



**UNIVERSIDADE TIRADENTES - UNIT
CURSO DE SISTEMAS DE INFORMAÇÃO**

Darlan Menezes
Denysson Mota
Fernando Teles
José Jornando
Tiego Barreto

EXTREME PROGRAMMING

ARACAJU
2007

Sumário

1	Introdução	5
2	Historia do XP	7
3	Objetivos e Diferenciais	8
4	Metodologia, método ou processo de desenvolvimento de software?	9
5	Valores do Extreme Programming	10
5.1	Comunicação	11
5.2	Feedback	12
5.3	Comunicação	13
5.4	Simplicidade	14
5.5	Coragem	15
6	Princípios	16
6.1	Auto-Semelhança	16
6.2	Benefício Mútuo	17
6.3	Diversidade	17
6.4	Economia	18
6.5	Falha	18
6.6	Fluxo	19
6.7	Humanismo	19
6.8	Melhoria	20
6.9	Oportunidade	20
6.10	Passos pequenos	21
6.11	Qualidade	21
6.12	Redundância	21
6.13	Reflexão	22
6.14	Responsabilidade Aceita	23
7	Papéis	23
7.1	Analistas de teste	23
7.2	Arquitetos	24
7.3	Designers de Interação	24
7.4	Executivos	24
7.5	Gerentes de Projeto	25
7.6	Gerentes de Produto	25
7.7	Programadores	26
7.8	Recursos Humanos	26
7.9	Redatores Técnicos	26
7.10	Usuários	27
8	Práticas Primárias	27
8.1	Ambiente Informativo	29
8.2	Build de Dez Minutos	29
8.3	Ciclo Semanal	30
8.4	Ciclo Trimestral	31
8.5	Desenvolvimento Orientado a Testes	31
8.6	Design Incremental	32
8.7	Equipe Integral	32

8.8 Folga.....	33
8.9 Integração Contínua	33
8.10 Programação em Par	34
8.11 Sentar-se Junto.....	35
8.12 Trabalho Energizado.....	35
9 Práticas Corolárias	36
9.1 Análise da Raiz do Problema.....	36
9.2 Base de Código Unificada.....	37
9.3 Código Coletivo.....	38
9.4 Código e Testes.....	38
9.5 Continuidade da Equipe.....	39
9.6 Contrato de escopo negociável.....	40
9.7 Envolvimento do Cliente Real.....	45
9.8 Equipes que Encolhem	45
9.9 Implantação Diária	46
9.10 Implantação Incremental.....	47
9.11 Pagar por uso	47
10 Outras Práticas.....	48
10.1 Reunião em Pé	48
10.2 Informações	49
10.3 Emoções	51
10.4 Refatoração	52
10.5 Metáfora.....	53

1 Introdução

A muito tempo a indústria de software vem passando por grandes transformações e novos desafios, entre eles desenvolver softwares com qualidade, no menor tempo possível e que atendam as necessidades dos clientes.

Com estes novos desafios a indústria de software passou a dar valor a algumas áreas da informática, como a engenharia de software e qualidade de software, com intuito de atender as exigências do mercado.

O empenho em analisar a contínua expansão de nossa atividade acarreta um processo de reformulação e modernização das novas proposições.

No final dos anos 90, muitas metodologias de desenvolvimento começaram a atrair a atenção pública. Cada uma destas metodologias tinha uma combinação única de elementos já antigos e consolidados da gerência de projetos, novos componentes completamente únicos e inovadores e elementos antigos, porém remodelados para suprir as necessidades que surgiram.

Porém, a pesar de existirem diferenças entre as teorias e definições da base das metodologias, existia um elemento que todos tinham em comum: a ênfase na colaboração mútua entre a equipe de desenvolvimento do projeto e os especialistas do negócio da empresa. Para contribuir com esta colaboração eles propuseram que se substituísse a documentação escrita por comunicação presencial face-a-face, equipes pequenas, fortes, unidas e auto-organizadas, entre outros elementos que ajudam para que o projeto venha a ter sucesso.

Por outro lado, o julgamento imparcial das eventualidades garante a contribuição importante do grupo na determinação e utilização dos paradigmas

colaborativos. O empenho em analisar a valorização de fatores subjetivos promove a alavancagem dos índices pretendidos. Neste sentido, a contínua expansão desta atividade estimula a mudança dos modos de operação convencionais, com o simples objetivo de aumentar a produtividade por hora trabalhada. Percebemos, cada vez mais, que a revolução dos costumes estende o alcance e a importância da gerencia de projetos.

