uma aplicação e não leva em conta as outras. Cada classe descreve um conjunto possivelmente infinito de objetos individuais. Cada objeto é uma instância de classe. Cada instância de classe tem seu próprio valor para cada um dos atributos da classe, mas compartilha os nomes e as operações com as outras instâncias de classe.

A figura a seguir ilustra a modelagem de um projeto utilizando a orientação a objeto.

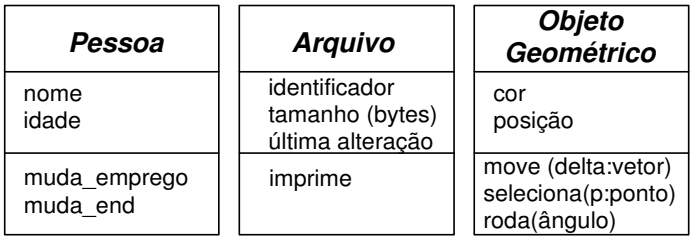

Figura 2.4 – Classes segundo a UML

A orientação a objetos se caracteriza principalmente pela abstração, encapsulamento, herança e polimorfismo.

Abstração: A abstração consiste em enfocar os aspectos mais importantes de um objeto (visão externa; o que é e o que ele faz), ignorando suas características internas (visão interna; como ele deve ser implementado).

Encapsulamento: O encapsulamento é o empacotamento de dados (atributos) e de operações sobre estes (métodos). No caso da orientação a objetos, os dados não podem ser acessados diretamente, mas através de mensagens enviadas para as operações. A implementação de um objeto pode ser mudada sem modificar a forma de acessá-lo.

Herança: A herança consiste no compartilhamento de atributos e operações entre as classes numa relação hierárquica. Este mecanismo permite a uma classe ser gerada a partir de classes já existentes; por exemplo, a classe automóvel herda da classe veículo algumas propriedades e operações. Uma classe pode ser definida de forma abrangente (como no caso do exemplo anterior) e posteriormente refinada em termos de sub-classes e assim sucessivamente. Cada subclasse herda todas as propriedades e operações da sua superclasse, (que não precisam ser repetidas) adicionando apenas as suas especificas.

Esta relação entre classes é uma das grandes vantagens de sistemas orientados a objetos por causa da redução de trabalho resultante durante o projeto e a programação destes.

Polimorfismo: O polimorfismo significa que uma mesma operação pode se comportar de forma diferente em classes diferentes. Exemplo de polimorfismo, a operação calcular o perímetro que é diferente para as instâncias de classe circulo e polígono ou a operação mover diferente para janela de computador e peça de xadrez. Uma operação que tem mais de um método que a implementa é dita polimórfica.

Este paradigma cobre todas as fases do ciclo de vida de desenvolvimento de software, desde a Análise até a Implementação. O projeto é dividido em três fases: Análise, Projeto do Sistema e Projeto de Objetos.

Coad e Yourdon refletem sobre essa questão quando escrevem:

> A OOA – analise orientada a objeto – baseia-se em conceitos que começamos a aprender no jardim de infância: objetos e atributos, classes e membros, o todo e suas partes. Por que demorou tanto tempo para que aplicássemos esses conceitos a analise e especificação de sistemas de informação é uma coisa que qualquer um pode adivinhar – talvez estivéssemos ocupados demais "seguindo o fluxo" durante o apogeu da analise estruturada para considerarmos alternativas. (COAD, 2004).

Os métodos de analise de requisitos de software orientados a objetos possibilitam que o analista modele um problema ao representar classes, objetos, atributos e operações como os componentes de modelagem primordiais. O ponto de vista orientado a objeto combina classificação de objetos, herança dos atributos e comunicação de mensagens no contexto de uma notação de modelagem.

2.2 – eXtreme Programming como metodologia ágil

A Engenharia de Software vem a anos criando técnicas de modelagem, projeto e desenvolvimento de sistemas. Dentre os desafios dos pesquisadores da área, pode-se citar a preocupação em desenvolver softwares com qualidade garantida, no prazo estabelecido e sem alocar recursos além do previsto.

Hoje em dia muitas metodologias de Engenharia de Software são usadas na prática, mas uma delas vem se destacando no mercado atual, a "eXtreme Programming (XP)".

A "eXtreme Programming" faz parte de uma serie de métodos denominados ágeis (agile), estes métodos foram inicialmente considerados leves (lightweight) para diferenciá-los dos métodos tradicionais de desenvolvimento considerados pesados (heavyweight), os quais seriam baseados na produção de uma grande quantidade de documentação e modelos para guiar a programação.

De acordo com a figura 2.5 podemos comparar a XP ao modelo tradicional de desenvolvimento de software.

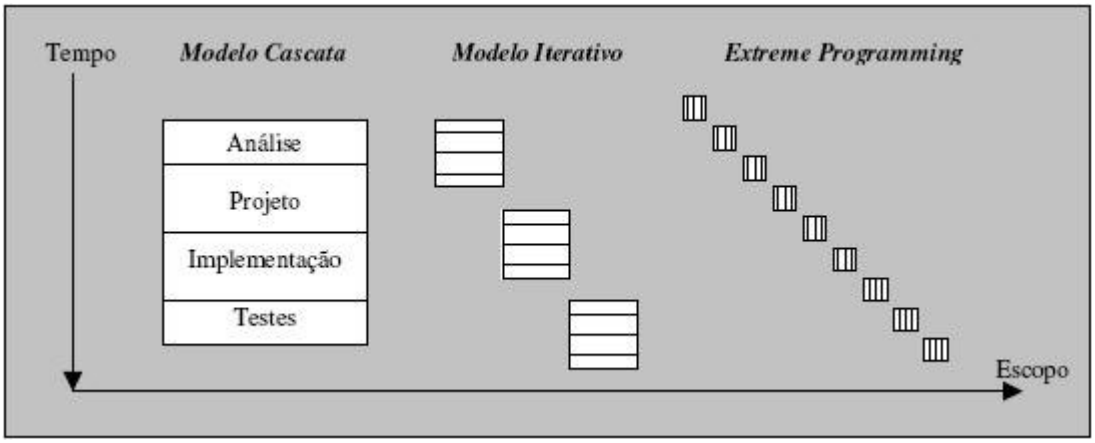

Figura 2.5 – XP comparado a desenvolvimento em cascata e a desenvolvimento interativo. Fonte: BECK, 1999.

Como se vê na figura, o modelo em cascata estabelece uma ordenação linear no que diz respeito à realização das diferentes etapas, já o modelo iterativo é um modelo de processo de desenvolvimento que busca contornar algumas das limitações existentes no modelo cascata, a idéia por trás do modelo iterativo é eliminar a política de "congelamento" dos requisitos antes do projeto do sistema ou da codificação. Já a XP trabalha com iterações de menor tamanho possível, contendo os requisitos de maior valor para o negócio, sendo assim, a equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente de modo que o cliente já possa utilizar o software no dia-a-dia e se beneficiar dele.

Há diversos relatos em que afirma-se a obtenção de melhores resultados com a utilização de métodos ágeis do que com os métodos tradicionais. No entanto, por terem estes métodos, em sua maioria, uma publicação recente, é ainda incipiente a pesquisa e a comprovação acadêmica sobre o assunto.

Segundo Laurie Williams e Alistair Cockburn (2003, p.59): “Os métodos ágeis definitivamente atingiram um nervo da comunidade de desenvolvimento de software. Algumas pessoas argumentam ferozmente a seu favor, enquanto outras argumentam contra os mesmos com a mesma energia, e outras estão trabalhando para misturar as abordagens ágeis e guiadas a planos. Um número maior ainda de pessoas ainda está tentando descobrir o que à agilidade significa".

De acordo com Scott W. Ambler, presidente da Ronin International, um modelo ágil é um modelo bom o suficiente, nada mais, o que implica que ele exibe as seguintes características:

- Ele atende seu propósito;
- Ele é inteligível;
- Ele é suficientemente preciso;
- Ele é suficientemente consistente;
- Ele é suficientemente detalhado;
- Ele provê um valor positivo;
- Ele é tão simples quanto possível. (AMBLER, 2002)

Modelagem Ágil (AM) é uma metodologia baseada na prática para modelagem efetiva de sistemas baseados em software. A metodologia AM é uma coleção de práticas, guiadas por princípios e valores que podem ser aplicados por profissionais de software no dia a dia. AM não é um processo prescritivo, ela não define procedimentos detalhados de como criar um dado tipo de modelo, ao invés ela provê conselhos de como ser efetivo como modelador. AM surge como uma arte, não como uma ciência.

AM tem três objetivos: (AMBLER, 2002).

1. Definir e mostrar como colocar em prática uma coleção de valores, princípios e práticas pertinentes à modelagem efetiva leves.
2. Explorar como aplicar técnicas de modelagem em projetos de software através de uma abordagem ágil tal como XP, DSDM ou SCRUM.
3. Explorar como melhorar a modelagem sob processos prescritivos como o Processo Unificado da Rational (RUP), ou o Enterprise Unified Process (EUP).

Ao contrario dos métodos tradicionais que são orientados ao planejamento, previsibilidade e ao controle, os métodos ágeis são orientados a construção, testes e principalmente, às pessoas. As metodologias ágeis enfatizam os aspectos humanos, as relações pessoais, uma vez que buscam valorizar o pensamento criativo dos indivíduos envolvidos no projeto, onde o conhecimento é fator importante para o sucesso do projeto.

No desenvolvimento ágil a metodologia deve produzir conhecimento não documentação. Mas isto não significa que nos ambientes ágeis não existe documentação, apenas deixa de existir a filosofia de que “tudo tem que ser documentado” e sim documentar apenas o necessário uma vez que a documentação apenas auxilia e não guia o desenvolvimento.

Desta forma podemos resumir os valores agilistas através do Manifesto Ágil, publicado em 2001 por 17 conceituados metodologistas, dentre eles podemos destacar Kent

Beck, Martin Fowler e Ron Jeffries. Esse manifesto declara que em ambientes ágeis se valoriza:

- Indivíduos e iteração **mais que** processos e ferramentas;
- Softwares em funcionamento **mais que** documentação abrangente;
- Colaboração com o cliente **mais que** negociação de contratos;
- Responder a mudanças **mais que** seguir um plano.

O manifesto reconhece que existe valor nos itens da direita, mas que os itens da esquerda são os que fazem à diferença em relação ao sucesso na entrega eficaz de valor constante ao cliente. Uma boa forma de se pensar sobre o manifesto é que ele define preferências, não alternativas.

2.3 – Surgimento, conceito e promessas

A "eXtreme Programming" foi criada por Kent Beck baseada em seus vários anos de prática em desenvolvimento de softwares orientados a objetos, juntamente com Ward Cunningham seu parceiro de programação, sendo muito usada com sucesso por grandes empresas do mercado de software.

"eXtreme Programming é uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança." Kent Beck, criador da XP.

Ela baseia-se em permanente revisão do código, testes freqüentes, participação do usuário final, refinação continua da arquitetura e planejamento, design e redesign a qualquer hora.

A XP não é novidade, ela é baseada nas conclusões atingidas após mais de uma década de pesquisas e aplicações de diversas práticas em projetos de software, na verdade XP é uma recombinação das melhores práticas de engenharia de software que comprovadamente contribuem para o sucesso dos projetos.

O nome "eXtreme Programming" vem trazendo a idéia de maximizar as práticas e técnicas de sucesso, como por exemplo:

- Se revisar o código é bom, revisaremos código o tempo inteiro (programação em par, pair programming).
- Se testar é bom, todos vão testar o tempo inteiro (testes de unidade, unit testing), até mesmo os clientes (testes funcionais, functional testing).

- Se o projeto é bom, ele fará parte do dia-a-dia de todos (refinamento, refactoring).
- Se simplicidade é bom, o sistema estará sempre na forma mais simples que suporte as funcionalidades necessárias para o momento (Design Simples, the simpliest thing that could possibly work).
- Se a arquitetura é importante, todos estarão definindo e refinando a arquitetura todo o tempo (metáfora, metaphor).
- Se testes de integração são bons, a integração será feita varias vezes por dia (integração continua, contínuos integration).
- Se iterações curtas são boas para que se possa ter um feedback rápido do cliente, elas serão planejadas para que possam ser suficientemente pequenas (pequenos lançamentos, small releases) permitindo a entrega de novas funcionalidades constantemente para o cliente (jogo do planejamento, the planing game).

A eXtreme Programming trabalha com quatro valores básicos, sendo eles Comunicação, Objetividade, Feedback e Coragem. Com a maior comunicação e contatos freqüentes com o cliente permite-se reduzir o risco de não aderência do produto à necessidade final, o foco é realizar o serviço de modo mais simples e eficaz possível, com o feedback constante do cliente os erros são mais identificados mais cedo logo mais barato e rápido é o seu conserto, mas tem que se ter bastante coragem para assumir uma postura proativa frente ao desenvolvimento do sistema, assumindo a responsabilidade pelo sucesso do projeto como um todo.

Podemos definir a XP com vários adjetivos porem ela é um processo de desenvolvimento que possibilita a criação de software de alta qualidade, de maneira ágil, econômica e flexível uma vez que prima pela satisfação do cliente e pela qualidade do software final. Sendo uma metodologia voltada para projetos cujos requisitos são vagos e mudam com freqüência, onde se utiliza orientação a objetos, com equipes pequenas, preferencialmente até 12 desenvolvedores e o desenvolvimento seja incremental (ou iterativo), isto é, o sistema começa a ser implementado logo no inicio do projeto e vai ganhando novas funcionalidades ao longo do tempo.

Em 1981, o Dr. Barry Boehm um dos profissionais mais influentes e mais frequentemente citados na indústria de software, em seu livro clássico Software Engineering Economics sugeriu que o custo de uma mudança em um projeto cresce exponencialmente “à medida que se avança nas fases de desenvolvimento. Um problema que pode custar R$ 1,00 durante a analise de requisitos pode custar R$100,00 quando o software estiver em produção, como demonstrado na figura 2.6.

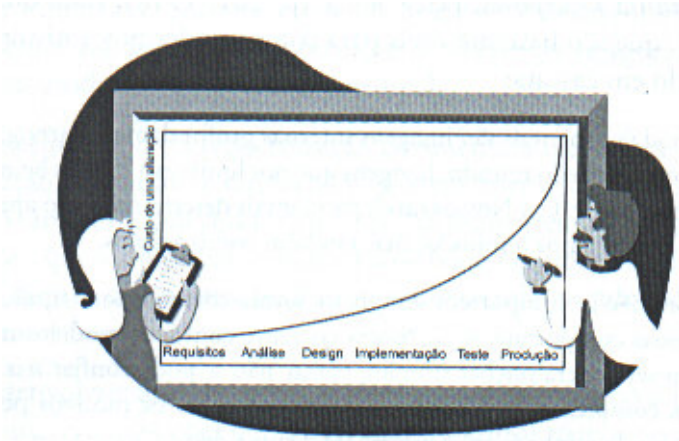

Figura 2.6 – Custo de uma alteração no projeto de acordo com suas fases (visão tradicional). Fonte: TELES, 2004.

Atualmente a comunidade de desenvolvimento de software tem feito um enorme esforço para diminuir o custo das mudanças, melhorando linguagens de programação, melhorando banco de dados, utilizando melhores praticas de programação e melhores ferramentas e ambientes.

A XP não apenas se baseia nesta curva como sugere práticas para que se necessárias às mudanças no software, estas sejam feitas de forma simples e rápida.

Trabalhar com XP indica que o custo de mudança proposto por Barry Bohem não é mais valido, ou seja, sua curva no gráfico está defasada o que era valido há 30 anos atrás hoje não é mais.

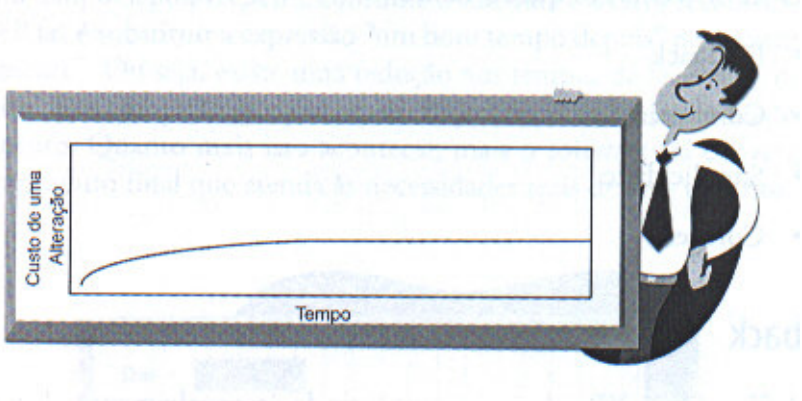

Figura 2.7 – Custo de uma alteração não cresce muito ao longo do tempo. Fonte: TELES, 2004.

Uma das premissas da XP, uma premissa técnica, é que o custo das mudanças se mantenha linear ao longo de todo o projeto, como visto na figura 2.7, independente do ponto onde o projeto esteja.

Em XP você deixará as grandes decisões para o final, desta maneira diminui a chance de tomar uma decisão errada, pois quanto mais à equipe de desenvolvimento e principalmente o cliente aprendem com o sistema, mais condições eles tem de acertar na decisão. Você só implementará o que se tem certeza que é necessário, e da forma mais simples possível.

Utilizando XP os programadores serão capazes de trabalhar com coisas que realmente importam, todos os dias e a todo momento. Eles não terão que encarar situações assustadoras. Serão capazes de fazer tudo que estiver ao seu alcance para fazer um sistema bem sucedido. Tomarão decisões que são capazes de tomar e não tomarão as decisões em que eles não são as pessoas mais qualificadas a fazer.

Para clientes e gerentes, XP promete que receberão o máximo de valor a cada semana de programação. A cada dois ou três meses eles serão capazes de ver o progresso concreto do projeto através de metas mensuráveis. Serão capazes de mudar a direção do projeto no meio de desenvolvimento sem incorrer em custos exorbitantes.

Em resumo, XP promete reduzir os riscos do projeto, melhora sensibilidade a mudanças de negócio, melhora a produtividade por todo o ciclo de vida de um sistema, e adiciona divertimento ao desenvolvimento de software em equipes - tudo ao mesmo tempo.

2.4 – Valores e práticas

Para implantar a XP é necessário que se norteie o trabalho baseado em quatro valores: comunicação, simplicidade, feedback, coragem.

2.4.1 – Comunicação

A comunicação é o principal valor da XP. Grande parte das técnicas da XP está baseada na comunicação, e se esta não for eficiente, pode causar problemas e atrasos no desenvolvimento do sistema. Uma equipe deve ter meios de se comunicar de maneira rápida e eficiente com o cliente, com o gerente do projeto e com os outros desenvolvedores envolvidos.

2.4.2 – Simplicidade

A XP está fazendo uma aposta. Ela está apostando que é melhor fazer algo simples e de desenvolvimento rápido hoje, e ter de gastar um pouco mais no futuro se for necessário remodelar um processo. A XP utiliza-se da simplicidade para implementar apenas aquilo que é suficiente para o cliente, não se procura fazer especulações sobre necessidades futuras, pois quase sempre são especulações errôneas, deixamos os problemas do futuro para o futuro.