É importante questionar o quanto o comprometimento entre as equipes exige a precisão e a definição dos conhecimentos estratégicos para atingir a excelência. No mundo atual, a complexidade dos estudos efetuados pode nos levar a considerar a reestruturação das formas de ação.

Neste sentido, o desafiador cenário globalizado, onde todos os projetos tem um prazo pequeno, se comparado com o trabalho que acarreta, nos obriga à análise dos paradigmas colaborativos e da gerencia de projetos de software.

Dentre estas novas metodologias destacamos o Extreme Programming, metodologia focada completamente no gerenciamento de projetos de software e que tem sido testada em empresas de todos os portes, com grande numero de sucessos. Abordaremos nos próximos capítulos todos os elementos do XP, seu inicio, objetivos, propostas e abordagens frente a determinadas situações.

Analisando os motivos para a baixa taxa de sucesso dos projetos desenvolvidos com modelos tradicionais, cita-se os principais e mais comuns motivos comuns entre eles o tempo elevado entre cada fase do projeto, não acompanhando as mudanças de requisitos do projeto; a falta de conhecimento por parte do cliente da sua real necessidade, dando margem às especulações dos desenvolvedores e a forte linearidade no desenvolvimento do projeto.

2 Historia do XP

No começo dos anos 90, um americano chamado Kent Beck pensou em uma maneira mais eficiente de desenvolver softwares. Ele tinha recentemente trabalhado um tempo com um colega, chamado Ward Cunningham. Eles tinham experimentado uma abordagem para o desenvolvimento de software que tinha como meta fazer tudo ser bem mais simples e eficiente.

Desta maneira, a percepção das dificuldades passadas em projetos anteriores e com outras abordagens de gerencia causou impacto tanto direto como indireto na reavaliação do modelo de gestão utilizado, fazendo com que ele idealizasse um método completamente inovador.

Kent observava tudo aquilo que facilitava a criação de software e também tudo aquilo que dificultava essa tarefa. Em março de 1996, Kent deu inicio a um projeto na Daimlerchrysler usando novos conceitos de desenvolvimento de software. O nome deste projeto era **C3** (sigla do inglês Chrysler Comprehensive Compensation). O resultado foi a metodologia da Programação Extrema (do inglês Extreme Programming, XP). Muitas das práticas pregadas pelo XP surgiram e foram 'alimentadas' nesse projeto.

Segundo os autores do XP, existem duas observações a serem feitas sobre o projeto C3:

- É um simples sistema de folha de pagamento.
- O problema do "simples folha de pagamento" é que um time utilizando contrato fechado tentou fazer o sistema e não obteve sucesso.

O que Kent percebeu trabalhando nesse projeto é que existem quatro dimensões através das quais você pode melhorar qualquer projeto de software.

Você precisa melhorar a comunicação. Você precisa procurar simplicidade. Você precisa ter retorno em quanto à qualidade das tarefas que esta realizando. E você precisa sempre proceder com coragem.

Comunicação, Simplicidade, Retorno e Coragem, estes são os quatro valores principais que devem ser buscados por programadores adeptos ao XP.

3 Objetivos e Diferenciais

A XP tem como diferencial básico que foi criado e desenvolvido para a criação de softwares, enquanto outras metodologias são genéricas, funcionando para todo tipo de projeto.

A XP também assume que os requisitos do sistema mudam constantemente, diferente das outras metodologias, mesmo as mais leves.

A partir dessa premissa criou-se o termo "Metodologia Ágil", que na verdade serve para classificar metodologias criadas para responder com velocidade à mudanças nas especificações do projeto. É importante frisar aqui que XP criou o termo "Metodologia Ágil" e não o contrário.

A partir do momento em que você assuma, verdadeiramente, que o normal é que os requisitos mudem, sua maneira de encarar o desenvolvimento de software muda, pois você pára de por a culpa no cliente pelos atrasos devido a mudança de escopo, uma vez que isso é normal; você encara retrabalho em classes e mudanças no sistema como coisa comum já que mudar algo que está funcionando não pode causar pânico; você pára de pensar que a Análise do Sistema é a mesma coisa que desenvolver o sistema, e passa a encará-la como uma ajuda, um suporte ao desenvolvimento de software.

4 Metodologia, método ou processo de desenvolvimento de software?

Existe ainda muita discussão em torno disso. Alguns profissionais crêem que, por definir a maneira de como se faz as coisas, seria uma metodologia, outros afirmam veemente que seria na verdade um processo devido a que eles encontram no XP vários procedimentos a serem realizados para alcançar o objetivo, outros um simples método de desenvolvimento.

Porém um dos seus criadores, Kent Beck, refere-se à XP como "DISCIPLINA" e não "METODOLOGIA". E ele está correto, pois XP não se baseia em um procedimento atrás do outro, e sim em regras que devem funcionar o tempo todo, sempre sendo aplicadas e observadas. Não há realmente uma "seqüência de passos" a ser seguida, e sim um maneira de se fazer todas as coisas.

Para um total aproveitamento do XP, essas "regras" devem ser sempre seguidas. Aqui na XP estas regras são chamadas de PRÁTICAS. Existem 12 práticas básicas, sem contar as que são recomendadas, sugeridas ou que vem com a execução das outras. São elas que garantem que a VISÃO do XP sobre o desenvolvimento de software seja possível.

Outra coisa diferente dos outros métodos é que XP é "sinérgico", ou seja, o todo é maior que a soma das partes. Isso significa que as práticas do XP suportam umas às outras. As falhas que venham a ocorrer com uma prática são corrigidas por outras práticas.

Isso não significa que você precisa adotar tudo para perceber alguma diferença no desenvolvimento, mesmo a implantação de algumas práticas já mostra

uma diferença significativa.

Outra dúvida que existe é que seria correto afirmar que o XP é um paradigma. Na verdade o XP tem um papel importante no paradigma atual. "Menos processos, mais simplicidade, maior capacidade de reação". Mas ele em si não é um paradigma.

O paradigma anterior: "Formalidade dá controle, mais genérico para não mudar" já vem sendo combatido a um bom tempo, é possível achar artigos desde 1994, ou antes, mesmo com idéias contrárias a ele. Mas diversos autores passam a impressão que o XP é quem deu a maior força a esse movimento. Por isso de todos os termos para descrever o XP, o menos aconselhado é "paradigma". Mas sim é recomendado tratá-lo como catalisador importante que acelerou muito a aceitação dessa outra maneira de se desenvolver software.

5 Valores do Extreme Programming

Assim como valores, no âmbito filosófico, significa tudo o que corresponde às normas ideais para o seu tipo e, por isso, é desejado e desejável, no escopo do XP os valores vão pelo mesmo caminho, tornando-se tudo aquilo que é recomendado ou desejado que as equipes tenham para seguir em frente com o projeto.

Cada pessoa que se envolve com desenvolvimento de software tem um sentimento sobre aquilo que realmente importa. Quando juntos em uma equipe, esses sentimentos individuais podem ou não ser semelhantes. Uma pessoa pode pensar que o que realmente vale é pensar atentamente em todas as decisões de design concebíveis antes de implementá-las, enquanto outra pode achar que

importante mesmo é não ter nenhum tipo de restrições sobre sua liberdade pessoal.

O maior problema que existe em relação ao que as pessoas pensam saber sobre o processo de desenvolvimento de software é o fato de que elas geralmente se concentram unicamente em ações individuais. Mas o que realmente importa durante este processo não é como uma única pessoa se comporta, mas sim como os indivíduos que formam parte de uma equipe e parte de uma organização se comportam.

Por exemplo, geralmente os desenvolvedores são adeptos de estilos de codificação. Apesar de haver estilos que são, sem lugar a dúvidas, melhores que outros, a questão mais relevante em relação a estilos de codificação é o fato de que a equipe como um todo escolha adotar um estilo em comum, ajudando a ter uma compreensão somente do código, eliminando tempo em entender também o estilo de codificação do editor. Estilos de codificação muito peculiares e os valores revelados por eles, como liberdade pessoal a qualquer custo, não ajudam em hipótese alguma a equipe a ter sucesso.

Assim é necessário que os participantes da equipe tenham em mente todos estes valores e que, além de conhecê-los, os ponham em prática diariamente e, se possível, inconscientemente. Somente desta maneira é possível que sejam contornados todos aqueles problemas que não são diretamente relacionados ao projeto e sim com os indivíduos e suas personalidades.

Abaixo citaremos e explicaremos estes valores tão defendidos pelos precursores do XP.

5.1 Comunicação

O cliente de um projeto de software tem um conjunto de problemas que deseja solucionar com o sistema em desenvolvimento e possui algumas idéias sobre que funcionalidades podem resolvê-los. Por sua vez, desenvolvedores possuem conhecimento sobre aspectos técnicos que influenciam a forma de solucionar o problema do cliente. Para que os desenvolvedores compreendam o que o cliente deseja e este último entenda os desafios técnicos que precisam ser vencidos, é preciso que haja comunicação entre as partes.