2.4.3 – Feedback

O cliente deve receber o sistema o quanto antes, a fim de poder dar um feedback rápido, guiando assim o desenvolvimento do software. Quanto mais cedo o cliente tiver com o sistema em produção, mais rápido podem ser feitos os ajustes para que o mesmo fique de acordo com o que o cliente realmente quer. Geralmente o cliente tende a aprender mais sobre o sistema desejado à medida que este utiliza-se de protótipos para teste.

2.4.4 – Coragem

É preciso muita coragem para mudar a maneira pela qual desenvolve-se sistemas. Colocar um sistema em produção assim que ele tenha valor para o cliente, fazer apenas o que se precisa para o momento e calcar o processo de análise principalmente na comunicação não é fácil, e precisa que a equipe esteja realmente decidida a mudar o seu processo de desenvolvimento.

A XP é baseada em um conjunto de técnicas fundamentais, que podem ser aplicadas individualmente, cada uma delas gerando melhorias no processo de desenvolvimento, mas que funcionam melhor em conjunto, uma vez que uma técnica reforça o resultado da outra. São elas:

1. Cliente Presente (On-site customer);
2. Jogo do Planejamento (The Planning Game);
3. Pequenos Lançamentos (Small Releases);
4. Desenvolvimento guiado pelos testes (Test First Design);

5. Integração continua (Continuous Integration);
6. Projeto Simples (Simple Design);
7. Refatoração (Refactoring);
8. Programação em pares (Pair Programming);
9. Propriedade coletiva (Collective Ownership);
10. Padrões de codificação (Coding Standards);
11. Ritmo Sustentável (40 Hour Week);
12. Metáfora (Metaphor);

Os valores da XP assim como estas doze técnicas serão apresentados em mais detalhes no próximo capitulo.

Através da figura a seguir podemos ver o relacionamento entre as práticas do XP.

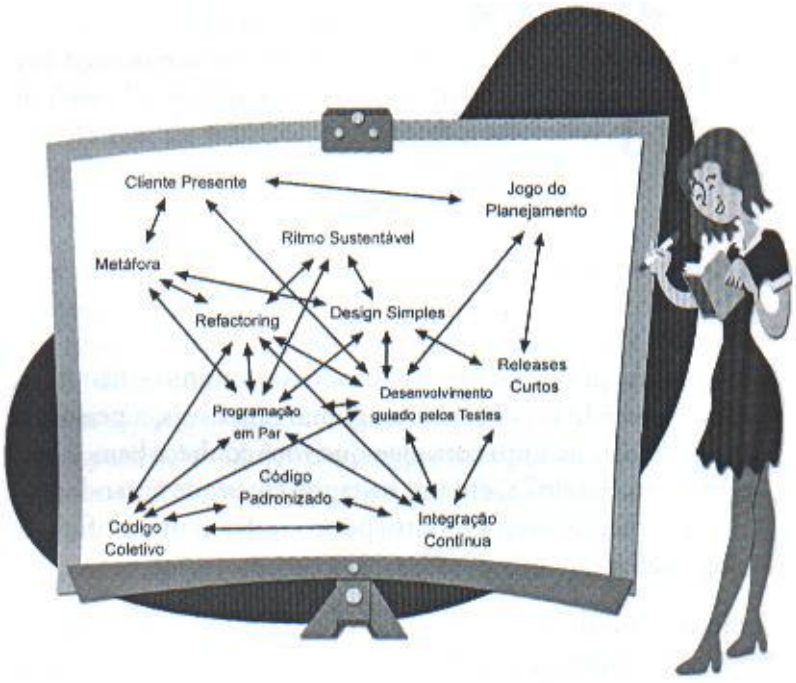

Figura 2.8 – O relacionamento entre as técnicas da XP. Fonte: TELES, 2004.

Relacionando as práticas com valores da XP podemos "casa-las" da seguinte forma:

1. Com o cliente sempre presente reforçamos a necessidade de comunicação entre as partes, ajudando assim a fazer a coisa mais simples possível e isto pode ser feito por causa do feedback constante que o cliente proporciona. Lembrando que para manter o cliente presente é necessária muita coragem uma vez que serão visualizadas pelo cliente todas as dificuldades que a equipe enfrentará.
2. A comunicação é sempre exaltada pela XP, inclusive e principalmente no jogo do planejamento onde são definidos os escopos das iterações. Nesta definição entra a

simplicidade para garantir que a equipe trabalhe sempre naquilo que ira gerar maior valor para o cliente. Com este planejamento pode-se obter um rápido feedback para o cliente, que terá suas prioridades atendidas mais rapidamente.

3. Utilizando-se de pequenos lançamentos o feedback tanto do cliente para a equipe de desenvolvimento como da equipe para o cliente é extremamente rápido o que ajuda a sempre trabalhar de forma mais simples possível.
4. Para se utilizar do desenvolvimento guiado pelos testes é necessária coragem, pois muda a cultura do programador que esta acostumado a testar somente no termino do software isso quando testa. Escrevendo os testes primeiro ajuda a se tornar o código mais claro e simples, além de passar a idéia do código mais facilmente.
5. A integração contínua estimula a simplicidade, uma vez que expõe o estado atual do desenvolvimento e oferece feedback sobre todo o sistema. Além de necessária coragem para estar sempre integrando, pois integrando sempre há o medo de entrar com bugs no sistema.
6. Antes de tudo para se refinar um código é preciso ter muita coragem, porque você estará modificando um código que já funciona, arriscando a "estragar" o software. Alterando o código ele se torna muito mais simples, passando a sua idéia mais claramente.
7. Programando em duplas a comunicação fica muito mais evidente entre os pares, cada integrante da dupla vai fornecer feedback para o outro o tempo todo, ajudando na simplicidade do código.
8. Como o código é coletivo todos irão visitar alguma parte do código algum dia o que força o desenvolvedor a escrever o código da maneira mais simples possível para que outra pessoa possa entender. Outro ponto a ponderar é que o código coletivo demanda coragem, pois o desenvolvedor terá que mexer em partes do código que não foram construídas por ele.
9. Com a utilização de padrões de codificação fica muito mais fácil a comunicação, pois todo o código escrito será simples o suficiente e entendido por todos.
10. É necessária coragem para se manter um ritmo sustentável, indicadas 40h semanais, uma vez que a maioria dos projetos de software passam por dificuldades de cumprimento de cronograma e possuem gerentes de projetos com mentalidade industrialistas expondo seus funcionários a excessivas cargas de trabalho.
11. É possível começar o desenvolvimento com apenas metáforas apenas se o feedback for concreto e rápido. O uso de metáforas ajuda a comunicação entre os

desenvolvedores e facilita a disseminação do conhecimento entre a equipe, ajudando a tornar o design simples.

As técnicas da XP são aplicadas em diferentes níveis. Há práticas organizacionais, práticas de equipe e até práticas em pares. Ilustraremos este conceito na figura a seguir.

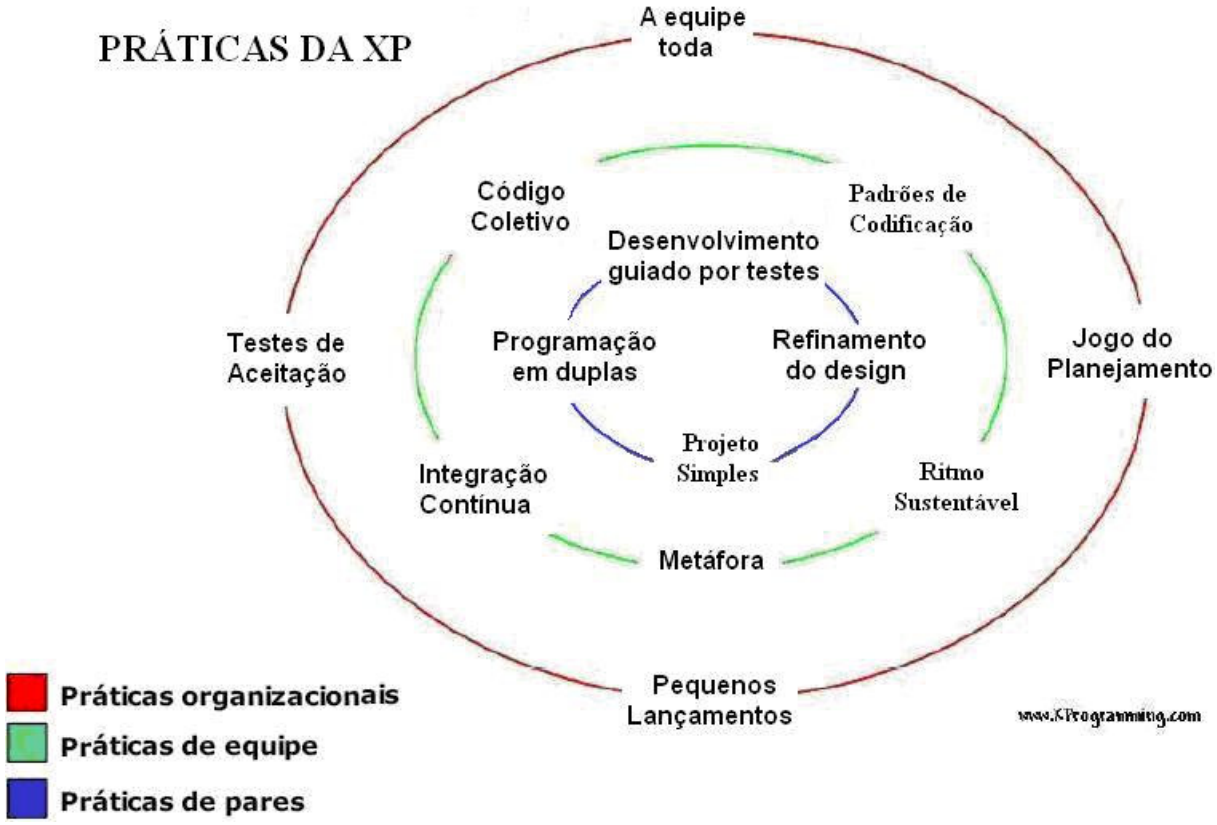

Figura 2.9 – Práticas da XP. Fonte: www.xprogramming.com.

Como vimos na figura 2.9 às práticas da XP são divididas em níveis, existem as práticas organizacionais que são o jogo do planejamento, os pequenos lançamentos, testes de aceitação e toda a equipe, existem também as práticas de equipes que correspondem aos padrões de codificação, ritmo sustentável, metáfora, integração continua e código coletivo, por ultimo temos as praticas de pares de programação que são o desenvolvimento guiado por testes, o refinamento de código, o design simples e a programação em par.

2.5 – Episódio de Desenvolvimento

Para melhor visualizar as técnicas e valores da XP veremos um exemplo de ciclo de desenvolvimento:

“Eu olho para a minha pilha de cartões de tarefas. O do topo diz “Exportar deduções do trimestre”. Na reunião (stand-up meeting) desta manhã, eu me lembro de você dizer que

terminou o calculo do trimestre atual. Eu pergunto se você (meu hipotético colega de equipe) tem tempo para me ajudar com a exportação. “Claro”, você diz. A regra é, se alguém lhe pede ajuda você tem que dizer “sim”. Nós acabamos de nos tornar parceiros de programação em dupla.

Nós gastamos alguns minutos discutindo o trabalho que você fez ontem. Você fala sobre os módulos que você adicionou, como são os testes, talvez um pouco sobre você ter notado ontem que programação em duplas funciona melhor quando você move o monitor 30cm para trás.

Você pergunta “Quais os casos de teste para essa tarefa?”.

Eu digo, “Quando rodamos a estação de exportação, os valores no registro de exportação devem ser iguais aos valores nos módulos”.

“Quais campos devem ser populados?” você pergunta.

“Eu não sei. Vamos perguntar ao Eddie”.

Interrompemos Eddie por 30 segundos. Ele explica os cinco campos que ele sabe estar relacionados com o trimestre atual.

Nós olhamos a estrutura de alguns dos casos de teste de exportação existentes. Nós achamos um que é quase o que precisamos. Abstraindo uma superclasse, nós podemos implementar nosso caso de teste facilmente. Nós fazemos o refinamento do código (refactoring). Rodamos os testes existentes. Todos eles rodam.

Notamos que vários outros casos de teste de exportação poderiam tirar vantagem da superclasse que acabamos de criar. Nós queremos ver alguns resultados na tarefa, então nós apenas escrevemos “Reaproveitar AbstractExportTest?” em nosso cartão de pendências.

Agora nós escrevemos o caso de teste. Já que nós acabamos de escrever sua superclasse, escrever o caso de teste é fácil. Nós acabamos em alguns minutos. Na metade do caminho, eu digo, “Já posso até ver como iremos implementar isso. Nós podemos...”

“Vamos acabar o caso de teste primeiro,” você interrompe. Enquanto estamos escrevendo o caso de teste, idéias de três variações vêm à mente. Você as escreve no cartão de pendências.

Nós acabamos o caso de teste e o executamos. Ele falha. Naturalmente. Nós não implementamos nada ainda. “Espere um pouco,” você diz. “Ontem Ralph e eu estávamos trabalhando em uma calculadora de manhã. Nós escrevemos cinco casos de teste que pensamos que iam falhar. Somente um não rodou de primeira”.

Nós passamos um debugger no caso de teste. Olhamos os objetos com os quais temos que calcular.

Eu escrevo o código. (Ou você o faz, quem tiver a idéia mais clara.) Enquanto estamos implementando, nós notamos mais alguns testes que devemos escrever. Nós os colocamos no cartão de pendências. O caso de teste roda.

Nós vamos para o próximo caso de teste, e para o próximo. Eu os implemento.

Você nota que o código poderia ser mais simples. Você tenta me explicar como simplifica-lo. Eu fico frustrado tentando te ouvir e implementar ao mesmo tempo, por isso eu lhe passo o teclado. Você refina (refactoring) o código. Você roda os casos de teste. Eles passam. Você implementa os outros casos de teste.

Após algum tempo, olhamos para nosso cartão de pendências e o único item nele é reestruturar os outros casos de teste. Tudo correu suavemente, então nós vamos em frente e os reestruturamos, tendo certeza que eles rodam quando terminamos.

Agora a lista de pendências está vazia. Notamos que a máquina de integração está vazia. Carregamos a última versão. Então carregamos nossas alterações. Rodamos então todos os casos de teste, os nossos novos e os testes que todas as outras pessoas já escreveram. Um falha. “Isso é estranho. Já tem quase um mês desde que eu tive uma falha durante a integração”, você diz. Sem problemas. Nós debugamos o caso de teste e corrigimos o código. Então rodamos todos de novo. Desta vez eles passam. Nós liberamos o nosso código.” (BECK, 2000).

Este é todo um ciclo de desenvolvimento em XP.

Como vimos, as técnicas são aplicadas continuamente no ciclo de desenvolvimento, através do exemplo identificamos práticas como à programação em par (Par programming), uma vez que o desenvolvimento é feito por duas pessoas em apenas um micro; a Reunião rápida (Stand Up Meeting) foi feita logo no inicio do dia para se discutir o trabalho; o desenvolvimento guiado por testes (Tests First Design); a integração continua (Continuous Integration) e o refinamento de código (Refactoring); notamos também algumas práticas comuns como os cartões de pendências e os casos de testes.

3. XP NA PRÁTICA

Neste capítulo discutiremos como colocar em prática as estratégias da XP. Ao escolher um conjunto de estratégias radicalmente simplificado, você repentinamente terá muito mais flexibilidade para trabalhar. Você pode usar essa flexibilidade para muitos propósitos, mas precisa estar ciente de sua existência e das possibilidades que ela oferece.

3.1 - Valores da XP

XP se baseia em quatro valores:

- Comunicação
- Feedback
- Simplicidade
- Coragem

3.1.1 – Comunicação

A comunicação é o principal valor da XP. Grande parte dos problemas de projetos é causada por falhas na comunicação ou até mesmo ausência dela, além do que a maioria das técnicas da XP está baseada na comunicação, e se esta não for eficiente, pode causar problemas e atrasos no desenvolvimento do sistema. Falhas na comunicação não acontecem intencionalmente, há muitas circunstâncias envolvidas, casos como: o desenvolvedor reporta ao gerente um erro no sistema e logo é repreendido ou punido, o cliente diz algo importante ao desenvolvedor que ignora o fato.

Dado que a maioria dos sistemas são feitos por equipes é essencial que essa se comunique da melhor forma possível, sempre trocando idéias e compartilhando conhecimentos e informações para que o software ganhe forma e atinja o objetivo desejado.

Praticamente todas as equipes hoje em dia buscam uma maior comunicação, porém elas focam em objetivos equivocados como, por exemplo, a comunicação escrita, que é um mecanismo pouco eficaz de comunicação.

Vendo que o tipo de comunicação influencia bastante à compreensão da idéia que se quer passar a XP preza o método face a face para a comunicação por ser um método onde se tem mais elementos interpretativos como os nossos gestos, nossa expressão facial, o tom de voz, entre outros. Com isso estimulamos vários dos nossos sentidos ficando a compreensão da idéia mais fácil.

Como sabemos o dia-a-dia de um profissional de informática é composto primordialmente por atividades sociais que envolvem muita comunicação e a necessidade de trabalhar em equipe cooperativamente. Ou seja, atividades como levantamento de requisitos, analise, design, teste, negociação de escopo, negociação de prazos, entre tantas outras, fazem parte do quotidiano deste profissional e requerem uma grande capacidade de comunicação e interação social para serem realizadas adequadamente.

Na XP uma equipe deve ter meios de se comunicar de maneira rápida e eficiente com o cliente, com o gerente do projeto e com os outros desenvolvedores envolvidos. Os desenvolvedores podem se comunicar livremente com os clientes e usuários e são autorizados a prover qualquer informação que seja requerida. Programadores e usuários decidem juntos estratégias de arquitetura e modelagem. Logo, contatos freqüentes permitem reduzir o risco de não aderência do produto à necessidade real.

Uma vez que a comunicação não se limita a procedimentos formais pode-se usar e-mail, bate papos, telefonemas, conversas informais, etc. Porem se dá preferência ao tipo de comunicação mais ágil, como telefonemas a e-mails, presença física melhor que comunicação remota, etc. A comunicação também pode ser melhorada entre os desenvolvedores com códigos auto-explicativos sendo melhor que documentação escrita.

3.1.2 – Simplicidade

Simplicidade não é fácil. A coisa mais difícil do mundo é não olhar para frente, para as coisas que terão que ser implementadas amanhã, na próxima semana e no próximo mês. Mas ficar compulsivamente pensando no futuro é dar ouvidos ao medo da curva do custo exponencial da mudança.

XP incentiva ao extremo práticas que reduzam a complexidade do sistema dado que as soluções a serem adotadas devem ser sempre as mais simples que alcance os objetivos esperados. O foco é realizar o serviço, do modo mais simples e eficaz possível. Em desenvolvimento de software nada é melhor que um design limpo, claro e eficaz.

Dada uma funcionalidade a ser implementada esta tem que ser codificada da maneira mais simples possível, ou seja, esta funcionalidade tem que ser desenvolvida apenas com o suficiente para que o cliente tenha seu pedido atendido e possa validar se foi atendido da forma que gostaria.