Embora existam inúmeras formas de se comunicar idéias, algumas são mais eficazes que outras. Por exemplo, quando duas pessoas estabelecem um diálogo presencial, inúmeros elementos da comunicação colaboram para a compreensão do assunto, tais como gestos, expressões faciais, postura, palavras verbalizadas, tom de voz, emoções, entre outros. Quanto maior a capacidade de compreensão, maiores as chances de evitar problemas como ambigüidades, entendimento equivocados, entre outros. Diálogos são mais eficazes que videoconferências, que são melhores que telefonemas, que são mais expressivos que emails e assim sucessivamente. Conscientes disso, aqueles que trabalham com XP priorizam o uso do diálogo presencial, com o objetivo de garantir que todas as partes envolvidas em um projeto tenham a chance de se compreender da melhor maneira possível.

5.2 Feedback

O cliente aprende com o sistema que utiliza e com este aprendizado consegue reavaliar o produto recebido, com isso pode realimentar a equipe de desenvolvimento com as suas reais necessidades. Com o feedback , o cliente

conduz o desenvolvimento do seu produto, estabelece prioridades e informa aquilo que é realmente importante.

Analogamente, há o feedback dado pelo desenvolvedor ao cliente, onde o mesmo aponta riscos, estimativas e alternativas de design. Este retorno é o aprendizado do desenvolvedor sobre o que o cliente deseja.

Com este valor, o tempo de defasagem entre as fases do projeto são extremamente pequenos, o cliente está o tempo todo em contato com a equipe de desenvolvimento, muito diferente dos modelos tradicionais, onde o cliente entrava em contato com a equipe um bom tempo depois do último feedback dado.

Um ponto que muitas empresas de software falham é não dar valor ao que o cliente realmente deseja, utilizam cegamente metodologias e acabam esquecendo o real propósito de um software: facilitar o trabalho de pessoas.

Com isto muitos sistemas acabam dificultando e burocratizando as tarefas das pessoas e como defesa as empresas alegam ter um produto genérico e que atenda as normais legais.

5.3 Comunicação

Para que o feedback entre cliente e desenvolvedor possa ser efetuado com sucesso é necessário ter uma boa comunicação entre eles. A XP prega que esta comunicação ocorra da forma mais direta e eficaz possível, oferecendo agilidade aos assuntos tratados. Recomenda-se o contato direto (face-a-face) entre cliente e desenvolvedor, para evitar qualquer tipo de especulação ou mal entendido entre as partes e para que possíveis dúvidas possam ser resolvidas de imediato.

Além de sanar as dúvidas no desenvolvimento, o cliente deverá estar

disponível para a equipe, ou mesmo presente no ambiente de trabalho da empresa. Isto fará com que o cliente entenda o sistema e enriquecerá os relacionamentos pessoais, criando um elo de parceria e confiança mútua.

Algumas equipes não se adaptam bem a este valor. Este problema deve ser trabalhado em conjunto com a equipe. Enquanto não se acostumarem a falar e a trocar idéias com seus companheiros o sucesso da metodologia estará comprometido. Membros introvertidos são desaconselháveis para equipes de XP.

5.4 Simplicidade

Para que o cliente possa aprender durante o projeto e consiga dar o feedback necessário à equipe, não basta apenas uma boa comunicação, é necessário que os desenvolvedores implementem da forma mais simples possível o que o cliente deseja.

A lei é: faça a coisa mais simples que pode funcionar. Com esta filosofia, o cliente terá a funcionalidade rapidamente e da forma desejada, dando um feedback instantaneamente evitando especulações. O desenvolvedor deve implementar apenas o necessário para que o cliente tenha seu pedido atendido.

Ser simples não é um ato de desespero, é um ato de consciência e absoluta precisão. Muitas pessoas confundem simplicidade e facilidade. O mais simples nem sempre é o mais fácil e também não é escrever menos código. Simplicidade significa codificar o necessário para que um requisito seja atendido e entregue ao cliente.

Evita-se suposições, o futuro é incerto e por causa disso não necessita atenção. Os requisitos evoluem gradativamente em conjunto com o sistema e a

arquitetura do projeto. Algumas vezes, o que é necessário hoje será descartado amanhã, e outras vezes o que seria necessário num futuro próximo nunca será utilizado.