Um dos erros mais freqüentes que vai contra o principio da simplicidade é o trabalho especulativo. Este trabalho é feito utilizando premissas das quais não se tem certeza. Ocorrendo alguma dúvida o desenvolvedor não pode assumir que sabe a resposta, fazendo isso ele poderá estar implementando algo erroneamente e terá que ter um grande re-trabalho ao final, quando descobrir que a sua visão de negocio não era necessariamente a visão do cliente.

Porem pior que assumir posições equivocadas é criar soluções excessivamente sofisticadas para um dado problema, as chamadas generalizações. Elas frequentemente são desnecessárias, portanto todo o esforço feito para implementá-las é perdido.

A XP está fazendo uma aposta. Ela está apostando que é melhor fazer algo simples e de desenvolvimento rápido hoje, e ter de gastar um pouco mais no futuro se for necessário remodelar um processo do que implementar muitas funcionalidades que julga-se que possam vir a ser importantes num futuro, e que acabem nunca sendo necessárias.

“O principal objetivo da simplicidade é, portanto, evitar o re-trabalho que resulta do desconhecimento ou da precipitação.” (TELES, 2004).

3.1.3 – Feedback

“O feedback é a realimentação que o cliente fornece à equipe de desenvolvimento quando aprende algo novo sobre o sistema, quando aprende mais sobre os requisitos ou quando aprende mais sobre a forma como foram implementados.” (TELES, 2004).

Varias práticas da XP garantem um rápido feedback sobre várias etapas do processo. Podemos obter feedback sobre qualidade do código com os testes de unidade, programação em pares e posse coletiva, dentre outras, podemos obter feedback sobre o estado do

desenvolvimento com as estórias do usuário, a integração contínua e o jogo do planejamento, dentre outras.

Isto nos permitirá maior agilidade, pois erros são detectados e corrigidos imediatamente, requisitos e prazos são reavaliados mais cedo facilitando a tomada de decisões e permitindo estimativas mais precisas, e teremos uma maior segurança e menos riscos para os investidores.

O cliente deve receber o sistema o quanto antes, a fim de poder dar um feedback rápido, guiando assim o desenvolvimento do software. Quanto mais cedo o cliente tiver com o sistema em produção, mais rápido podem ser feitos os ajustes para que o mesmo fique de acordo com o que o cliente realmente quer.

Quanto mais cedo um erro é identificado, mais rápido e barato é o seu conserto. A XP encoraja o máximo de validações intermediarias realizadas pelos clientes durante o projeto quanto for possível. Se possível, o melhor é iniciar o uso dos módulos prontos, enquanto o restante ainda está sendo produzido.

Com a utilização extrema do feedback todo problema é evidenciado o mais cedo possível para que se possa ser corrigido o mais cedo possível. Toda oportunidade é descoberta o mais cedo possível para que possa ser aproveitada o mais cedo possível.

O feedback pode ser tanto do cliente para o desenvolvedor, quando este aprende alguma coisa do sistema, como do desenvolvedor para o cliente, que ocorre quando estes apresentam estimativas, apresentam riscos técnicos ou até sugerem alternativas de design.

Vale ressaltar que a idéia do feedback já existia nos processos tradicionais, o que a XP prega é que o tempo de feedback seja reduzido a horas, dias ou semanas e não a meses ou quem sabe anos. Quanto mais curto o tempo de feedback melhor, pois assim garantimos que estaremos trabalhando no que realmente importa para o cliente.

Uma questão interessante a respeito do feedback é a psicologia do aprendizado, que diz que o tempo entre uma ação e o seu feedback é crítico para o aprendizado. Ou seja, quando você recebe uma repreensão por alguma ação logo em seguida da ação fica fácil à assimilação da causa, logo se for repreendido tempos depois não conseguirá assimilar a repreensão à ação feita. Isto é exatamente o que acontece com os desenvolvedores, dada uma funcionalidade implementada erroneamente mais fácil será corrigí-la o quanto antes o erro for reportado, pois assim o desenvolvedor ainda terá “frescas” na cabeça as idéias a respeito.

Um dos riscos que enfrentamos em projetos é o otimismo, porém o feedback é um tratamento para isto, uma vez que não deixará a equipe se perder no projeto.

Veremos na ilustração a seguir o ciclo de vida do feedback no projeto.

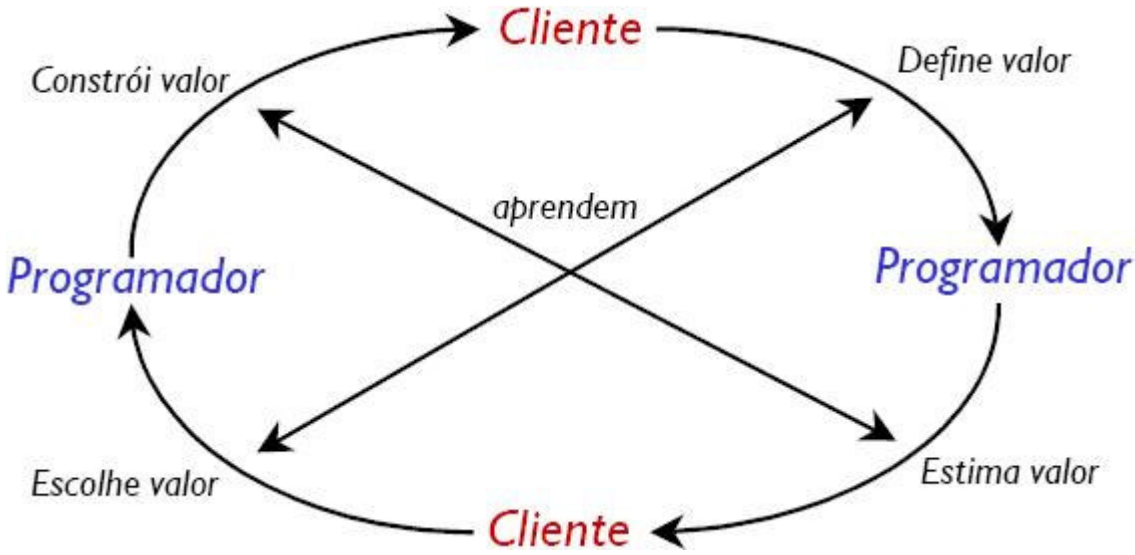

Figura 3.1 – O ciclo de vida do feedback. Fonte: www.argonavis.com.br.

3.1.4 – Coragem

É preciso muita coragem para mudar a maneira pela qual desenvolve-se sistemas. Colocar um sistema em produção assim que ele tenha valor para o cliente, fazer apenas o que se precisa para o momento e calcar o processo de análise principalmente na comunicação não é fácil, e precisa que a equipe esteja realmente decidida a mudar o seu processo de desenvolvimento.

É preciso coragem para apontar um problema no projeto, parar quando você está cansado, pedir ajuda quando necessário, simplificar código que já está funcionando, dizer ao cliente que não será possível implementar um requisito no prazo estimado, fazer alterações no processo de desenvolvimento. Ou seja, fazer a coisa certa mesmo que não seja a coisa mais popular no momento.

Com isso temos que assumir uma postura proativa frente ao desenvolvimento do sistema, assumindo a responsabilidade pelo sucesso do projeto como um todo.

O desenvolvimento guiado por testes, integração continua, programação em pares e outras técnicas do XP aumentam a confiança do desenvolvedor e ajudam-no a ter coragem para:

- Melhorar o design de código que está funcionando para torná-lo mais simples;
- Jogar fora código desnecessário;
- Investir tempo no desenvolvimento de testes;
- Mexer no design em estágio avançado do projeto;

- Pedir ajuda aos que sabem mais;
- Abandonar processos formais e fazer design e documentação em forma de código.

"A XP é um processo de desenvolvimento de software novo e se baseia em diversas premissas que contrariam os processos tradicionais de desenvolvimento." (TELES, 2004). Sendo assim a adoção da XP exige que a equipe de desenvolvimento tenha bastante coragem para aplicar suas técnicas e práticas.

Coragem por si só não vai te ligar ao XP, porém se combinada com os outros três valores de XP (Comunicação, Simplicidade e Feedback) a coragem se torna extremamente valiosa para o processo.

3.2 – Formação da equipe

No mundo do desenvolvimento de software sempre estamos cumprindo regras em todo o ciclo do projeto, em XP essas regras são formadas por experiências positivas obtidas ao longo do tempo. O XP recomenda não tentar mudar as pessoas que pertencem ao projeto, não muito, caso alguma regra não esteja sendo cumprida pela equipe é recomendado tentar mudar a regra, baseado nas opiniões da equipe a fim de melhorar a integração de toda a equipe com as regras. As regras do XP são importantes e devem ser seguidas, mas no final são somente regras, as quais podem ser adaptadas e aprimoradas conforme a necessidade, desde que haja um consenso no time de desenvolvimento sobre o impacto de tal mudança.

Dentre estas regras podemos definir os papéis de cada membro da equipe de XP, sendo eles compostos por:

- Gerente de projeto
- Treinador
- Analista de testes
- Redator técnico
- Rastreador
- Desenvolvedor

É importante salientar que pessoas podem ocupar mais de um papel dentro da equipe, e que como a modelagem, o projeto e o desenvolvimento são feitos por todos a todo instante, os papéis de engenheiro de software e de analista de sistemas não existem neste tipo de equipe da maneira pela qual se está acostumado a ver. Na verdade, na XP todos os desenvolvedores são também engenheiros e analistas.

Um outro fator interessante também é que papéis como o rastreador, redator técnico, analista de teste e treinador são importantes em alguns projetos, mas não têm necessariamente que estar presentes em todos os projetos XP. Certamente, espera-se que todos os projetos XP tenham, no mínimo, a participação de desenvolvedores e usuários. Os demais papéis devem ser considerados se as necessidades do projeto justificarem ou demandarem os mesmos. Por exemplo, alguns projetos têm baixa necessidade de documentação, portanto, pode-se considerar desnecessário o redator técnico. Outros possuem um nível baixo de automação de testes de aceitação, demandando assim a participação de um analista de teste. Enfim, tudo depende da necessidade cada projeto. O que não varia é a necessidade de desenvolvedores e usuários, juntos, na equipe de desenvolvimento XP.

3.2.1 – Gerente de Projeto

O gerente de projeto é o responsável pelo projeto como um todo, dentre suas atribuições destacamos o relacionamento com o cliente, a formação da equipe, a obtenção de recursos e a gerência de problemas e pessoas. É o gerente que tem a visão do todo em um projeto.

Ele é responsável por marcar reuniões com o cliente para reportar toda a situação do projeto assim como dificuldades e riscos enfrentados. É do gerente a responsabilidade de manter o cliente sempre presente no ambiente de projeto, uma vez que esta prática é essencial para o sucesso do projeto. Igualmente importante é a necessidade de fornecer material de trabalho para a equipe de desenvolvimento, desde equipamentos, softwares e até material de escritório para o perfeito funcionamento dos trabalhos.

É muito importante que o gerente conheça os valores e práticas de XP, porque ele tem que trabalhar sempre em sintonia com o treinador, que é o responsável técnico do projeto, uma vez que em XP os parâmetros de gerencia são específicos e particulares.

Por ser um modelo que difere dos modelos comumente usados, não é recomendado que gerentes com visões industriais gerenciem projetos que utilizem XP, pois isso representa uma verdadeira ameaça ao projeto já que as idéias do XP geralmente vão contra os modelos industriais. Gerentes com estas visões dificilmente aceitam algumas práticas de XP o que pode vir a gerar grandes conflitos entre Gerente e o treinador que tem por obrigação assegurar que os valores e práticas do XP sejam devidamente respeitados.

Um gerente de XP tem que passar segurança e coragem para sua equipe, utilizando-se de um dos valores de XP, a comunicação. Em conversas freqüentes e honestas o gerente tem que passar confiança para sua equipe, uma vez que algumas práticas de XP podem parecer "loucas" para alguns, como a forte ênfase nos testes e a programação em par.

De acordo com Teles (2004, p.205) uma frase pode definir bem o legado do gerente: "O seu sucesso depende da sua capacidade de sintonizar equipe e cliente, buscando sempre o equilíbrio entre ambas as partes".

Como vimos anteriormente os membros da equipe podem assumir mais de um papel, o Gerente também pode assumir o papel do treinador e do Redator Técnico simultaneamente, porém isso só é viável em projetos pequenos com pequenas equipes uma vez que o treinador e o redator técnico tem que estar muito ligado à equipe. Se ele tiver que estar saindo sempre para resolver problemas administrativos e se reunindo com o cliente isto com certeza afetará bastante o desenvolvimento. Além de ser emocionalmente difícil lidar com estes cargos simultaneamente.

Como XP é bastante disciplinado existem coisas que não devem ser feitas por um Gerente, tais como:

- Não atribuir prioridades, o cliente fará isso;
- Não atribuir tarefas, os desenvolvedores farão isso;
- Não estimar tempo para estórias ou tarefas, os desenvolvedores farão isso;
- Não ditar horários, o cliente e os desenvolvedores negociarão isso.

3.2.2 – Treinador

O treinador é o responsável pelo gerenciamento da equipe. Ele deve sempre motivar a equipe a não perder o foco, a não abandonar as técnicas da XP e deve auxiliar a equipe em tudo que for possível. O treinador é fundamental, pois a XP diferentemente dos modelos tradicionais atua sobre o comportamento das pessoas e não é eminentemente prescritivo. "Ele possui bastante conhecimento técnico e conhece profundamente o processo de desenvolvimento" (TELES, 2004).

Sendo o responsável pelo processo o treinador verifica e faz correções de eventuais erros, além de orientar e guiar a equipe distribuindo responsabilidades, tornando cada membro parte integrante e importante da equipe. Ele também é o responsável pela negociação com o cliente quanto ao escopo de cada iteração e pela coordenação do jogo do planejamento.

Ele não só aplica as práticas como é um profundo conhecedor dos valores da XP, mantendo sempre a disciplina. Um desenvolvedor pode assumir o papel de treinador assim garantindo estar sempre presente, para o melhor gerenciamento possível.

A idéia de um treinador não é a de um programador líder ou de um arquiteto de sistemas, mas sim a de um bom comunicador para poder ajudar a equipe sempre que possível.

Algumas das qualidades de um treinador seria a maturidade, se não calmo, centrado também, o treinador seria uma pessoa bastante respeitada e acima de tudo respeitador, uma pessoa disposta a falar, mas também a escutar. Na maioria dos casos o treinador é um programador especialista, experiente e competente.

Alguns especialistas dizem que a principal tarefa do treinador é adquirir brinquedos e comida para o ambiente de desenvolvimento. Em projetos que utilizam XP existem no ambiente, lugares destinados ao descanso e a alimentação.

De acordo com Beck (2000, p.81) são definidas algumas tarefas pertinentes ao treinador, são elas:

- Estar disponível para um parceiro de programação, principalmente se forem programadores começando a pegar responsabilidades ou para implementar tarefas difíceis;
- Caso esteja acontecendo grandes refinamentos, incentivar que sejam feitos em pequena escala, pouco a pouco e não tudo de uma vez;
- Ajudar desenvolvedores com perícias técnicas individuais, como testes, formatação e refinamento de código;
- Explicar o processo para gerentes.

3.2.3 – Analista de Testes

"O analista de testes é a pessoa responsável por testar e garantir a qualidade do sistema." (TELES, 2004). Ele deve testar todo o software pelo menos uma vez por dia. Todo o erro encontrado é reportado aos desenvolvedores, sendo assim o analista de testes é considerado o "chato" da equipe, uma vez que quanto mais o analista trabalha mais trabalho ele produz para os desenvolvedores.

É o analista de testes quem ajuda o cliente a escrever os testes de aceitação, que são efetuados sobre cada funcionalidade, ou estória do sistema.

Como vimos anteriormente membros da equipe de trabalho podem assumir mais de um papel, e em muitos projetos dependendo da carga de trabalho o papel de analista de sistemas pode ser feito em conjunto com o redator técnico sem problemas. Porém é altamente recomendado que o analista não seja um desenvolvedor e que não esteja envolvido com nenhuma tarefa de codificação, podendo o projeto correr sérios riscos caso isto ocorra. Isto porque o analista de testes tem que possuir uma visão de fora, uma visão imparcial sem tendências já que ele trabalha como se fosse o cliente.

3.2.4 – Redator Técnico

O redator técnico é um profissional que auxilia a equipe na geração da documentação do projeto. A presença dele pode ser útil em alguns casos para reduzir a carga de trabalho sobre os desenvolvedores e assegurar que os mesmos se concentrarão basicamente na implementação de funcionalidades e não na documentação.

Com já foi dito o analista de testes pode tranquilamente, dependendo da carga de trabalho, assumir este papel.

É imprescindível que o redator técnico trabalhe em sintonia com desenvolvedores e cliente para estar sempre com a documentação atualizada, pois uma vez que tenha uma documentação defasada, que não reflete a realidade, você corre o risco de tomar decisões baseadas em dados irreais.

3.2.5 – Rastreador

O rastreador é uma pessoa que ajuda a registrar o desempenho da equipe ao longo de uma iteração. Isto é, ele coleta informações que serão úteis para calcular a velocidade da equipe na iteração, a qual é utilizada para definir a carga de trabalho da iteração seguinte. Mas este papel pode ser descartado dependendo da abordagem utilizada. Podendo a coleta de dados ser feita de forma coletiva, diariamente, na reunião rápida e registrada em um quadro ou em documentação. Essa abordagem elimina a necessidade do rastreador.

As funções do rastreador são basicamente, coletar sinais vitais do projeto (métricas) uma ou duas vezes por semana, manter todos informados do que esta acontecendo e tomar atitudes sempre que as coisas parecerem ir mal.

3.2.6 – Desenvolvedor

O desenvolvedor é o coração do processo em XP, é ele quem lida com decisões a respeito do software uma vez que ele é a pessoa que analisa, projeta e codifica o sistema, ou seja, é a pessoa que efetivamente constrói o software.

Na XP não há divisões entre estes cargos, o desenvolvedor tem que ter humildade e habilidade para desempenhar todos os papéis que o desenvolvimento demanda.

A programação em par é uma grande aliada dos desenvolvedores, pois ajuda a igualar o conhecimento entre toda a equipe, através da troca de conhecimento toda a equipe amadurece suas idéias.

Superficialmente o desenvolvimento em XP não se distingue de outras metodologias, porem o foco é um pouco diferente, você trabalha acima de tudo perseguindo a perfeição, fazendo que o cliente receba o máximo de valor de cada dia de codificação.

Algumas habilidades são requeridas para programadores em XP, devido à introspectividade da maioria de programadores, isso acontece no Brasil e no mundo, a comunicação entre os membros da equipe é vital para o sucesso do projeto, um programador também deve ter simplicidade, uma vez que deve aceitar naturalmente o caso de outro programador alterar o seu código, ele deve aceitar e aprender com isso. E antes de tudo ser corajoso, pois todos têm medo, medo de parecermos idiotas, inúteis e descobrir que não somos tão bons, porem um programador corajoso tentará constantemente não falhar, estará disposto a pedir ajuda a sua equipe e reconhecer seus medos.

Diferentemente dos desenvolvedores de metodologias tradicionais que projetam, codificam e testam os desenvolvedores XP primeiro criam os testes, codificam e depois refinam o código. A figura a seguir demonstra o ciclo de um desenvolvedor XP.

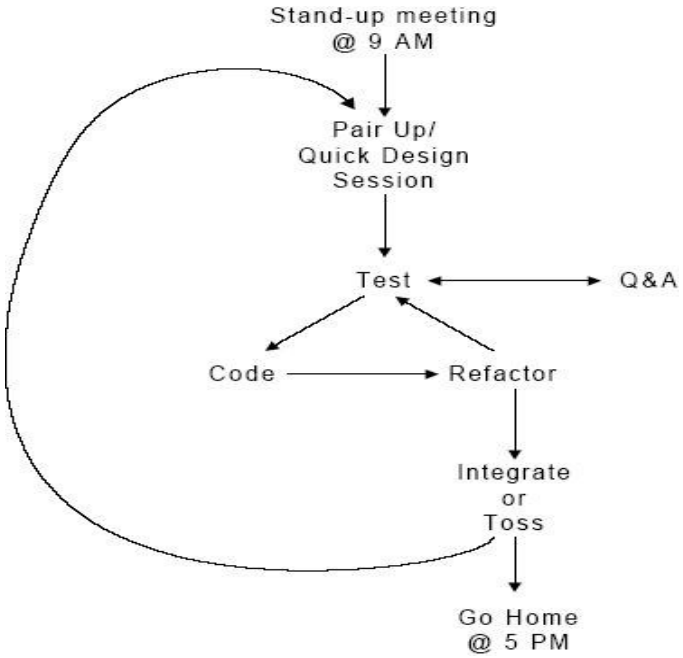

Figura 3.2 – Ciclo de atividades de um desenvolvedor em XP. Fonte: WAKE, 2000.

De acordo com a figura 3.2 podemos verificar que o dia de um desenvolvedor começa na reunião rápida (Stand up meeting) onde ele se reúne com a equipe por aproximadamente 10 minutos para identificar os problemas e decidir quem irá resolvê-los. Seguindo, o desenvolvedor escolhe seu par de programação e faz um rápido projeto do dia (Pair up / Quick Design Session), onde quem digita pensa taticamente e o parceiro de programação pensa estrategicamente. Logo ele entra em definitivo na codificação lembrando que primeiro ele escreve os testes (Test) para depois codificar (Code) o sistema, no ato de codificar podem surgir algumas dúvidas (Q&A) que serão sanadas pelo cliente, após codificar ele se possível refina (Refactor) o código implementado. Uma vez feito isso ele faz a integração (Integrate) de seu código com o sistema ou descarta (Toss) este código para confeccionar outro. No final do dia, terminado seu trabalho ele vai tranqüilo para casa (Go Home), seguindo um ritmo de trabalho sustentável.

Alguns aspectos dos programadores que são importantes em XP:

- Sempre tem um parceiro;
- Obtém respostas antes de codificar;
- Utilizam o desenvolvimento guiado por testes;
- Estão sempre dispostos a ajudar um parceiro;
- Vão para casa com a cabeça limpa.

3.3 – A organização do ambiente de trabalho

"A organização do ambiente de trabalho tem importância vital para um projeto em XP porque a forma como a equipe se organiza influencia diretamente a sua capacidade de adotar as práticas do XP." (TELES,2004).

XP recomenda que o ambiente de trabalho público e aberto e antes de tudo confortável para os membros da equipe trabalhem, proporcionando divertimento enquanto se trabalha. O ambiente deve proporcionar aos membros da equipe escutar eventuais perguntas de seus colegas, trabalharem em conjunto, próximos uns dos outros, e até escutar uma conversa de colegas de equipe para quem sabe poder ajudá-los.

O ambiente deve possuir mesas e cadeiras confortáveis que proporcionem a aplicação das práticas de XP como a programação em par, é altamente recomendável o uso de cadeiras ergonômicas, pois o afastamento de algum membro da equipe por doenças, por exemplo, a LER, pode causar grande perda para a equipe e prejuízos para o projeto. Podemos ver na figura a seguir um exemplo de ambiente de trabalho da XP:

Figura 3.3 – Ambiente de trabalho do XP. Fonte: Beck, 2000.

A XP também recomenda o uso de computadores muito potentes, com muita memória e alto poder de processamento, alem da tarefa de desenvolvimento de softwares já ser "pesada" a XP trabalha com o desenvolvimento guiado por testes, o que significa que testes em sua totalidade serão executados diversas vezes ao dia enquanto o desenvolvimento prossegue e para isto ser viável é essencial trabalhar com computadores poderosos.

Geralmente é separado um computador para a integração, isto serve para que a equipe saiba quando um par esta fazendo sua integração, o que ajuda cada par a saber sua hora de integrar uma vez que não é permitido a mais de um par fazer a integração ao mesmo tempo.

Figura 3.4 – Quadros e murais em um ambiente de trabalho utilizando XP. (Fonte: http://fairlygoodpractices.com/).

Podemos ver na figura 3.4 a utilização de quadros e murais, esses utensílios são bastante úteis para a comunicação da equipe podendo ser utilizados para a alocação de cartões de iterações e também como apoio as discussões e debates da equipe.

Figura 3.5 – Espaço para lanche em ambiente de trabalho utilizando XP. (Fonte: http://fairlygoodpractices.com/).

Como vemos na figura 3.5 um outro fator importante no ambiente de trabalho é a comida, sempre se procura ter algum tipo de comida no ambiente tais como biscoitos, chocolate, pipoca, doces ou frutas. “A comida é um elemento que ajuda a socializar, além de diminuir o stress e fornecer energia para os desenvolvedores.” (TELES, 2004).

Estabelecer um ambiente de trabalho saudável para a equipe é uma forma extremamente eficaz de obter ganhos de produtividade e evitar desperdícios de recurso no projeto. Os resultados demonstram claramente que o ambiente de trabalho afeta profundamente a produtividade, portanto, deve-se dedicar o máximo de atenção e esforço no sentido de melhorar este ambiente.

3.4 – Práticas da XP

A XP se baseia na sinergia entre práticas simples, que muitas vezes foram abandonadas décadas atrás por serem consideradas simplórias, simples demais ou impraticáveis.

3.4.1 – Cliente Presente (On-site Customer)

Na XP todas as decisões sobre o rumo do projeto devem ser tomadas pelo cliente. Ele deve priorizar as tarefas, ser responsável pelos testes de aceitação, e, acima de tudo, orientar e tirar dúvidas dos desenvolvedores durante o processo de programação. É por isto que as estórias do usuário, os cartões nos quais os requisitos do sistema são passados pelo cliente, têm apenas tópicos sobre o que deve ser feito, e não uma descrição detalhada das tarefas, algo que será feito pelo cliente durante o andamento do projeto.

“O XP trabalha com uma visão diferente. Ele 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.” (TELES, 2004).

Esta técnica encontra resistência no fato de que muitas vezes as empresas acham que é muito caro afastar um membro de sua equipe de suas tarefas, para que ele possa ficar acompanhando o desenvolvimento de um software. Por isto é recomendável que seja provida para estes clientes estrutura para que eles possam continuar fazendo algum trabalho enquanto acompanham a equipe de desenvolvimento.

Se o cliente não puder estar junto dos desenvolvedores, algumas técnicas podem ser utilizadas, como, por exemplo, nomear um membro da equipe para representar o cliente. Alguém que, preferencialmente, já tenha algum conhecimento na área em que o software está sendo desenvolvido e que possa centralizar a comunicação com o cliente real, absorvendo os questionamentos dos desenvolvedores e repassando-os ao cliente real. Assim, esta pessoa vai acabar conhecendo bem o processo como um todo e vai passar a poder responder a maior parte das dúvidas dos desenvolvedores.

Também pode-se agendar reuniões de acompanhamento com o cliente, pelo menos uma vez por semana. Nestas reuniões haverá muita discussão sobre prioridades, mas deve-se destinar uma parte significativa da mesma para que os programadores possam saber o que e como desenvolver.

Kent Beck, em seu livro Extreme Programming Explained, apresenta uma metáfora onde compara o desenvolvimento de software a dirigir um carro. Dirigir um carro não é apenas coloca-lo na direção certa e deixa-lo ir, você tem que fazer pequenos ajustes, com pequenos toques no volante para que o carro não saia da pista. Assim é o desenvolvimento de software em XP, com o cliente presente para que possa fornecer vários pequenos feedbacks para a equipe assim como a equipe fornecer vários feedbacks para o cliente. Com isso todos trabalham para o sucesso do projeto, pois sabemos que a única constante nos projetos de XP é que mudanças acontecem. (BECK, 2000).

3.4.2 – Jogo do planejamento (The Planning Game)

"O XP utiliza o planejamento para assegurar que a equipe de desenvolvimento esteja sempre trabalhando naquilo que irá gerar maior valor para o cliente a cada dia de trabalho. Este planejamento é feito diversas vezes ao longo do projeto, para que todos tenham a oportunidade de revisar as prioridades". (TELES, 2004).

O jogo do planejamento é o coração do planejamento e acompanhamento de projetos na XP. É nele que o software é efetivamente planejado pela equipe de desenvolvimento e pela equipe do cliente. Nesta atividade, cada equipe tem o seu papel bem claro e definido: a equipe do cliente é responsável por definir escopo e prioridades; e a equipe de desenvolvimento por fornecer estimativas.

Muitas vezes é difícil manter as equipes focadas exclusivamente no seu papel. Por isto, cabe ao treinador da equipe de desenvolvimento ficar alertando sobre possíveis desvios.

Os desenvolvedores normalmente gostam de novos desafios e tendem a tentar "puxar" as tarefas mais desafiadoras ou interessantes para o começo. Mas, esta decisão deve ser apenas da equipe do cliente, não cabe ao desenvolvimento. Por outro lado, é muito comum o cliente querer fazer as estimativas pela equipe de desenvolvimento. É comum a frase "preciso desta funcionalidade em tantos dias". Acontece que o cliente não tem o conhecimento técnico necessário para fornecer estimativas de desenvolvimento.

Este é um ponto que deve estar presente a todo o momento na cabeça do treinador, que deve guiar com segurança a sua equipe e, por outro lado, não deve aceitar tudo que venha do cliente para não comprometer a credibilidade do projeto com constantes atrasos. Muitas tarefas não podem ser feitas mais rápido apenas colocando mais desenvolvedores no projeto. Inclusive, esta abordagem, normalmente, tem efeito contrário, pois estes novos membros precisam de tempo para ser treinados e se interarem do sistema sendo desenvolvido. "Uma mulher leva nove meses para gerar uma criança. Não adianta pegar nove mulheres que elas não vão gerar uma criança em um mês". (BECK, 2000).

Isto gera a declaração de direitos do cliente e do desenvolvedor (JEFFRIES et al., 2001):

Direitos do cliente

- Você tem o direito a um plano geral, para saber o que pode ser realizado, quando e a que custo;
- Você tem o direito de receber o maior valor possível por cada semana de desenvolvimento;
- Você tem o direito de ver o progresso de um sistema sendo executado, que prova seu funcionamento ao passar testes repetidos que você mesmo especifica;
- Você tem o direito de mudar de idéia, substituir funcionalidade e mudar suas prioridades sem pagar custos exorbitantes;
- Você tem o direito de ser informado de mudanças de cronograma, em tempo de escolher como priorizar o escopo para restaurar a data original. Você pode cancelar o projeto a qualquer tempo e ainda assim ter um sistema funcionando refletindo o investimento feito até o momento.

Direitos do desenvolvedor

- Você tem o direito de saber o que é necessário, com declarações claras de prioridade;
- Você tem o direito de produzir trabalho de qualidade o tempo todo;
- Você tem o direito de pedir e receber ajuda de seus pares, superiores e clientes;

- Você tem o direito de fazer e atualizar suas próprias estimativas;
- Você tem o direito de aceitar as suas responsabilidades, ao invés de tê-las impostas a você de cima para baixo.

Estando os papéis das duas equipes bem claros, pode-se entender o que efetivamente ocorre no jogo do planejamento. O cliente deve ter as estórias já definidas, pois é sobre elas que está baseada esta atividade. A figura 3.6 mostra um exemplo de cartão de estórias do usuário, que é utilizado nesta tarefa para que se faça o planejamento.

Com o cliente escrevendo suas estórias fazemos com que ele pense melhor na funcionalidade, é recomendável que esta estória seja curta, pois será usada como um lembrete e na hora de desenvolver será feito um detalhamento desta funcionalidade através da comunicação com o cliente presente.

Baseados em sua experiência ou conhecimentos anteriores, os desenvolvedores devem fornecer estimativas de desenvolvimento para cada estória. Se uma estória for muito grande para entrar em uma iteração ou não puder ser estimada adequadamente, significa que a mesma está muito complexa e deve ser segmentada em estórias menores.

Por outro lado, se for muito pequena, deve ser agregada a outras para formar uma estória de tamanho adequado. Normalmente, entende-se por adequadas estórias que possam ser desenvolvidas em dois ou três dias ideais de programação. A relação entre as funcionalidades entregues e as funcionalidades solicitadas por um cliente para uma iteração é chamada de velocity (velocidade) da equipe de desenvolvimento.

Data: _____ Tipo: Novo__ Conserto__ Melhoria__

Story #: _____ Prioridade Cliente: ____ Técnica: _____

Referência Anterior: _____ Risco: _____ Estimativa: _____

Descrição:

Notas:

Tarefas:

Data	Concluído	A Fazer	Comentários

Figura 3.6 – Cartão de estória do usuário. Fonte: POHREN, 2004.

Como vemos na figura 3.6 este é um cartão de estórias do usuário, nele o usuário indica as funcionalidades que deseja do sistema.

De posse dos cartões com as estórias do usuário e das estimativas da equipe de desenvolvimento, os clientes devem definir o escopo da próxima iteração, selecionando aquelas que desejam que sejam implementadas. Para tanto, devem levar em consideração o princípio de que devem receber antes o que vai gerar mais retorno para o negócio. Desta forma, busca-se que o retorno sobre o investimento comece a vir já nos primeiros releases.

Deve-se planejar no máximo duas iterações para que se evite muito retrabalho, uma vez que o feedback que o cliente vai passando em relação às iterações entregues pode vir a gerar novas estórias que precisarão ser estimadas, e entrar no planejamento da próxima iteração, e assim sucessivamente.

Durante o desenvolvimento, as dúvidas dos desenvolvedores podem vir a gerar novas estórias. Além disto, podem surgir necessidades técnicas, como por exemplo, a criação de uma interface para banco de dados, que podem vir a gerar tarefas de engenharia, que devem ser levadas em conta nas próximas iterações.

Nas reuniões do jogo do planejamento (planning game) também são repassadas para o cliente: as informações sobre o andamento do release atual e informações sobre eventuais atrasos no andamento do desenvolvimento, para que o cliente possa tomar a decisão sobre que funcionalidades devem ser deixadas para o próximo release, ou para que os desenvolvedores possam fornecer novas estimativas para as estórias ainda não implementadas.

Este é um ponto importante: caso não seja possível cumprir tudo o que havia sido planejado para a data de entrega de cada iteração, deve-se negociar com o cliente quais estórias devem ficar para a próxima, e não atrasada a data de entrega da iteração.

Na XP o planejamento é um processo contínuo, e o mesmo é constantemente refinado pelo cliente e pela equipe de desenvolvimento, deixando assim a metodologia bastante flexível e entregando para o cliente sempre o máximo valor pelo investimento dele. A figura 3.7 mostra um resumo desta importante tarefa de planejamento.

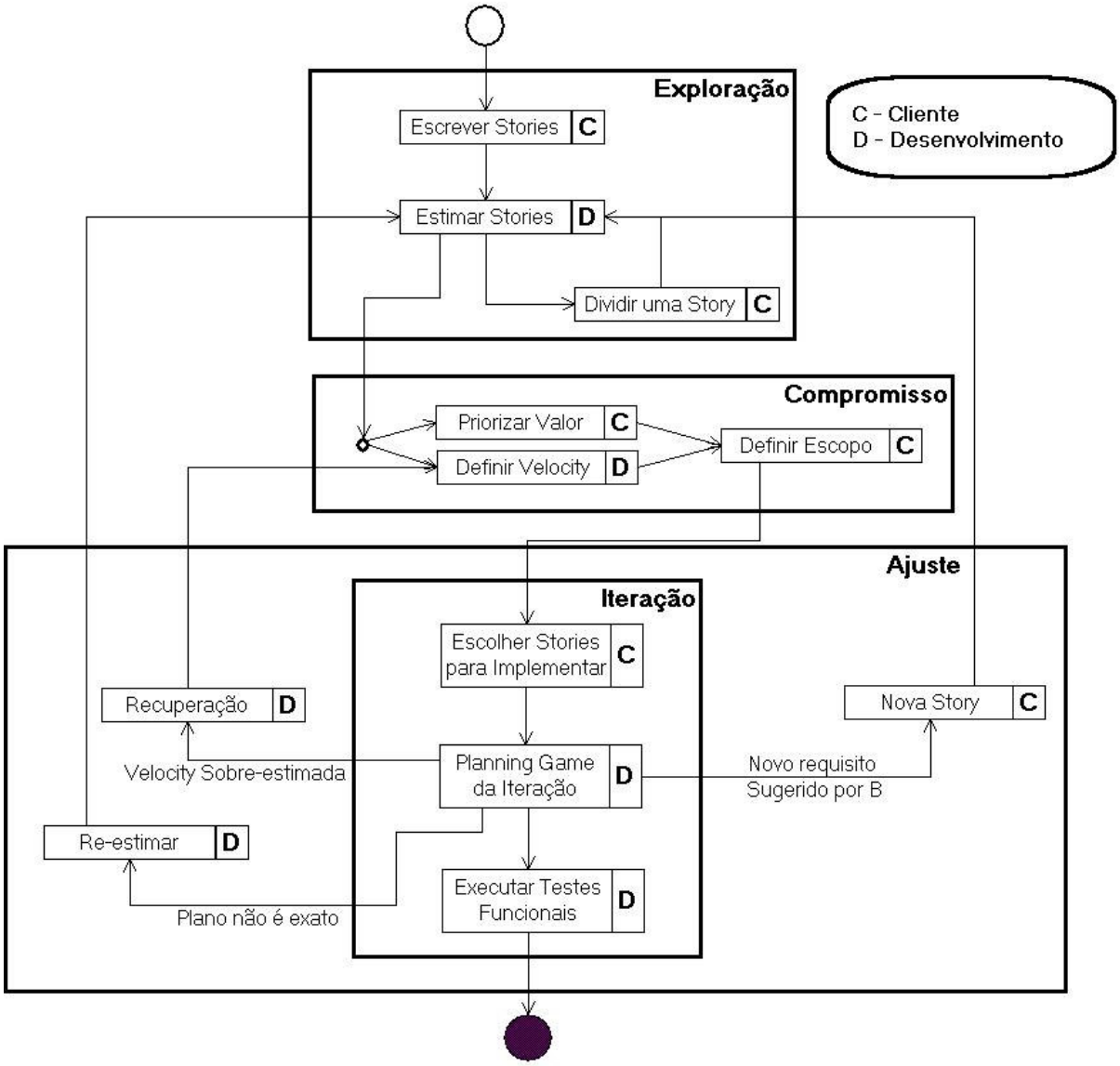

Figura 3.7 – O Jogo do Planejamento. Fonte: POHREN, 2004.