5.5 Coragem

Por ser um processo de desenvolvimento novo e baseado em diversas premissas que contrariam o modelo tradicional, o XP exige que os desenvolvedores tenham coragem para:

- Desenvolver software de forma incremental;
- Manter o sistema simples;
- Permitir que o cliente defina prioridades;
- Fazer desenvolvedores trabalharem em pares;
- Investir tempo em refactoring;
- Investir tempo em testes automatizados;
- Estimar histórias na presença do cliente;
- Expor código a todos os membros da equipe;
- Integrar o sistema diversas vezes ao dia;
- Adotar ritmo sustentável de desenvolvimento;
- Abrir mão de documentos que servem como defesa;
- Propor contratos de escopo variável;
- Propor a adoção de um processo novo.
- Assumir em relação ao cliente possíveis atrasos e problemas de implementação;
- Colocar desenvolvedores e clientes frente a frente;

- Implantar uma nova versão do sistema no cliente semanalmente;
- Apostar em seus colaboradores aumentando suas responsabilidades;
- Modelar e documentar apenas quando for de extrema necessidade.

6 Princípios

XP define um conjunto de princípios que devem ser seguidos por equipes que forem usar XP em projetos. Os princípios servirão para ajudar na escolha de alternativas de solução de problemas durante o curso do projeto. Alternativas para solução de problemas devem sempre buscar estar perto da filosofia do XP, deve-se usar o mínimo possível utilizar princípios que se afastem da filosofia do XP.

Os princípios são elos que aproximam os valores e as práticas. Valores são tudo aquilo em que acreditamos, são eles que guiam nossas ações. Práticas são nossas ações propriamente ditas. Princípios também são guias úteis para serem seguidos quando ainda não há uma prática específica para solucionar determinado problema. Os princípios descritos a seguir guiam as práticas de XP, porém no percorrer do desenvolvimento do projeto outros princípios podem ser adotados para solucionar determinados problemas.

6.1 Auto-Semelhança

O princípio da auto-semelhança defende que soluções bem sucedidas em determinado contexto deve ser adotado em outros contextos. "A proposta inicial para o ciclo semanal assemelhasse ao modelo Cascata de Desenvolvimento, no qual os testes de sistema são realizados após a implementação das funcionalidades"(BECK,

1999 Apud CAVALCANTI).

6.2 Benefício Mútuo

Para todos os problemas encontrados no decorrer do desenvolvimento devem se buscar soluções que beneficiem a todos os envolvidos. Em situações críticas, soluções unilaterais mais simples podem ser tentadoras, mas também pode provocar problemas de relacionamento entre os envolvidos, o que não se tornaria uma solução ideal. Soluções como programação em par podem ser adotadas, o que traria um grande melhora nos sistemas, pois o desenvolvimento em par diminuiria o número de bugs no sistema, além de resolver outros problemas como ausência de funcionários.

6.3 Diversidade

É importante possuir uma equipe onde os membros possuam diversas habilidades, e não montar equipes onde todos sejam iguais. O conceito de "equipe completa" é muito importante para exploração idéias diferentes e inovadoras. Em software, diferentes abordagens para um mesmo problema podem implicar em enormes diferenças no tempo e custo de implementação da solução. Um time diversificado pode ser uma fonte de conflitos, porém se tornam equipes maduras, que respeitam uns aos outros e vêem conflitos como oportunidades para a busca da melhor solução e não como brigas. Resolver conflitos de personalidade quando eles estão atrapalhando a equipe é função do gerente.

6.4 Economia

O desenvolvimento de projetos deve estar diretamente ligado as propósitos da empresa, ou seja, não deve-se apenas analisar o ponto de vista técnico, mas também o ponto de vista econômico. Os projetos devem ser aprovados de acordo com a necessidade da empresa, então é importante dar prioridade àqueles projetos onde o retorno será imediato ou que atenda as necessidades urgentes da empresa. Práticas de iteratividade permitem colocar a todo tempo novas funcionalidades no software, o que viabiliza que se coloque o mais cedo possível software no mercado, mesmo que este ainda não possua todas as funcionalidades.

Outro ponto importante a ser analisado são as opções que este novo software trará, não se deve apenas analisar as funcionalidades de um software hoje, é importante observar também o que ele será capaz de fazer no futuro, por quanto tempo aquele software atenderá as necessidades de minha empresa, e o quanto fácil migrar de plataforma caso a atual não atenda mais as necessidade da empresa.