A figura 3.7 explica bem o planejamento de um release, em um primeiro momento temos a exploração onde o cliente escreve as estórias e o desenvolvedor estima estas estórias, ou seja, ele prevê um custo para cada uma delas, podendo o desenvolvedor estar dividindo estas estórias em partes menores. Depois vemos a fase de compromisso, onde o cliente define as prioridades o desenvolvedor a velocidade e assim o cliente tem a base do escopo deste release. Definido o escopo do release o cliente escolhe as estórias que quer ver implementadas e os desenvolvedores fazem o planejamento daquela iteração. Na fase de ajuste os desenvolvedores podem ter que re-estimar estórias ou ate redefinir a velocidade de implementação delas, visto que pode haver enganos nas primeiras estimativas, podem também surgir nesta fase um novo requisito que deve ser estimado pelo desenvolvedor. Definida a

iteração são executados todos os testes funcionais até que estes rodem em sua totalidade e obtendo 100% de acerto o release é liberado.

3.4.3 – Pequenos Lançamentos (Small Releases)

"Um release é um conjunto de funcionalidades implementadas que representam um valor bem definido para o cliente e que é colocado em produção para que todos os usuários possam se beneficiar dele." (TELES, 2004).

Para que o cliente possa fornecer constantemente feedback sobre o andamento do sistema, fazendo possível que o jogo do planejamento (planning game) seja executado, é importante que o sistema tenha releases pequenos, a fim de ajustar o desenvolvimento às necessidades que vão surgindo no decorrer do processo.

Normalmente, trabalha-se com pequenas iterações gerando releases mais curtos, mas algumas empresas vêm suprimindo a etapa das iterações, lançando direto um release por semana.

Quanto menor o intervalo de tempo entre cada versão que o cliente recebe, mais fácil será corrigir eventuais problemas, pois não terá sido entregue uma quantidade muito grande de funcionalidades, e mais fácil será fazer algum ajuste no software para atender a uma mudança de requisitos. Além disso, o XP recomenda que o cliente busque selecionar as funcionalidades que mais vão gerar valor para ele, com isso fazemos com que o cliente tenha um retorno de investimento o mais cedo possível.

Com pequenos lançamentos os usuários têm a chance de avaliar o software mais cedo, gerando um feedback mais rapidamente. Com isso temos a chance de reavaliar o direcionamento do projeto a fim de poder fazer pequenos ajustes no sentido de adequar o sistema para as reais necessidades dos usuários. Um outro fator interessante é a confiança entre a equipe e o usuário que vê suas necessidades sendo atendidas prontamente.

Também interessantes são as principais unidades de tempo utilizadas no método XP (BECK, 1999):

Liberação (release): O cliente escolhe o menor conjunto de estórias possível o qual faz sentido estando juntas. Estas estórias são implementadas em primeiro lugar, e colocadas em produção. As demais estórias são implementadas posteriormente. Um conjunto de estórias é selecionado para ser desenvolvido e forma uma liberação do sistema para a produção, que poderá ser medida na escala de semanas ou meses, de acordo com a figura 3.8. Isto é feito

através da negociação entre os clientes e o time de desenvolvimento, levando-se em consideração a prioridade de valor para o negócio – definida pelos clientes – e o risco técnico – definido pelos desenvolvedores.

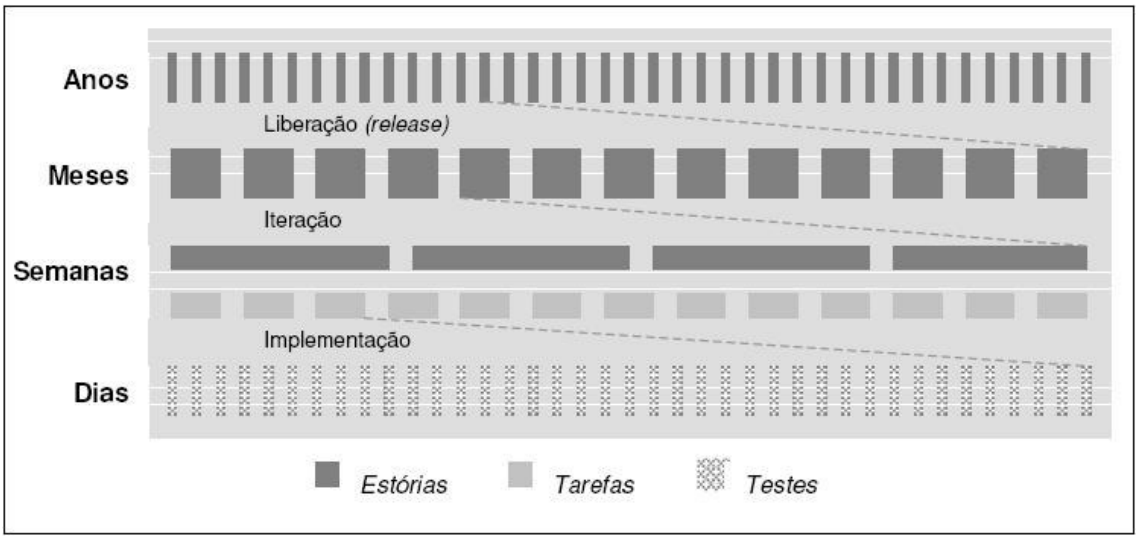

Figura 3.8 – Ciclos de desenvolvimento em XP. Fonte: BECK, 1999.

Iteração: Uma iteração inicia com um planejamento simples de quais serão as estórias a serem implementadas e quais as tarefas que cada programador deverá realizar. O objetivo da iteração é implementar algumas poucas estórias em um espaço curto de tempo, e disponibilizá-las em produção. Enquanto as funcionalidades estão sendo implementadas pelo time de desenvolvimento, o cliente especifica um conjunto de testes funcionais, os quais serão aplicados para verificar se o objetivo da iteração, ao seu final, foi devidamente atingido.

Durante a iteração as estórias são divididas em tarefas. Uma tarefa é uma unidade básica de trabalho no projeto, sendo que um programador deve poder realizá-la em alguns poucos dias. Após a realização de cada tarefa, o par de programadores integra o código com o repositório principal do projeto e roda testes unitários para verificar que seu trabalho está completo e não afetou outras porções da aplicação. Ao final da iteração, todos os testes unitários e funcionais devem rodar com sucesso.

Tarefa: As tarefas são implementadas por pares de programadores. Se for necessário, uma pequena reunião de 15 minutos pode ser realizada com o cliente para esclarecer detalhes da implementação, ou outros programadores que conheçam o código que vai ser alterado, em maior detalhe. Em seguida o par define quais serão os casos de teste necessários para verificar que a tarefa esteja completa. A partir de então, estes repetem um ciclo em que escolhem um caso de testes da lista e implementam o código necessário para atendê-lo. Idealmente todos os casos de teste são implementados sem problemas, mas pode ser que o par de programadores detecte que precisa fazer um refinamento no código para conseguir fazer com que o teste

funcione. Neste caso, o refinamento é feito de forma que todo o conjunto de teste passe. Após todos os testes passarem, se houver ainda oportunidade para refinamento, este é feito.

Teste: A técnica de testes unitários é uma das mais importantes no método XP. Os testes não somente fazem parte do dia-a-dia de cada programador, como também estes são escritos ainda antes de o código ser escrito. Segundo Beck (2000, p.91), se programação é um eterno aprendizado, então o melhor é que o aprendizado ocorra o mais cedo possível. Os testes fazem com que os programadores recebam feedback sobre o código que foi implementado por outros o mais rápido possível, e muitos destes aprendem a partir disto.

A principal diferença entre as liberações e as iterações é que as liberações são ditadas pelo ritmo do negócio, enquanto que as iterações são ditadas pelo ritmo do desenvolvimento (BECK, 2000).

Para definir uma liberação, é necessário que haja negociação entre os clientes e os programadores. Os clientes escolhem as estórias que são mais importantes de serem implementadas e que faça sentido estarem juntas. A quantidade de estórias que poderá ser implementada durante uma liberação deve ser definida considerando-se o histórico de velocidade do time de desenvolvimento. Cada estória deve ser estimada da melhor forma possível, conforme descrito anteriormente, utilizando comparações com estórias semelhantes implementadas no passado, sempre que possível.

O planejamento das iterações, por outro lado, é feito pelos desenvolvedores. O tempo de duração de cada iteração pode variar, sendo que uma duração de duas semanas é um tempo adequado, pois seria o suficiente para que o time (BECK, 2000):

1. Desenvolva algumas novas funcionalidades.
2. Faça um redesenho significativo.
3. Desenvolva alguma infra-estrutura necessária.
4. Faça alguns experimentos.
5. Se recupere de alguns erros ou frustrações.

Uma questão importante em relação ao método XP é que uma vez definido o objetivo de uma iteração, a data de fim da mesma deve ser respeitada. Seu escopo pode ser alterado e diminuído, mas a data não. O autor afirma que se um time se acostumada a tomar mais tempo que o planejado para terminar as iterações, então freqüentemente haverá problemas no projeto, sendo que datas mais importantes, como datas planejadas para as liberações, não serão respeitadas.

A solução para a situação em que há mais tarefas para fazer em uma iteração do que é possível é reduzir a funcionalidade. Segundo BECK (2000, p.100), somente quando o time e

os clientes se acostumarem a agirem desta forma em todas as iterações é que todos estarão prontos a agirem desta forma quando uma data realmente importante se aproximar. A estratégia consiste em garantir que as estórias mais importantes para os clientes – as que foram selecionadas primeiro – sejam implementadas, mesmo que o cliente seja forçado a protelar a implementação de algumas estórias menos importantes.

3.4.4 – Desenvolvimento guiado pelos testes (Test First Design)

Testar é uma parte muito importante em um projeto de software, porem a maioria dos programadores acha esta atividade muito chata e que gasta muito tempo, geralmente os testes são deixados para o final do projeto, só que, a maioria dos projetos atrasam ficando o tempo destinado a testes prejudicado. Normalmente os testes só são valorizados quando o software entra em produção e apresentam muitos erros, estes que poderiam ser sanados com testes. Para fazer com que os testes não sejam a parte "chata" do desenvolvimento é necessário que estes não tomem muito tempo, sejam simples de serem executados e principalmente façam parte natural do ato de programar.

O desenvolvimento guiado pelos testes define que qualquer método de um objeto que possa falhar deve ter um teste automatizado que garanta o seu funcionamento. É muito difícil saber previamente que métodos possam vir a falhar, a fim de que não se esqueça de ter um teste automatizado para métodos que sejam importantes e também não se perca tempo escrevendo testes para rotinas que não tem como conter erro, por serem extremamente simples. Apenas com a experiência se vai conseguindo distinguir o que deve ter um teste automatizado.

O XP trabalha com dois tipos de testes: os testes de unidade e os testes de aceitação.

"O teste de unidade é aquele realizado sobre cada classe do sistema para verificar se os resultados gerados são corretos. Já os testes de aceitação são efetuados sobre cada funcionalidade, ou estória do sistema. Eles verificam não apenas uma classe em particular, mas a iteração entre um conjunto de classes que implementam uma dada funcionalidade". (TELES, 2004).

Os testes são importantes não apenas para certificar ao programador que a rotina que ele escreveu está certa, no XP eles tem um papel muito maior. No XP, os testes de unidades fazem parte da modelagem do sistema, pois tanto à definição das funcionalidades quanto a documentação de um objeto é expressa por seus testes. Este é um dos motivos do XP precisar

de pouca documentação, pois se alguém precisar saber qual a função de um objeto dentro do sistema, basta olhar seus testes de unidade para descobrir o que ele faz.

3.4.4.1 – Testes de unidade (Unit Test)

Numa rotina normal de desenvolvimento da XP, após os programadores receberem a estória a ser implementada, eles passam a escrever os testes unitários para ela. Por isto se chama esta técnica de desenvolvimento guiado pelos testes.

Devem ser escritos primeiro os testes mais simples para o objeto (verificar se ele está criado, por exemplo), e executar o teste. O mesmo irá falhar, pois ainda não foi escrito o código para criação do objeto. Então, este código deve ser escrito, de maneira que passe no teste. Após passar por este teste, deve ser escrito outro, e depois codificado para que se passe neste, e assim sucessivamente, até que não se consiga pensar em mais nenhum teste a ser criado.

A cada novo passo devem sempre ser executados todos os testes, a fim de verificar se a rotina recém inserida não estragou nada que estava funcionando. Assim, depois de encerrada a implementação, o código já vai estar integralmente testado, e vai dar segurança para que o programador possa ir adiante com maior velocidade.

Os testes automatizados são fundamentais para a XP, uma vez que dão suporte a outras técnicas da metodologia, como o refinamento de código e o código coletivo. Sem testes automatizados, é muito difícil e penoso se trabalhar nestes dois princípios, pois não se tem a confiança necessária para saber se o código alterado não vai apresentar problemas, ou alterar algo que já estava funcionando.

Várias linguagens de programação oferecem frameworks de testes automatizados. No Java tem-se o jUnit, no Smalltalk o sUnit (ambos criados por Kent Beck) e no Delphi tem-se o dUnit. Estes são alguns exemplos, mas existem vários outros frameworks disponíveis.

3.4.4.2 – Testes de Aceitação (Acceptance Tests)

Os testes de aceitação são a melhor maneira pela qual o cliente pode fornecer feedback sobre o sistema. São eles que indicam para os desenvolvedores quando a estória que eles estão desenvolvendo está pronta ou não. Os testes devem ser escritos pela equipe do cliente com a ajuda do analista de testes caso haja alguém na equipe com este cargo, fornecendo material

para que os desenvolvedores do sistema possam avaliar o progresso da implementação, e ter certeza de que o que estão desenvolvendo está correto.

Os testes não precisam ser escritos logo no começo, antes do jogo do planejamento, mas devem estar escritos antes que se inicie o desenvolvimento da estória. Eles devem ser preferencialmente automatizados, o que vai gerar ganho de tempo por parte da equipe de desenvolvimento, pois testes manuais tendem a ser mais demorado. O cliente também deve ficar com uma cópia dos testes que ele escreveu, a fim de que ele possa homologar cada versão entregue.

À medida que o cliente vai ficando mais habituado aos testes de aceitação, ele vai passando a escrever testes cada vez melhores e mais completos. Não é necessariamente o mesmo membro da equipe do cliente que define as estórias e os testes que vai ser o responsável por executar estes testes de aceitação. Esta tarefa pode ser deixada a cargo de uma equipe de testes do cliente, pessoas não necessariamente ligadas ao desenvolvimento do projeto e que tem como única incumbência dar retorno para o encarregado da equipe do cliente sobre como os testes estão sendo executados. Podemos exemplificar um teste de aceitação com os passos a seguir:

Cenário Principal

1. Listar todos os produtos, verificar que não existe o produto ‘Mouse123’.
2. Ir para a página de cadastro e cadastrar o produto ‘Mouse123’.
3. Listar todos os produtos, verificar que existe o produto ‘Mouse123’ e não existe o produto ‘Teclado123’.
4. Editar o produto ‘Mouse123’ e mudar seu nome para ‘Teclado123’.
5. Listar todos os produtos, verificar que existe o produto ‘Teclado123’.
6. Apagar produto ‘Teclado123’.
7. Listar todos os produtos, verificar que não existe o produto ‘Teclado123’.

Cenário Alternativo

1. Listar todos os produtos, verificar que não existe o produto ‘Monitor123’.
2. Ir para a página de cadastro preencher o nome do produto com ‘123Monitor’ e submeter.
3. Voltar para a página 2 com a mensagem "Erro: nome do produto não pode começar com numero".
4. Ir para a página de cadastro cadastrar produto ‘Monitor123’.
5. Listar todos os produtos, verificar que existe o produto Monitor123.
6. Apagar produto ‘Monitor123’.

7. Listar todos os produtos, verificar que não existe o produto ‘Monitor123’.

Este é mais um diferencial da XP. Os testes sendo definidos pelo cliente e não pela equipe de desenvolvimento, fazem com que o produto final fique muito mais de acordo com o que o cliente realmente precisa, uma vez que cada tarefa só vai ser dada por concluída depois de passar pelos testes que o próprio cliente definiu.

Colocando a criação de testes como uma tarefa do cliente também o obriga a participar de forma realmente efetiva do desenvolvimento do sistema e cria uma “cumplicidade” maior entre equipe de desenvolvimento e equipe do cliente.

3.4.5 – Integração Continua (Continuous Integration)

Quando uma equipe desenvolve um software é comum que o software seja “quebrado” em algumas partes a fim de cada desenvolvedor implemente a sua parte., esta visão é interessante porque assim cada desenvolvedor tende a se preocupar só com a sua parte do sistema o que reduz a complexidade do problema. Neste caso são definidas interfaces para comunicação entre estas partes para posterior agrupamento e formação de um software coeso. Porém é muito comum acontecerem erros na integração, estes erros acontecem geralmente por que:

- As interfaces definidas não foram respeitadas;
- Não houve compreensão por parte de um desenvolvedor da interface do outro;
- O sistema como um todo é pouco coeso.

Devido a estes erros e também da instabilidade das interfaces, por estarem sempre em constante mudança, a integração continua se faz necessária para amenizar e até suprimir esses erros, uma vez que expõe todo o estágio atual de desenvolvimento, oferece feedback constante e estimula agilidade no desenvolvimento do software.

A técnica de integração contínua diz que o código desenvolvido por cada par de desenvolvedores deve ser integrado ao código base constantemente. A integração ocorre em termos de horas ou dias no máximo. Muitas vezes a integração causa sérios problemas às equipes de desenvolvimento. Então por que integrar em intervalos curtos? Porque geralmente os problemas de integração ocorrem por haver muita diferença entre os códigos desenvolvidos, pois muito trabalho foi feito por cada membro da equipe entre cada integração. Com a integração contínua, aliada aos testes de unidade, estes problemas se reduzem bastante, pois cada vez que algum código é integrado, todos os testes de unidade devem ser executados,

e se algum deles falhar, é porque o código recém integrado foi o responsável por inserir este erro no sistema.

Segundo Jeffries et al. (2001), quando um par de desenvolvedores está trabalhando com XP, o código avança através de três fases:

1. Local – É a primeira fase do desenvolvimento. O par acabou de começar a trabalhar no código e as mudanças que executaram ainda não estão disponíveis para os outros desenvolvedores.
2. Candidato à liberação – O par terminou a sua tarefa e está pronto para disponibilizar o novo código contendo suas alterações para os demais desenvolvedores.
3. Disponibilizado no repositório (ferramenta que armazena o código fonte do sistema e controla a versão dos arquivos) – Esta é a versão oficial corrente do código. Em princípio, este código funciona necessariamente, visto que todos os testes de unidade executam com 100% de sucesso. Neste momento, as alterações do par se tornam disponíveis para todos os desenvolvedores.

Jeffries et al. (2001) recomendam a seguinte abordagem:

1. Sempre comece a trabalhar com o código mais atual que se encontra no repositório. Isto assegura que você esteja sempre trabalhando com a melhor e mais recente versão de todas as partes do sistema.
2. Escreva os testes de unidade que são necessários para a sua tarefa. E lembre-se de sempre escrever os testes de unidade primeiro e permitir que o desenvolvimento seja guiado por eles.
3. Rode todos os testes de unidade.
4. Concerte qualquer teste que esteja quebrado. Considerando que você começou a trabalhar usando o código que veio do repositório, os únicos testes que irão quebrar devem ser aqueles que você acabou de escrever. Portanto, é sua responsabilidade concertá-los.
5. Quando todos os testes de unidade estiverem rodando 100% de sucesso as suas alterações locais se tornaram candidatas à liberação. Isto é, você já pode colocá-las no repositório.
6. O código que é candidato à liberação é integrado com o código atual do repositório. Recomendamos que esta integração seja feita em uma maquina separada, de modo que iremos descrevê-la considerando esta abordagem. Vá até a máquina de integração e carregue todo o código do repositório. Carregue o código

que você editou e é candidato a mudança. Verifique a existência de conflitos. O código que veio do repositório deve ser o mesmo com o qual você começou a trabalhar. Caso não seja, isso significa que outro par já colocou suas mudanças no repositório antes de você. Isso normalmente não ocorre com muita freqüência, mas vamos assumir que tenha ocorrido.

7. Se o código do repositório tiver sido alterado enquanto você fazia as suas mudanças, compare as diferenças entre as suas mudanças e o código do repositório. Use uma ferramenta para fazer isso e não apenas os próprios olhos. Identifique as mudanças que tiverem sido adicionadas pelo outro par e integre estas mudanças com as suas. Peça ajuda ao outro par se for necessário. Lembre-se que a comunicação é um valor fundamental do XP que deve ser utilizado em todos os momentos do projeto. Conversar com os colegas da equipe pode fazê-lo avançar muito depressa em diversas situações de dúvida.
8. Uma vez que você tenha integrado as suas mudanças, execute os testes de unidade na máquina de integração. Eles devem rodar com 100% de sucesso e normalmente terão este comportamento. Se isso não ocorrer, significa normalmente que você deixou de integrar alguma coisa ou que as suas mudanças conflitam com as mudanças de outra pessoa. Os testes de unidade que tiverem falhado irão indicar o problema. Corrija o problema e lembre-se de pedir ajuda de outros pares caso você necessite.
9. Quando os testes de unidade estiverem executando com 100% de sucesso, coloque todo o seu código no repositório, isto é, faca com que o código da máquina de integração se torne o oficial.

Existe um termo usado com freqüência para descrever o avanço cauteloso da equipe quando integra com freqüência executando todos os testes a cada iteração. Costuma-se chamar esta prática de baby steps, ou passos de bebê. A equipe avança um passo de cada vez, sempre assegurando que o seu avanço não será comprometido por erros que estejam latentes no sistema.

3.4.6 – Projeto Simples (Simple Design)

A técnica de manter o projeto simples (Simple Design) é uma grande quebra de paradigma para quem está acostumado a trabalhar em desenvolvimento de sistemas e é muitas vezes difícil de se acostumar a projetar apenas "a coisa mais simples que possa funcionar".

Como dito anteriormente uma das premissas técnicas do XP é de que o custo de uma alteração cresce lentamente ao longo do projeto, por isso você pode deixar as grandes decisões para o futuro, ou seja, deixe os problemas do futuro para o futuro. A equipe deve implementar apenas os problemas de hoje, e fazendo isso da maneira mais simples possível.

Fazendo isso evitamos o trabalho especulativo que ocorre quando não se tem certeza sobre uma determinada funcionalidade e mesmo assim ela é implementada, geralmente esta é implementada de forma errônea, fazendo com que se tenha um alto grau de retrabalho ou over-engineering causando desperdício de tempo e dinheiro do cliente.

De acordo com Vinicius Teles a prática de trabalhar com design simples respeita os valores do XP:

- Comunicação – Um design simples comunica a sua intenção de forma mais eficaz que um design complicado. Isto ocorre porque é mais fácil compreendê-lo. Naturalmente, o design precisa ser simples, mas também deve ser capaz de comunicar todos os aspectos importantes do software.
- Simplicidade – Um design simples torna o software mais leve e mais fácil de ser alterado a qualquer momento, visto que é mais fácil de ser compreendido.
- Feedback – Criando um design simples, a equipe é capaz de avançar de forma mais ágil e obter feedback do cliente mais rapidamente.
- Coragem – A equipe trabalha o design apenas até o ponto em que ele resolve o problema de hoje. Problemas do futuro, mesmo que já sejam previstos, são deixados para o futuro. É necessário coragem para assumir que a equipe é capaz de resolver o problema no futuro caso ele realmente se manifeste. (TELES, 2004).

Kent Beck (2000, p.114) utiliza quatro regras básicas para definir simplicidade, são elas em ordem de prioridade:

1. O sistema (código e testes em conjunto) deve expressar tudo àquilo que você deseja comunicar.
2. O sistema não deve conter nenhum código duplicado.

3. O sistema deve ter o menor número de classes possível.
4. O sistema deve ter o menor número de métodos possível.

"O propósito do design de um sistema é, em primeiro lugar, comunicar a intenção dos programadores e, em segundo lugar, prover um lugar para acomodar a lógica do software" (TELES, 2004).

Seguidamente leva-se mais tempo tentando tornar uma rotina flexível, reaproveitável, do que implementando a mesma. Um exemplo foi dado por Ron Jeffries, um dos criadores da XP, onde ele menciona que até mesmo eles, os autores da XP, às vezes se esquecem deste princípio.

O caso descrito por ele ocorreu enquanto escrevia o livro Extreme Programming Installed. Jeffries tinha cada capítulo do livro em um arquivo independente. Ele sentiu a necessidade de desenvolver um sistema simples, onde ele tivesse acesso aos arquivos da pasta onde estavam os capítulos, e tivesse controle sobre quais já haviam sido mandados para os revisores e quais ainda faltavam ir. Então, ele resolveu criar um programa na linguagem Doplhin. Quando ia começar, ele achou que o objeto que ele iria criar ficaria bem melhor se ele pudesse definir uma pasta qualquer, para poder fazer este controle para qualquer novo livro que ele viesse a escrever. Então ele perdeu dois dias tentando desenvolver esta ferramenta genérica, até que se deu conta que não era isto que ele precisava para o momento, mas sim indexar os capítulos deste livro. Então, em pouco mais de uma hora ele desenvolveu este aplicativo, que resolveu o problema do momento.

Esta então é a aposta da XP, que não deve ser desenvolvido nada que não seja necessário para a iteração na qual se está trabalhando agora, "resistindo à tentação" de fazer uma rotina "mais completa".

Este princípio é conhecido na XP pela sigla YANGNI (you are not gonna need it, em português, você não precisará disto), que diz que normalmente se acaba não precisando de todas as funcionalidades e flexibilidade que se coloca em um código que poderia ser implementado de forma simples.

3.4.7 – Refatoração (Refactoring)

Refatoração (Refactoring) é o processo de mudar um sistema de software de tal forma que não se altera o comportamento externo do código, mas melhora sua estrutura interna. É

um meio disciplinado de limpar o código e que reduz as possibilidades de introduzir erros. Em essência quando você refina o código, você melhora o projeto depois que este foi escrito.

"Melhorar o projeto depois que foi escrito." Esta é uma premissa do XP. Em nosso entendimento atual de desenvolvimento de software nós acreditamos que projetamos e então codificamos. Um bom projeto vem primeiramente, e a codificação vem posteriormente. A todo o tempo o código será modificado, e a integridade do sistema, sua estrutura de acordo com esse projeto, gradualmente diminui.

Refinamento do projeto é o contrário desta prática. Com o refinamento pode-se pegar um projeto ruim, mesmo que esteja um caos, e retrabalhar ele para que se torne um código bem-projetado e robusto. Cada passo é simples, pode-se mover um campo de uma classe para outra, retirar algum código para fora de um método, deslocar algum código para cima ou para baixo em uma hierarquia ou até mesmo renomear métodos e classes. Mas o efeito cumulativo destas pequenas mudanças pode radicalmente melhorar o projeto. É o contrário exato da noção normal de deterioração de software.

A técnica de refinamento do design é utilizada sempre com o intuito de simplificar o código. Refactoring significa reescrever o código, melhorando e simplificando o mesmo, sempre que se tiver oportunidade. O projeto do código vai sendo aos poucos melhorado através de modificações que o aprimorem. Estas modificações partem de um código fonte o qual passa em todos os testes, e devem levar a um código-fonte que continue passando em todos os testes.

Na XP, o refactoring é visto como uma tarefa de engenharia, pois trata da melhoria contínua do código-fonte, e conseqüentemente da documentação (uma vez que na XP o código deve falar por si, sendo a mais importante documentação, juntamente com os testes).

Deve-se aplicar esta técnica sempre que for incluída uma nova funcionalidade e for necessário “preparar” a classe para receber a mesma, quando for realizada manutenção em um código já existente, logo após implementar uma nova funcionalidade, ou a qualquer momento que se perceber que o código não está na sua forma mais simples.

Indícios de que um código precisa de refinamento são: código duplicado (a mesma rotina em dois pontos distintos do código), métodos de classes muito longos, métodos cujos nomes não indicam claramente o que fazem e excesso de comentário no código, pois se existem muitos comentários é sinal de que o código não está suficientemente claro ou os nomes dos métodos não expressam muito bem o que eles fazem. Esta técnica é tão importante dentro da XP que existe um livro que trata exclusivamente de refinamento. (FOWLER, 1999).

O refinamento de código é uma prática que “casa” com várias outras, uma delas é o código coletivo que permite que desenvolvedores atuem em qualquer parte do sistema. Com isso além de poderem refinar qualquer parte do sistema, torna-se muito mais fácil à identificação da necessidade de se fazer o refinamento, porque como várias pessoas lêem o mesmo código fica fácil identificar partes mal formuladas e aplicar as correções necessárias.

Uma importante visão a respeito do refinamento de código é que o refactoring tem que ser feito a cada momento do desenvolvimento do sistema, e não em uma etapa posterior refatorando tudo, pois caso fosse assim gastaríamos muito tempo e dinheiro para refatorar e muitas das vezes não lembraremos o que diz respeito determinado código, potencializando assim uma possível falha no refinamento.

Para que se possa fazer um refinamento com segurança, é fundamental que se tenha implementado a técnica de testes automatizados, pois antes de se iniciar o refinamento a classe deve passar por todos os testes, e após a alteração deve continuar passando e retornando os mesmos resultados.

Refinamento, Design Simples e Testes automatizados são técnicas que se complementam. Com o refinamento se consegue um design simples e os testes automatizados dão segurança ao processo. Por outro lado, fica muito mais fácil prever testes para tudo que possa falhar se as rotinas estiverem na forma mais simples que possa funcionar, o que é conseguido pelo refinamento.

Neste ponto pode-se perceber porque se diz que cada técnica da XP individualmente traz um bom retorno para quem as aplica, mas se utilizadas em conjunto o retorno proporcionado é muito maior que a soma do retorno de cada uma.

3.4.8 – Programação em pares (Pair Programming)

Todo o código que vai para a produção é escrito por um par de programadores que utilizam uma mesma estação de trabalho ao mesmo tempo, ou seja, um computador, um teclado e dois desenvolvedores.

Na XP todo o código deve ser produzido por duas pessoas utilizando o mesmo computador. Enquanto um dos parceiros se preocupa com detalhes da implementação, ficando responsável pela digitação do código, o outro deve tentar ter uma visão mais ampla da rotina, imaginando as suas peculiaridades. Não apenas o código deve ser produzido por duas pessoas, como também todo o projeto da classe na qual vai se trabalhar.

Trabalhar em pares pode parecer antiprodutivo à primeira vista, pois ao invés de se ter duas pessoas produzindo código independentemente, têm-se apenas um código sendo implementado por vez, ocupando horas/homem de duas pessoas. Mas se realmente a velocidade da produção pode ser menor, os ganhos gerados pela programação em duplas compensam este esforço adicional:

Duas pessoas conseguem pensar em um número maior de testes automatizados que possam servir para garantir o funcionamento da classe, tendo assim uma maior diversidade de idéias, técnicas e algoritmos;

A troca de experiências entre os membros da dupla faz com que muitos problemas possam ser resolvidos rapidamente durante o desenvolvimento;

Ocorre o nivelamento da equipe de desenvolvimento, pois a tendência é que as pessoas vão adquirindo novos conhecimentos e melhorando seu nível como programador à medida que trabalham com pessoas mais experientes;

Garante que todos os membros da equipe já tenham trabalhado pelo menos alguma vez em todos módulos dentro do sistema, nem que tenha sido como parceiro de outro programador que estivesse no momento controlando, disseminando o conhecimento do sistema por toda equipe e acabando com "feudos" dentro do sistema (partes dele que apenas um determinado desenvolvedor conheça);

Com duas pessoas projetando e a revisão de código sendo feita instantaneamente, pelo parceiro que não está digitando, o trabalho final sai com maior qualidade.

Assim, o trabalho em pares se justifica na qualidade do código gerado, uma vez que muitas vezes se passa muito mais tempo procurando um erro em uma rotina do que se implementando a mesma. Com o código saindo correto da primeira vez, se elimina muito do trabalho de refactoring posterior, pois um dos parceiros pode sempre enxergar uma maneira mais simples de se escrever a rotina que está sendo desenvolvida.

Além disto, pesquisas demonstram que duplas produzem código de melhor qualidade em aproximadamente o mesmo tempo que programadores trabalhando sozinho, outro fato interessante apontado pelas pesquisas é que 90% dos desenvolvedores que aprendem programação em duplas a preferem.

Ao programar em duplas produzimos um efeito colateral que chamamos de "pressão do par", este efeito corrige diversos problemas, tais como, a distração, a improvisação dentre outros. Isto funciona porque quando estamos trabalhando em duplas o programador deixa de ter um compromisso apenas com si mesmo. Distrações acontecem e quando acontecerem o seu par estará ali para te ajudar a retomar o foco do desenvolvimento, por outro lado

trabalhando em dupla você poderá ficar com “vergonha” ao ter que improvisar alguma coisa no código, algo como uma gambiarra.

Deve-se ainda observar que as duplas devem ser trocadas periodicamente, para que realmente a programação em pares cumpra seu papel de disseminar o conhecimento entre a equipe.

3.4.9 – Propriedade Coletiva (Collective Ownership)

O código deve ser de propriedade de todos e todos devem ter permissão para alterar o que for necessário para que seu trabalho possa ser desenvolvido. Em estruturas onde determinadas rotinas são de “propriedade” de algumas pessoas, podem ocorrer atrasos no desenvolvimento devido à necessidade de que seja alterado algo nestas rotinas para que outras pessoas possam continuar com o seu trabalho.

Esta visão de “propriedade” de código é muito perigosa, pois cria “ilhas de conhecimento” dentro de um projeto. Uma “ilha de conhecimento” pode ser compreendida como sendo uma pessoa que possui um domínio sobre uma área do projeto.

Caso ocorra alguma coisa com esta pessoa, por exemplo, um afastamento por doença ou até mesmo desligamento do projeto, esta não poderá ajudar a equipe na parte do código que é de sua responsabilidade. Se alguém precisar mexer nos seus códigos, simplesmente não será capaz de fazê-lo ou terá grandes dificuldades, isto porque o código pode estar absolutamente claro para a pessoa que o desenvolveu, entretanto pode ser completamente incompreensível para outros membros da equipe.

Utilizando a prática do código coletivo todos são donos, logo responsáveis por todo o código. Isto é muito diferente de se dizer que o código não é de ninguém, pois quem fizer uma alteração é responsável por ela.

“Sendo o código coletivo, cada parte dele pode eventualmente sofrer a visita de diferentes desenvolvedores ao longo de sua vida. A conseqüência direta é que o código estará sempre sendo revisado.” (TELES, 2004).

Para garantir o mínimo de problemas com esta técnica, existem os testes automatizados e a integração continua, que garante que as alterações feitas vão ser anexadas rapidamente ao código principal, diminuindo muito a chance de duas duplas estarem trabalhando no mesmo código.

3.4.10 – Padrões de Codificação (Coding Standards)

Como vimos anteriormente à comunicação é um valor fundamental do XP, esta comunicação tem que ser não apenas verbal, mas também através do código produzido. Para que um código fale por si só ele deve ser simples, legível e principalmente deve seguir um padrão.

O XP recomenda a utilização de padrões de codificação, estes padrões devem ser adotados no inicio do projeto com o consentimento de todos os desenvolvedores. Desta forma, qualquer desenvolvedor poderá ler o código com velocidade, sem se preocupar em compreender estilos de formatação especiais.

Também é importante que os desenvolvedores sigam o padrão de desenvolvimento para que todos possam ter acesso a qualquer parte do código e que as pessoas possam trabalhar em pares. O ideal é que, com o passar do tempo, alguém que olhe o código não saiba dizer por quem o mesmo foi escrito. Assim fica mais fácil conseguir um código simples, e muito mais ágil o processo no caso da necessidade de um refinamento.

Jeffries et al.(2001) apresenta alguns tópicos que devemos considerar em um padrão de codificação:

- Indentação – Procure usar um estilo de indentação consistente:
    - Mesma largura na tabulação.
    - Mesmo posicionamento de chaves.

  Sempre faça a indentação de um código que esteja subordinado a outro.
- Letras maiúsculas e minúsculas – Seja consistente com o tamanho da letra. Para isso, procure seguir o padrão comumente utilizado entre os programadores de sua linguagem. Evite usar padrões de outra linguagem, mesmo que você esteja mais habituado com eles.
- Comentários – Procure evitá-los tanto quanto possível. Manter o foco na construção de um código que seja claro, simples e que transmita a sua intenção é muito mais valioso que qualquer comentário que você possa adicionar ao seu código. Escreva um comentário apenas quando você precisar dele, por exemplo, para explicar uma otimização ou algo que você simplesmente não consegue simplificar. Não escreva um comentário quando ele for desnecessário. Por fim, lembre-se que o código evolui e os comentários devem evoluir com ele.

- Nomes – Use nomes que comuniquem e não nomes que sejam convenientes para digitar. A idéia principal é que o código deve ser capaz de comunicar com você e com os seus colegas hoje e no futuro. O impacto de se ter nomes consistentes é pequeno no início do projeto, contudo, à medida que ele cresce e a equipe aprende a avançar mais rapidamente, o uso de nomes consistentes ajuda a manter o desenvolvimento na velocidade máxima e de torna um investimento valioso.

Utilizando-se um padrão de codificação, padrão este definido pelos desenvolvedores, o código se torna familiar como se tivesse sido escrito por um único individuo, competente e organizado. Já um código com estrutura familiar facilita e estimula muito a posse coletiva, a comunicação mais eficiente, a simplicidade, a programação em pares e o refinamento do design.

"Lembre-se, o formato do padrão não é muito relevante para o projeto. O que realmente importa é que todos os membros da equipe adotem o padrão e o utilizem sempre." (TELES, 2004).

3.4.11 – Ritmo Sustentável (40 Hour Week)

Um dos grandes problemas que projetos de desenvolvimento de software enfrentam é o curto prazo de entrega do sistema. Com um prazo cada vez mais curto e sistemas cada vez mais complexos uma das alternativas dos desenvolvedores é o trabalho em excesso. As pessoas passam a trabalhar diariamente até mais tarde invadindo também finais de semana e feriados, porém trabalhar por longos períodos é contraproducente, e normalmente tende a piorar a situação.

"Desenvolver software é um trabalho que exige muita concentração e criatividade, características improváveis de serem encontradas em pessoas exaustas." (TELES, 2004).