6.5 Falha

"É melhor tentar e falhar do que gastar tempo tentando realizar tudo certo da primeira vez". (CAVALCANTI, Ricardo)

Falhar é aprender, e não desperdício, além de que há uma perda enorme de tempo ao tentar buscar a solução perfeita na primeira tentativa principalmente em se tratando de desenvolvimento de software onde busca de solução para os mais diversos problemas desconhecidos são comumente encontrados.

6.6 Fluxo

Deve-se buscar as entregas freqüentes e regulares ao invés de versões com grandes mudanças. Pequenos pedaços, entregues freqüentemente, dão fluidez ao trabalho da equipe e permitem que ela receba feedback cedo sobre o que está sendo produzido. Por sua vez, clientes têm a oportunidade de receber coisas novas freqüentemente, o que ajuda a administrar a relação com o cliente, e sobre o ponto de vista econômico, traz receitas o mais breve possível. Trabalhar em "pequenos passos" traz um fluxo para equipe, facilitando na exposição de problemas, como também na resolução dos mesmos.

6.7 Humanismo

É importante entender que softwares são desenvolvidos por seres humanos, e como ser humanos, estão sujeitos a falhas. Conhecer a equipe e ponto crucial no desenvolvimento de um software, deve-se buscar o máximo de potencialidade do que há de melhor em cada membro, e suprir o que há de pior, e assim trazer equilíbrio ao balancear as necessidades pessoais de cada membro da equipe e as necessidades da organização. Desequilíbrio pode gerar baixa produtividade.

Segundo CAVALCANTI (2006), as necessidades pessoais dos trabalhadores que podem ser supridas no ambiente de trabalho são: a noção de segurança ligada à manutenção do emprego; o sentimento de realização pessoal na execução do seu trabalho; o sentimento de pertinência à equipe; a oportunidade de

evolução das suas competências e perspectivas; e a habilidade de entender e ser entendido pelos outros. Outras necessidades, como descanso e socialização, não precisam, necessariamente, ser atendidas pela organização. É importante perceber que um projeto de desenvolvimento de software é extremamente dependente da capacidade criativa dos integrantes da equipe. Eles também precisam de tempo fora do trabalho para recarregar as energias.

6.8 Melhoria

Nenhum software é perfeito, mas em XP a busca de melhoria deve ser diária. O aprendizado traz melhoria, deve-se buscar sempre o aprendizado mesmo que este seja decorrente de erros, pois só com o aprendizado é que se torna possível buscar melhoria. O uso de ciclos semanais são importante para o aprendizado, pois os membros podem verificar em semanas passadas o que deu certo e o que deu errado, e buscar não cometer os mesmos erros.

6.9 Oportunidade

Em XP nem sempre tudo dá certo, vários acontecimentos ocorrem durante todo o processo de desenvolvimento, o que pode se tornar um problema dependendo de como sua equipe reage. A equipe deve enxergar problemas como oportunidades de aprendizado, essa cultura deve estar sempre presente na equipe porque em se tratando de desenvolvimento de software problemas sempre ocorrerão.

6.10 Passos pequenos

Mudanças devem ocorrer a passos pequenos, mudanças grandes são muito perigosas. Trabalhar a passos de bebê traz segurança além de facilitar na mudança de direção sem grandes prejuízos. Trabalhar a pequenos passos não significa que o desenvolvimento será devagar, ao contrário trabalhar com pequenos passos, mas rápidos, que parecerão largos saltos externamente.

6.11 Qualidade

Deve-se sempre manter padrões altos de qualidade. A má qualidade de um software pode trazer perdas econômicas, além de trazer insatisfação e ansiedade para o cliente. Outros pontos negativos também podem ser observados em um software de má qualidade, que seria a perda de tempo da equipe em consertar problemas decorrentes da má qualidade, além de trazer dificuldades para adaptação do software a novas necessidades do cliente.

Sem sombra de dúvidas trazer qualidade para um software pode custar caro, pois qualidade tem preço e isso é indiscutível, mas a falta de qualidade pode ter um preço ainda maior. Empresas que investem em qualidade são certamente as empresas mais lucrativas do mundo, como podemos citar a TOYOTA que tem seus carros consistentemente considerados de altíssima qualidade.