Um programador cansado coloca mais erros no código e o tempo ganho em uma noite de hora extra muitas vezes é perdido no dia seguinte se procurando por um bug dentro de uma rotina. O número 40 é apenas uma sugestão, cada pessoa tem sua própria capacidade de trabalho, este tempo poderia ser de 35 ou 45 horas, por exemplo.

Portanto, trabalhar além do horário pode até acelerar o desenvolvimento nos primeiros dias, mas tão logo os desenvolvedores se cansem erros aparecerão.

Segundo Teles, (2004, p.172) existem duas razões fundamentais para a recomendação de 40 horas de trabalho por semana, a primeira diz que permitir que os desenvolvedores

trabalhem apenas oito horas por dia é mais humano e demonstra respeito à individualidade e a integridade de cada membro da equipe, em segundo lugar, esta pratica aumenta a agilidade do projeto.

"Ao redor do mundo e aqui no Brasil, diversas equipes têm conseguido adotar esta prática e hoje colhem os frutos dela. Seus projetos são mais ágeis, as pessoas convivem de forma mais harmoniosa e, sobretudo, não precisam sacrificar a vida pessoal em nome do trabalho." (TELES, 2004).

3.4.12 – Metáfora (Metaphor)

Equipes XP mantêm uma visão compartilhada do funcionamento do sistema. Isto é feito através do uso de metáforas. Fazendo uso de metáforas podemos facilitar a explicação de qualquer aspecto dentro do projeto e ao mesmo tempo fixa-la com mais força na mente do ouvinte.

Utilizam-se metáforas para vários aspectos na XP, dentre eles procura-se criar uma visão comum sobre todo o projeto de forma simples e objetiva facilitando assim a comunicação entre toda a equipe, uma vez que serve de base para os padrões de codificação e entre a equipe e o cliente, visto que o cliente, na maioria das vezes, não entende os termos técnicos relacionados ao desenvolvimento e usados diversas vezes pelos desenvolvedores e equipe.

Usando-se de metáforas para exemplificar, por exemplo, comportamentos de classes e objetos têm-se facilitada e muito a disseminação do conhecimento entre a equipe.

Para a criação de metáforas é necessária muita criatividade, e isto não se pode alcançar se você estiver sobrecarregado, você deve ter a mente tranqüila para poder pensar em algo realmente útil. Por isso o uso de metáforas anda muito relacionado com outra prática importante da XP, o ritmo sustentável. Não se pode criar funcionários extremamente voltados para o trabalho, porque com isso rapidamente você terá membros da equipe altamente estressados, ficando com pouca criatividade para gerar metáforas e baixa produtividade, criando assim códigos de má qualidade.

A metáfora é utilizada para guiar o desenvolvimento, ajudando a todos os envolvidos no projeto a entenderem os elementos básicos e seus relacionamentos.

3.5 – Métricas e Gerenciamento

Toda metodologia de desenvolvimento deve fornecer ao seu gerente condições para avaliar o andamento do projeto, através de medições de qualidade (tanto do produto quanto do processo de desenvolvimento) e prazos. Estas medições são conhecidas no desenvolvimento de sistemas como métricas. As métricas mais importantes na XP são:

- **Velocity** – Número de funcionalidades entregues comparado ao número de funcionalidades prometidas. O treinador pode, durante o desenvolvimento de uma iteração, ir verificando o quão próximo do seu encerramento ela está, baseando-se na velocity de cada estória. A XP prega que se deve sempre trabalhar baseado na quantidade de dias gastos até o momento e na quantidade de dias que o desenvolvedor acha que ainda vai precisar, e não com percentual de compleição da tarefa. Exemplificando, se tivermos no primeiro release 15 estórias prometidas e entregarmos 12 a nossa velocity será 0,8.
- **40 Hour Week** – Quantidade de horas trabalhadas por semana por desenvolvedor. Esta métrica é importante para medir a qualidade interna do trabalho, pois como prega a XP trabalhar longos períodos por muito tempo é contraproducente.
- **Test Coverage of the Code** – Quantidade de Unit Tests por classe, também é uma métrica de qualidade interna de trabalho, pois serve para medir o quão testado está um sistema.
- **Acceptance Test Coverage** – Quantidade de testes de aceitação do projeto como um todo.
- **Acceptance Test Pass** – Percentual de testes de aceitação aprovados para cada iteração enviada, em relação ao total de testes de aceitação da mesma.
- **Relative Bug Density** – Quantidade de erros encontrados pelo usuário para cada classe multiplicado pelo tempo que cada um levou para ser corrigido. Para fazer o cálculo, atribui-se pontos para cada unidade de tempo gasta. Por exemplo, 1 ponto se for corrigido em até meia hora, dois pontos se levar até 2 horas e quatro pontos por turno adicional de serviço.
- **Load Factor** – Quantidade de horas trabalhadas por mês comparadas com as horas trabalhadas em desenvolvimento por mês.

  EX: desenvolvedor Daniel, agosto de 2004.

Horas gastas em atendimento ao cliente: 20

Horas gastas em treinamento: 20

Horas gastas em cafezinho: 5

Horas gastas em desenvolvimento: 165

Horas totais trabalhadas no mês: 220

Load factor = 220 / 165 = 1,333

Ou seja, cada dia ideal de programação deste desenvolvedor corresponde a 1,33 dias de calendário (ou dias reais...) com a velocity média do projeto e o load factor médio da equipe, você pode ajustar as estimativas.

Para isto utiliza-se: (estimativa / velocity) * load factor

No exemplo, uma estória estimada em 3 dias ideais de programação receberia uma "estimativa ajustada" de 4,99 dias horas trabalhadas no mês.

Baseado nestas métricas, o treinador pode ajustar a qualquer momento o projeto, tanto na questão de prazos e estimativas quanto no quesito de qualidade, sugerindo um maior número de testes de unidade, por exemplo, se verificar que os indicadores Acceptance Test Pass e Relative Bug Density começarem a demonstrar muitos problemas de qualidade.

Estas medidas são possíveis graças ao caráter dinâmico da XP, onde o cliente pode fornecer feedback sobre o desenvolvimento durante o mesmo, e não apenas após ter o produto final em suas mãos. Por este aspecto, e também por suas características de ser fortemente calcado em testes, tanto unitários quanto de aceitação, que a gerência de projetos em XP difere tanto da gerência tradicional.

Dentro do gerenciamento da XP utilizamos muito o rastreamento (Tracking) que é o acompanhamento, a medida das métricas dentro da metodologia XP. O rastreador deve, uma ou duas vezes por semana, coletar os dados relativos às métricas escolhidas para acompanhamento pelo treinador e elaborar os gráficos de medida correspondentes às mesmas. Este trabalho deve ser o menos invasivo possível, a fim de não burocratizar o processo de desenvolvimento e de não gerar perda de produtividade por parte da equipe de desenvolvedores. Por este motivo, qualquer ferramenta que auxilie o rastreador na sua tarefa deve ser integrada ao dia-a-dia da equipe.

Existem técnicas de gerência de projeto para as mais diversas metodologias de desenvolvimento. Para a maioria delas, as técnicas são comuns, uma vez que estas possuem fases bem definidas dentro do projeto.

A XP, pelas suas características, precisa de uma gerência diferente de projetos, onde o seu caráter de planejamento contínuo possa ser implementado, levando-se muito em conta o

fator de ser uma metodologia que prega a informalidade do processo. Qualquer metodologia de gerência de projetos que se queira aplicar à XP deve ser centrada nas estórias do usuário, nos testes de aceitação e no rastreamento das métricas da XP.

Por estes motivos as técnicas tradicionais de gerência de projetos, e conseqüentemente as ferramentas desenvolvidas para elas, como é o caso do MS-Project, da Microsoft, não são adequadas para gerência de projetos XP, uma vez que não prevêem a natureza dinâmica e as características de gerenciamento da metodologia.

4. SITUAÇÃO ATUAL DA UTILIZAÇÃO DA XP NAS EMPRESAS.

Para a realização do estudo de como esta a utilização da metodologia XP nas empresas atualmente usaremos como base o estudo de caso.

O estudo de caso é um dos vários modos de realizar uma pesquisa sólida. Outros modos incluem experiências vividas, histórias, e a análise de informação de arquivo (como em estudos econômicos).

Cada estratégia tem vantagens e desvantagens que dependem de três condições: 1) o tipo de foco da pesquisa; 2) o controle que o investigador tem sobre eventos comportamentais atuais, e 3) o enfoque no contemporâneo ao invés de fenômenos históricos.

Em geral, estudos de casos se constituem na estratégia preferida quando o "como" e/ou o "por que" são as perguntas centrais, tendo o investigador um pequeno controle sobre os eventos, e quando o enfoque está em um fenômeno contemporâneo dentro de algum contexto de vida real.

Estudos de casos podem ser classificados de várias maneiras, explicativos, cognitivos, expositivos. Porém o que iremos tratar neste trabalho é "estudo de caso explicativo".
Este estudo se dará em cima de cinco questões básicas que são elas:

- O que levou ao uso da eXtreme Programming?
- Quais os resultados obtidos?
- Pretende continuar utilizando esta metodologia?
- Qual a porcentagem de utilização da metodologia nos projetos da empresa?
- Quais as maiores dificuldades enfrentadas na implantação?

4.1 - APOENA Software Livre

APOENA é uma palavra do vocabulário tupi-guarani e significa "aquele que enxerga longe".

Com esta determinação e com o objetivo de tornar-se excelência em desenvolvimento de software, surgiu a APOENA Software Livre.

A APOENA Software Livre é uma empresa de consultoria e desenvolvimento de software nas modalidades GPL e open-source.

Nascida em novembro de 2000, a empresa é incubada na IETEC – Incubadora Empresarial Tecnológica de Porto Alegre, da qual recebe apoio logístico e mercadológico na prospecção de negócios.

Com sua experiência no desenvolvimento de soluções baseadas em GNU/Linux, a APOENA vem implantando softwares nas áreas de gestão pública e gestão empresarial.

Quem respondeu as perguntas feitas a APOENA Software Livre foi Guilherme Silva de Lacerda que é Diretor de Tecnologia da APOENA.

O que levou ao uso da eXtreme Programming?

Bom, na realidade sou um entusiasta do XP e dos métodos ágeis já faz algum tempo, desde 2001. Mantenho um ótimo contato desde esta época com o Klaus (Klaus é pioneiro na utilização de XP no Brasil e atualmente é diretor de desenvolvimento da Objective Solutions).

Muitos das práticas XP eu já tinha usado em um projeto, quando trabalhava na Urcamp em Bagé. No final 1996, precisei desenvolver um software de Avaliação Institucional para a Urcamp. Inicialmente, quem desenvolveria era o NPD da Instituição. Na época, eu era estagiário e me aventurei (coragem) a desenvolver este software. Convidei um colega de aula (Fabrízio Mello, que faz parte da nossa lista também). Tínhamos apenas uma máquina (486 DX2 66), que nos "forçou" a desenvolvermos juntos (pair programming). Primeiro definimos um estilo de codificação (padrão de codificação). Como trabalhávamos direto, um pouco eu pilotava, outra hora ele, alterávamos o código sem "ressentimento" (posse coletiva).

Além disso, o código sofria uma revisão quase que "doentia".

Na época, nem sabíamos o que estávamos fazendo.... mas a situação nos fez desenvolver assim. Desenvolvemos em dois meses... Pelo que sei, funcionou até pouco tempo, nos sete campus da universidade. Detalhe: usamos na época uma linguagem procedural: Clipper.

Depois quando tomei conhecimento do XP, vi que realmente funcionava. Daí direcionei inclusive minha dissertação de mestrado p/ esta área (inicialmente eu iria fazer em IA).

Aqui na empresa (APOENA), como sou o responsável pela definição de tecnologia e processos, venho adotado práticas XP desde 2001. Como um dos produtos entregues ao cliente é também a documentação técnica, acabamos criando o PADS (Processo APOENA de

Desenvolvimento de Software), baseado no UP, PMR de Craig Larman, práticas XP e outros artefatos que julgamos importantes. Ele foi apresentado em 2002 no eXtreme Programming Brasil 2002 (1º evento no Brasil do gênero).

Quais os resultados obtidos?

Muitos. Notamos que o XP não afetou somente a forma de desenvolvimento, mas até como negociávamos nossos contratos. Criamos uma relação de fidelidade com o cliente e criamos um conceito dentro de nossa equipe: Base de conhecimento comum (Todos conhecem tudo).

Pretende continuar utilizando esta metodologia?

Com certeza. Aprendemos muito com ela e, principalmente, com os clientes e colaboradores.

Qual o percentual de utilização da metodologia nos projetos da empresa?

Atualmente, quase que 100%. Tem algumas práticas que, dependendo dos projetos, fica complicado aplicar.

O XP possui, em geral, o que julgo dois principais ensinamentos:

1) Ensine e aplique XP "aprendendo". Cada projeto é um aprendizado e não tem "receita de bolo". Devemos realizar adaptações. Agora vem o segundo...

2) Conserte o processo quando ele quebrar... Se eu não consigo ter o cliente no local (por exemplo), procuro fazer pequenos lançamentos com janelas mais curtas de tempo, iniciando inclusive com o protótipo.

Quais as maiores dificuldades enfrentadas na implantação?

Seres humanos são resistentes a mudanças. O que acontece é que os clientes já estão acostumados com o formato que as empresas de TI trabalham. Aí você chega com uma novidade dessas, eles ficam bem desconfiados. Geralmente ganhamos o cliente pelo nível e rapidez da resposta.

4.2 – Improve It

A Improve It é uma empresa de consultoria em tecnologia, especializada em desenvolver softwares estratégicos que possibilitem a seus clientes a conquista de vantagens competitivas através da organização de informações essenciais, redução de custos e aceleração das operações.

Focada no desenvolvimento orientado a objetos utilizando Java e também pioneira na adoção do Extreme Programming (XP) - um processo de desenvolvimento de software altamente produtivo que garante qualidade superior e excelente retorno de investimento - a Improve It conta com a participação de profissionais amplamente capacitados e experientes.

Dentre os clientes, encontram-se empresas como a Companhia Vale do Rio Doce, Telemar, Fininvest, Procuradoria Geral do Município, Andima, Tema Sistemas entre outros.

A Improve It acredita que cada dia de trabalho de sua equipe deve gerar o máximo de valor para seus clientes. Por esta razão, seus treinamentos são organizados de modo a facilitar e acelerar o aprendizado, bem como assegurar que seus alunos sejam capazes de colocar os novos conhecimentos em prática de forma segura, simples e rápida.

Quem respondeu as perguntas feitas a Improve It foi Vinicius Manhães Teles que é sócio-diretor da empresa.

O que levou ao uso da eXtreme Programming?

Insatisfação com os resultados de projetos anteriores realizados pelos membros da equipe da empresa.

Quais os resultados obtidos?

Conseguimos realizar projetos de porte considerável no qual o cliente demonstrou muita satisfação com o resultado final e manteve excelente relacionamento com a equipe de desenvolvimento ao longo de todo o projeto. Além disso, conseguimos reduzir os riscos de prejuízos tanto para a Improve It, quanto para nossos clientes. Finalmente, alavancamos um novo mercado para a empresa, de modo que atualmente a maior parte dos negócios giram em torno de consultoria e treinamento em Extreme Programming.

Pretende continuar utilizando esta metodologia?

Sim.

Qual a porcentagem de utilização da metodologia nos projetos da empresa?

100%.

Quais as maiores dificuldades enfrentadas na implantação?

Adoção de testes automatizados e venda de contratos de escopo variável.

4.3 – Naphta Informática

A Naphta Informática é uma empresa que atua exclusivamente com desenvolvimento de sistemas. A empresa foi fundada em 1995, e é sediada na cidade de Novo Hamburgo - RS.

Nossa missão é dar um atendimento de qualidade aos nossos clientes, a fim de manter a sustentação e o crescimento contínuo da empresa, uma vez que nosso marketing é feito exclusivamente pelos nossos clientes.

Como trabalhamos com desenvolvimento de sistemas não possuímos um produto pronto, e não temos uma área específica de atuação, embora mais de 90% de nossos clientes serem do segmento industrial, principalmente na área química e de componentes para calçados.

Quem respondeu as perguntas feitas a Naphta Informática foi Daniel Pohren que é Analista de Sistemas da Naphta.

O que levou ao uso da eXtreme Programming?

Comecei a estudar a XP na faculdade, e fiz meu trabalho de conclusão sobre XP. Estudando a metodologia, vi que ela tem muita coisa boa, e que é a única que eu conseguiria encaixar em uma empresa do tamanho da minha, somos cinco pessoas. Hoje sou um entusiasta da XP, pois ela trouxe bons resultados para nós, mas o início foi mesmo como uma "experiência".

Quais os resultados obtidos?

Principal: melhoria na qualidade do código desenvolvido. Tivemos um baque de queda de produtividade grande no início, mas agora estamos tendo retorno to tempo investido no aprendizado da metodologia.

Pretende continuar utilizando esta metodologia?

Sim, com certeza.

Qual a porcentagem de utilização da metodologia nos projetos da empresa?

Em todos os projetos utilizamos alguma coisa de XP, mas não a usamos na íntegra.

Quais as maiores dificuldades enfrentadas na implantação?

1 - Envolver o cliente no processo... A mais difícil de todas

2 - Quebrar o paradigma dos testes automatizados e do "test first design"... Ainda estamos patinando um pouco neste ponto.

4.4 – Estudos realizados através de pesquisa

Foi realizada uma pesquisa, via internet, de empresas que utilizam a XP como metodologia de desenvolvimento de software. Este estudo visa exemplificar alguns casos de sucesso de empresas na utilização da XP em seus projetos. As empresas dessa relação

utilizam algum grau de XP em seus projetos. Algumas muito pouco, somente declarando seu interessem em XP; outras usam 100%. Infelizmente, não é possível avaliar o quanto cada empresa utiliza:

- Apoena Software Livre de Porto Alegre, RS.
- Brasil Telecom de Brasília, DF.
- Embrapa Informática Agropecuária de Campinas, SP.
- Improve It do Rio de Janeiro, RJ.
- Objective Solutions de São Paulo, SP e Curitiba, PR.
- Trevisan Tecnologia de Porto Alegre, RS.
- XIMP de Uberlandia, MG.
- Oktiva de Fortaleza, Ceará.
- Qualiti do Recife, PE.

4.4.1 – Secretaria da Fazenda do Estado de São Paulo

A equipe de desenvolvimento da Secretaria da Fazenda do Estado de São Paulo está adotando XP. Porem estão colocando uma prática de cada vez, à medida que cada prática se consolida (ou seja, quando ninguém mais precisa ser "lembrado" de usá-la) passa-se a implementar a próxima prática.

Algumas de suas experiências estão espalhadas pelo site www.xispe.com.br nos tópicos que dizem respeito, mas sua documentação completa ainda está em fase de desenvolvimento.

As primeiras práticas a serem utilizadas foram:

- O jogo do Planejamento, Planejamento do Releases e Planejamento das Iterações (A parte de planejamento está funcionando perfeitamente);

- Cliente Presente;

- Integração Continua;

- Refatoração e Desenvolvimento guiado por testes;

- Programação em pares;

Vimos que em alguns pontos a metodologia foi adaptada, mas continua com seus valores guiando o desenvolvimento e tudo está funcionando muito bem.