6.12 Redundância

"Os problemas difíceis e críticos em desenvolvimento de software devem

ser resolvidos de várias formas diferentes. Mesmo que uma solução falhe completamente, as outras soluções irão prevenir um desastre. O custo da redundância é mais que pago pela economia de não ter um desastre". (TELES, Vinícius)

A busca por defeitos em XP é tratada de forma redundante, como podemos observar nas diversas práticas utilizadas, como programação em pares, integração contínua, programação orientada a testes, envolvimento do cliente, todas são práticas que auxiliam a busca e a eliminação dos defeitos. O tratamento redundante para este problema é contrabalanceado pelo ganho em qualidade do produto.

6.13 Reflexão

"Boas equipes não apenas fazem seu trabalho, mas também pensam sobre como estão trabalhando e por que estão trabalhando. Elas analisam o porquê de terem tido sucesso ou falhado. Elas não tentam esconder seus erros, mas os expõem e aprendem com eles". (TELES, Vinícius)

Reflexão não é um exercício puramente intelectual. Os membros da equipe podem se inspirar analisando dados, mas também podem aprender a partir de sua intuição. Refletir não somente no trabalho, mas também em casa, conversando com seus filhos e esposa, férias e atividades não relacionadas a desenvolvimento de software, como leituras e outras atividades provêm oportunidades individuais para pensar sobre como e por que você está trabalhando da forma como está trabalhando. Refeições em conjunto e pausas para um café provêm momentos informais para reflexão em conjunto.

6.14 Responsabilidade Aceita

Em um ambiente gerencial o ideal é atribuir responsabilidade, e não as impor, ou seja, os membros aceitar a responsabilidade sobre as tarefas. Muitas vezes a imposição de responsabilidade pode ocorrer, mas não é o ideal.

7 Papéis

Na metodologia XP temos uma grande variedade de papéis que podem ser desempenhados pelos membros da equipe. Alguns são estratégicos, outros de apoio, alguns descartáveis outros imprescindíveis. Contudo, como qualquer metodologia, requer disciplina e fiscalização. Há também a possibilidade de um mesmo membro da equipe desempenhar mais de um papel. Abaixo segue os papéis:

7.1 Analistas de teste

Analistas de teste têm um papel proativo. No início de cada iteração, eles ajudam clientes e desenvolvedores a escrever testes e também trabalham com os desenvolvedores ao longo da iteração, ajudando-os a automatizar os testes. Quando a equipe não consegue automatizar alguns testes, os analistas de teste os executam manualmente.

7.2 Arquitetos

Arquitetos de software ajudam os desenvolvedores no dia-a-dia através da programação em par. Além disso, utilizam seus conhecimentos para ajudar a equipe a fazer refatorações em larga escala, em passos curtos e seguros.

Arquitetos também podem ajudar a equipe a criar testes mais amplos, que exercitem a arquitetura como um todo. Tais testes são usados, entre outras coisas, para comunicar os propósitos e as características da arquitetura.

7.3 Designers de Interação

Designers de interação trabalham próximo aos clientes. Eles os ajudam a escrever histórias e escolher metáforas consistentes para o projeto. Além disso, ajudam a criar a interface e a refiná-la continuamente ao longo do tempo. Designers de interação também avaliam o uso das funcionalidades pelos clientes à medida que vão sendo entregues no final de cada ciclo semanal.

7.4 Executivos

Executivos ajudam na definição do escopo do projeto. Eles comunicam os objetivos do projeto dentro do contexto geral da organização e asseguram que as histórias estejam alinhadas com tais objetivos. Além disso, ajudam a comunicar o progresso da equipe para as demais áreas da organização. Executivos também ajudam a equipe a fazer seu trabalho, administrando as pressões dos usuários e removendo obstáculos. Isso ajuda a desenvolver a coragem da equipe e permite que

ela avance mais rapidamente e com menos transtornos.

7.5 Gerentes de Projeto

Gerentes de projeto servem de ponte entre a equipe, os clientes e eventuais fornecedores. Ele assegura que as pessoas certas dialoguem dentro da equipe e fora dela. Nesse sentido, age como um facilitador no fluxo de comunicação do projeto. Seu propósito não é controlar a informação, mas assegurar que as pessoas consigam se comunicar **prontamente**.

Gerentes de projeto também devem monitorar o progresso da equipe e ajudá-la a perceber continuamente tudo o que já foi conquistado. Devem motivar os membros da equipe através do orgulho de estar construindo um software bem feito. Nesse sentido, deve permitir que os desenvolvedores façam um trabalho de alta qualidade, o que significa administrar pressões externas e comunicar o que está acontecendo a todos os envolvidos no projeto.