Uma das maiores dificuldades na implantação da XP foi com a parte de Testes (não para fazer, mas para convencer o pessoal) mas agora está caminhando bem. Quase todas as novas funcionalidades estão sendo feitas via Desenvolvimento guiado por testes.

A próxima prática a ser atacada depois que conseguirmos consolidar a prática dos Testes é trabalhar com Metáforas.

Com esta visão podemos ver que a adoção da XP é mais facilmente feita quando implantada em partes, devido a complexidade de algumas partes, visto que a XP trabalho com a mudança de paradigma.

4.4.2 – TIM

A TIM (Telecom Italian Mobile) teve a XP aplicada a um de seus projetos. Este projeto visava à integração entre celulares e sistemas, tendo como uma de suas funcionalidades o controle de envio e recebimento de mensagens SMS.

Este projeto tinha como requisitos básicos a criação de uma interface segura e simples, proporcionar a facilidade de cobrança, propiciar total controle para o cliente, controlar um histórico para relacionamento, ter completo monitoramento da operação e um sistema "que não caia a toda hora...".

O cenário do projeto era conturbado, pois existia pressão de todos os lados e uma grande escassez de recursos. Entretanto a solução atual tinha que ser substituída rapidamente porque era instável e obtinha pouca robustez sem mecanismos de cobrança alem de não possuir escalabilidade (fontes, versão do protocolo).

Os requisitos do sistema estavam em constante mudança uma vez que o mercado é dinâmico e não maduro.

Era necessária uma abordagem muito criativa, pois o projeto não podia falhar, então foi feita uma construção em partes úteis onde cada parte pronta é um produto usável. O gerenciamento do projeto foi feito todo a partir do escopo que era a variável menos estável, mas bastante funcional.

Os resultados obtidos foram:

1. Sub-produto para remoção de CSP nos torpedos.
2. Prazo, Custo e Escopo atingidos.
3. Usuário interno satisfeito, com produto no mercado.
4. Solução com qualidade e Robustez.

Com isto podemos concluir que a adoção da XP neste projeto foi um sucesso porque atingiu todos os seus objetivos.

4.5 – Resultados

Com o estudo de varias empresas que utilizam a eXtreme Programming como metodologia pode-se notar o que levou a adoção de tal metodologia, em sua maioria foi à insatisfação com os resultados obtidos com outras metodologias e a verificação de que várias das técnicas de XP funcionam perfeitamente e geram grande valor para o desenvolvimento.

Dentre os resultados obtidos podemos citar a redução de riscos e a satisfação do cliente com o resultado final.

Em 100% dos casos estudados se pretende manter esta metodologia devido ao grande sucesso em sua implantação.

Das empresas estudadas apenas uma utiliza a XP em sua totalidade, as outras utilizam quase 100%, porque a implantação requer uma serie de mudanças, inclusive comportamentais. Sendo assim a melhor maneira de se aplicar a XP é aos poucos, mas sempre convergindo para a sua totalidade.

Em relação às dificuldades enfrentadas na adoção da XP a principal foi à quebra de paradigma, ou seja, a resistência à mudança. De acordo com o estudo existe mais dificuldade em adotar algumas práticas como o cliente presente e o desenvolvimento guiado por testes.

5. CONCLUSÃO

Neste trabalho fez-se um estudo em relação à metodologia eXtreme Programming, que faz parte das metodologias ágeis que são hoje uma realidade. Neste estudo foi feita uma abordagem da aplicação desta metodologia de desenvolvimento de software nas empresas.

5.1 – Revisão

No primeiro capítulo deste trabalho fizemos uma pequena introdução ao assunto onde contextualizamos o estudo feito, declaramos a proposta de estudo assim como citamos os objetivos mensuráveis. Além disto, descrevemos toda a organização do trabalho.

No segundo capítulo tivemos uma visão geral da metodologia. Primeiramente vimos um pouco da historia da engenharia de software assim como uma pequena descrição da analise estruturada, com suas fases e conceitos principais e da analise orientada a objetos com seus principais conceitos. Podendo assim traçar um comparativo entre a XP e a abordagem em cascata.

Explicitamos os principais pontos da modelagem ágil, da qual XP faz parte, com suas características, seus objetivos e valores da Aliança Ágil.

Ainda dentro do segundo capítulo levantamos o surgimento, os principais conceitos e as promessas desta nova metodologia. Alem de citar substancialmente seus valores e práticas. Criamos assim uma relação entre as práticas e os valores.

E finalmente exemplificamos um episodio de desenvolvimento, onde vimos como funciona a metodologia na prática.

No terceiro capítulo pudemos detalhar os valores nos quais a XP se baseia. Evidenciamos a formação básica que uma equipe podendo ter esta as suas variáveis. Para melhor compreender a prática de uma equipe que trabalhe com XP descrevemos o ambiente de trabalho ideal para se adotar esta metodologia.

Pudemos também detalhar as práticas centrais que a metodologia propõe. Alem disto citamos as principais métricas usadas para o gerenciamento de um projeto XP, assim como o tipo de gerência deste projeto.

No quarto capítulo vimos como a XP esta sendo empregada como metodologia de desenvolvimento de software nas empresas atuais. Como estudos de caso citamos empresas pioneiras na adoção da XP, como a Improve It do Rio de Janeiro – RJ, a Apoena Software Livre de Porto Alegre – RS e a Naphta de Novo Hamburgo – RS. Para completar o estudo citamos casos como o da Secretaria da Fazenda de São Paulo e da TIM (Telecom Italian Móbile), casos estes pesquisados pela internet. Depois dos estudos de caso apresentamos algumas conclusões que foram consideradas nos casos.

5.2 – Conclusões

Mudança. Esta é a palavra fundamental para se trabalhar com XP. Algumas de suas concepções e abordagens desafiam a forma tradicional de desenvolvimento de software. A XP pede que você faça as coisas de forma diferente. Às vezes, sua orientação é completamente contrária ao senso comum contrariando "verdades estabelecidas".

Com isso você pode amar ou pode odiar a XP, mas ela o levará a olhar de forma diferente a maneira como se desenvolve software.

Hoje em dia muitas empresas caíram e caem na armadilha das mudanças drásticas de coisas que não precisam de alteração, apenas aprimoramento. Porém vimos que não precisamos de uma reengenharia radical para ser mais eficiente. Muitas vezes, a grande mudança é uma simples questão de comportamento.

A XP inova o desenvolvimento de software melhorando a resposta às mudanças dos negócios, uma vez que ela acolhe as mudanças e não as evita.

A eXtreme Programming é baseada em quatro valores (comunicação, feedback, coragem e simplicidade) que são, na verdade, valores derivados. Kent Beck mostra que há um valor mais forte, que fica abaixo dos outros quatro – o respeito. Se os membros da equipe não se preocupam uns com os outros e com o que eles estão fazendo, a XP estará condenada.

As pessoas são cruciais. Elas são envolvidas no desenvolvimento de um produto de software ao longo do seu ciclo de vida, financiando o produto, fazendo agendamentos, gerenciando, testando, usando e se beneficiando disso. Consequentemente, o processo que

guia esse desenvolvimento deve ser orientado a pessoas, ou seja, que trabalhe bem para as pessoas envolvidas.

Por isso não podemos confiar somente no método sem antes termos confiança na própria equipe que desenvolverá o projeto.

Dentre os valores da eXtreme Programming ela dedica enorme atenção à comunicação, que inclusive é um de seus valores fundamentais. Isto porque a indústria de software é caracterizada historicamente por uma infinidade de projetos mal sucedidos. Dentre as mais freqüentes causas apontadas nos relatos, encontram-se deficiências sérias de comunicação entre as equipes de desenvolvimento e seus usuários, bem como deficiências de diálogo entre os próprios membros das equipes de desenvolvimento. Nos últimos anos, estes problemas vem se manifestando com força ainda maior, em função da ênfase acentuada em torno da comunicação escrita.

Podemos verificar que a XP trata os principais riscos de projetos, dentre eles podemos citar os deslizes no cronograma, o cancelamento de projetos, uma alta taxa de erros, uma manutenção impraticável e um negócio mal compreendido.

A XP exige ciclos curtos de entrega, fazendo com que a extensão de qualquer deslize seja limitada e monitorada pela equipe. Ela pede ao cliente que selecione o que é de maior importância para ele fazendo com que o software gerado até o momento tenha sempre o que é de maior valor para o negócio. A XP faz testes sobre a perspectiva tanto do programador quanto do cliente, desenvolvendo testes a cada funcionalidade do programa. Ela mantém um abrangente conjunto de testes, executando e reexecutando após cada modificação assegurando assim um alto padrão de qualidade e mantendo o sistema em excelente condição. Ela convida o cliente para fazer parte da equipe fazendo com que a especificação do projeto seja refinada continuamente, assim o software sempre refletirá a real necessidade do negócio.

Podemos ver que muitas pessoas se assustam com a XP, entretanto nenhuma das idéias da XP é nova. A maioria delas é tão velha quanto à programação, e de certa forma a XP é conservadora, todas as suas técnicas foram comprovadas por décadas. Porém a inovação da XP é colocar todas estas práticas juntas, garantir que elas sejam aplicadas a fundo e que elas apóiem umas as outras ao máximo. Conhecê-las bem, saber quando e como aplicá-las é uma das habilidades técnicas mais importantes de serem dominadas pelos profissionais que trabalham em projetos XP.

Conhecendo bem as técnicas da XP sabemos que determinadas práticas estão associadas de maneira mais forte a determinado valor e que podemos usá-las tanto em projetos pequenos, médios ou grandes.

Olhando para a probabilidade de sucesso de um projeto em função de sua complexidade podemos comparar a XP aos métodos tradicionais, então veremos que tanto faz aplicar processos definidos ou empíricos em projetos pouco complexos (geralmente de pequeno porte) ou muito complexos (geralmente de grande porte). No primeiro caso, você quase sempre terá uma grande chance de sucesso nos dois cenários. No segundo, você quase sempre terá uma grande chance de fracasso em ambos os cenários. Somente nos casos intermediários os métodos ágeis podem apresentar um aumento considerável nas chances de sucesso do projeto. Logo, a única forma de minimizar os riscos em projetos de grande porte é quebrá-los em projetos de média ou baixa complexidade. Aí você estará na zona de conforto para conquistar ganhos na aplicação de métodos ágeis em sua empresa, como a XP.

Não podemos dizer que um modelo é mais eficiente do que outro sem considerarmos o escopo do projeto em que iremos aplicá-lo. No entanto, podemos dizer que um modelo que faz pouco uso dos recursos da retroalimentação (feedback) tende a se tornar mais instável que aqueles retroalimentados. Não estamos fazendo uma suposição, mas uma observação sobre a teoria de sistemas de controle, que considera a realimentação a única forma de controlar de forma eficiente um sistema. Podemos entender a retroalimentação num projeto de software com a participação do usuário (comunicação inter-grupos) e a comunicação entre os membros da equipe (comunicação intra-grupo).

Por outro lado, você pode encarar este problema sob a ótica do determinístico x probabilístico (estocástico). Autores de processos ágeis defendem a idéia de que o desenvolvimento de software não é um processo determinístico (previsível), pois não respeita as leis da física, química, etc. Ele é empírico. Neste caso, escrever software é uma tarefa análoga à redação de um livro, por exemplo, e não à construção de uma casa. Ou seja, você pode recorrer diversas vezes a diversos pontos do texto antes de considerá-lo pronto. Com o modelo cascata, temos que supor que tudo o que faremos ao longo do projeto pode ser previsto no início do projeto. Por exemplo, primeiramente analisamos o sistema, depois projetamos sua arquitetura, depois codificamos seus componentes e depois testamos suas funcionalidades. Quando identificamos qualquer ponto de inconsistência, voltamos para uma etapa anterior do processo. Hoje sabemos que essa forma de desenvolvimento é ingênua perto da quantidade de análises, implementações, correções, testes e refatoração que fazemos diariamente. Por isso é que consideramos o método cascata ineficaz para projetos que exigem adaptabilidade em sua realização.

Ou seja, em projetos de pouca ou muita complexidade, qualquer método se torna eficaz ou ineficaz, respectivamente, independente dele ser determinístico ou probabilístico.

Como a XP está mais no nível operacional da empresa de software ela pode ser combinada com outros métodos de gerenciamentos de projetos. Até mesmo os usuários experientes de XP assumem que a metodologia pode quebrar, mas Beck diz que você deve consertar a metodologia quando ela quebrar. Isto a torna ágil e flexível. Existem inúmeras possibilidades de se utilizar XP com outras metodologias como o LSD, Scrum ou até mesmo RUP. Isto tudo depende muito do core business de cada empresa, por exemplo, produtos, serviços, etc. Depende também da cultura de cada uma delas, mas é totalmente aceitável e possível.

A aplicação da XP pode se tornar difícil às vezes, alguns obstáculos podem surgir na sua aplicação, são primeiramente a cultura e as emoções, principalmente o medo, sentimento esse que normalmente acompanha os homens em relação à mudança.

A XP é relativamente simples em suas técnicas, práticas e valores, mas, no entanto não são fáceis de ser aplicadas em conjunto. A parte difícil da XP é manter suas peças reunidas e equilibradas.

Porém se estiver muito difícil aplicar a XP você poderá estar optando pela metodologia errada. Porque mesmo os seus limites não sendo ainda muito claros existem casos onde não se deve utilizá-la.

Uma cultura que não contribui para a XP é aquela onde a equipe é requisitada a trabalhar horas e mais horas para provar o seu "comprometimento com a empresa". Não se consegue executar XP se você estiver cansado. Tamanho obviamente é importante, provavelmente um projeto com 100 programadores não terá sucesso com XP. Existem restrições em relação a equipes grandes (pela importância da comunicação), mas não a projetos grandes.

Vemos que não é indicado o uso da XP em empresas que utilizam sistema de premiação individual, pois XP se baseia fortemente no trabalho em equipe. E este tipo de premiação pode influenciar os membros da equipe a manterem comportamentos individualistas e competitivos. Contudo, XP é um processo que se baseia em cooperação e não existe espaço para competição.

Como o ambiente de trabalho influência bastante no método de trabalho, sem um ambiente adequado fica muito difícil aplicar XP. É fundamental que as pessoas possam trabalhar próximas umas das outras, é importante que os programadores tenham espaço para trabalhar em duplas e é interessante que a organização apóie mudanças estruturais caso seja necessário.

Os itens citados aqui são apenas alguns dos muitos que podem dificultar ou impedir completamente a adoça da XP. Tão importante quanto saber usá-la é saber quando não utilizá-la.

Temos observado que a primeira experiência no uso da XP geralmente é carregada de uma série de "pré-conceitos" sobre uma negociação de TI. As pessoas / empresas estão muito acostumadas ao antigo paradigma de desenvolvimento e negociação de TI. Quebrar este ciclo histórico requer um misto de coragem e ousadia tanto do desenvolvedor como do cliente.

Com toda esta discussão vimos que XP não é só uma metodologia, é praticamente uma filosofia. Um ritmo de vida, não só de desenvolvimento.

5.3 – Propostas para trabalhos futuros

Para trabalhos futuros fica a sugestão de implementação de um software de gerência de projetos baseados em XP, uma vez que existem poucos softwares específicos para gerência de projetos que utilizem XP.

Outra sugestão seria o estudo sobre a aplicação de métodos ágeis no desenvolvimento e gerenciamento de projetos de software, visto que nos últimos anos têm chamado a atenção da comunidade de desenvolvimento de software a publicação de vários métodos auto-denominados de ágeis.

6. BIBLIOGRAFIA

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. **Extreme Programming guia prático.** São Paulo: Campus. 2002.

BECK, Kent. **Extreme Programming explained.** Boston: Addison-Wesley Professional. 2000.

BECK, Kent. **Programação Extrema explicada.** Porto Alegre: Bookman. 2004.

BECK, Kent; FOWLER, Martin. **Planning Extreme Programming.** 1. ed. Boston: Addison-Wesley Professional. 2000.

BECK, Kent. **Embracing change with Extreme Programming**. IEEE Computer, 32:70--77, Oct. 1999.

BECK, Kent et al. **The Agile Manifesto**. Disponível em <http://agilemanifesto.org>. Acesso em 20 set. 2004.

COAD, Peter. **Página pessoal**. Disponível em <http://www.coad.com/peter>. Acesso em 25 out. 2004.

COCKBURN, Alistair; WILLIAMS, Laurie. **Agile Software Development: It's About Feedback and Change.** IEEE Computer, Nova Iorque, v. 36, n.6, Jun 2003. p. 57 – 66.

Extreme Programming: a gentle introduction. Disponível em <http://www.extremeprogramming.org>. Acesso em 25 out. 2004.

FOWLER, Martin et al. **Refactoring: Improving the Design of Existing Code.** 1. ed. Boston: Addison-Wesley Professional. 1999.

FOWLER, Martin. **Página pessoal.** Disponível em <http://www.martinfowler.com>. Acesso em 11 out. 2004.

FOWLER, Martin. **The New Methodology.** Disponível em <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em 26 out. 2004.

HARTMANN, Julio. **Estudo sobre a aplicação de métodos ágeis (Agile) no desenvolvimento e gerenciamento de projetos de software.** 2003. 69f. Trabalho individual (Instituto de Informática). Programa de Pós – Graduação em Computação, Universidade Federal do Rio Grande do Sul, Rio Grande do Sul. 2003.

JEFFRIES, Ron; ANDERSON, Ann; HENDRICKSON, Chet. **Extreme Programming installed.** Boston: Addison-Wesley Professional. 2001.

JEFFRIES, Ron. **Xprogramming.com: an Extreme Programming resource.** Disponível em <http://www.xprogramming.com>. Acesso em 13 nov. 2004.

NEWKIRK, James et al. **eXtreme Programming in Practice**. Boston: Addison-Wesley Pub Co. 1999.

PAULK, Mark C. **Extreme Programming from a CMM perspective.** Carniege Mellon University, Software Engineering Institute, 2001

POHREN, Daniel. **XpManager – Uma Ferramenta de Gerência de projetos Baseados em Extreme Programming.** 2004. 172f. Trabalho de Conclusão de Curso (Curso Ciência da Computação). Instituto de Ciências Exatas e Tecnológicas, Centro Universitário Feevale, Novo Hamburgo, Rio Grande do Sul. 2004.

PRESSMAN, Roger S. **Engenharia de software.** São Paulo: Makron Books. 1995.

PRESSMAN, Roger S. **R.S. Pressman & Associates, Inc.** Disponível em <http://www.rspa.com>. Acesso em 28 set. 2004.

RUMBAUGH, James et al. **Object-Oriented Modeling and Design**. Prentice-Hall.

TELES, Vinícius Manhães. **Extreme Programming.** São Paulo: Novatec. 2004.

XISPÊ. Disponível em <http://www.xispe.com.br>. Acesso em 05 ago. 2004.

YOURDON, Edward. **Can XP projects grow?** Disponível em <http://www.yourdon.com/articles/0107cw.html>. Acesso em 12 ago. 2004.

YOURDON, Edward. **Página pessoal**. Disponível em <http://www.yourdon.com>. Acesso em 22 ago. 2004.

WAKE, William C. **Extreme Programming explored.** Boston: Addison-Wesley Professional. 2002.

WAKE, William. **XP123 – Exploring Extreme Programming.** Disponível em <http://www.xp123.com>. Acesso em 28 set. 2004.