7.6 Gerentes de Produto

Gerentes de produto procuram definir histórias e temas que ajudem o produto a tomar um corpo coerente e harmônico. Eles ajudam a definir prioridades e esclarecem aspectos das histórias ao longo do desenvolvimento. Finalmente, gerentes de produto ajudam a reduzir o escopo quando a equipe está atrasada por alguma razão.

7.7 Programadores

Programadores trabalham em pares implementando histórias. Eles também estimam as histórias no Jogo do Planejamento e automatizam tarefas repetitivas. Também são responsáveis por criar testes automatizados para tudo o que produzem. Isso é feito com a prática de desenvolvimento orientado a testes. Além de criarem novas funcionalidades, também refatoram o sistema permanentemente para aprimorar a arquitetura, eliminar duplicações e tornar o código mais claro.

7.8 Recursos Humanos

Projetos XP afetam as práticas de recursos humanos de uma organização, em particular no que se refere a contratações e avaliações periódicas dos funcionários. O fato de os programadores trabalharem em pares, por exemplo, leva à necessidade de pessoas que não apenas tenham boas habilidades técnicas, mas também saibam interagir socialmente com naturalidade. Portanto, a contratação não pode se basear apenas em critérios técnicos e as avaliações individuais se tornam complexas porque é difícil isolar o rendimento do trabalho individual quando se trabalha em par a maior parte do tempo.

7.9 Redatores Técnicos

Redatores técnicos ajudam a equipe a criar e manter a documentação do projeto. Um dos desafios é o ritmo semanal de desenvolvimento. Se a equipe tiver

que manter documentos extensos e muito elaborados, ela dificilmente será capaz de entregar um conjunto significativo de novas funcionalidades a cada iteração. Além disso, equipes XP acolhem mudanças como algo natural no processo de desenvolvimento, o que afeta a documentação e pode fazer com que ela tenha de ser alterada freqüentemente.

Redatores técnicos asseguram que a documentação evolua de forma iterativa. Ao invés de investirem em documentar o projeto, extensivamente, desde o início de cada iteração, redatores técnicos atualizam os documentos mais perto do fim das iterações. Dessa forma, eles descrevem o que foi feito de fato pelos desenvolvedores, ao invés daquilo que eles combinaram que iriam fazer. Isso tende a gerar menos mudanças.

7.10 Usuários

Usuários em uma equipe XP ajudam a escrever e selecionar histórias e tomam decisões relativas ao domínio do negócio durante o desenvolvimento. Sua participação é tão valiosa quanto for seu conhecimento sobre o domínio do negócio. É ainda melhor quando os usuários têm experiência com sistemas semelhantes no passado.

Projetos XP procuram envolvê-los ativamente desde o início, como forma de obter feedback cedo e direcionar o desenvolvimento para a implementação das funcionalidades que mais puderem gerar valor imediato para a organização.

8 Práticas Primárias

As Práticas, em Extreme Programming, são basicamente as coisas que as equipes costumam fazer diariamente, como por exemplo: Programação em Par. Essas práticas sozinhas, ou seja, fazê-las só por fazer, não faz muito sentido e acaba sendo até mesmo frustrante para quem está fazendo. Já a Prática associada a algum propósito ou valor como simplificar o sistema, obter feedback ou capturar erros, isso sim faz todo sentido.

A utilização dessas práticas não é uma coisa puramente rotineira, uma coisa que se faz todos os dias independente do sistema que se está produzindo ou da situação que se está encarando. Há vários tipos de práticas que podem se encaixar nas diferentes situações enfrentadas no processo de desenvolvimento de softwares, e à medida que essas situações mudam, podemos selecionar quais as melhores práticas para se enfrentar a determinada situação e aí as incorporamos ao nosso processo de desenvolvimento.

As práticas usadas em Extreme Programming funcionam bem sozinhas, mas postas em ação em conjunto podem chegar a representar melhorias dramáticas no desenvolvimento. Isso porque a interação entre essas práticas tende a amplificar o efeito que as mesmas já possuem quando isoladas.

As práticas XP foram divididas em dois tipos básicos: as práticas primárias e as práticas corolárias. A diferença principal entre esses dois tipos de práticas é que as primárias, como o próprio nome já sugere, são as mais básicas e são úteis independente do que mais possamos estar fazendo, ou seja, vão gerar resultados benéficos usadas isoladamente e independente de qual se escolha usar. Já a utilização das práticas corolárias depende de que já se tenha alguma experiência com as práticas primárias, o que quer dizer que são práticas que devem ser usadas em conjunto com as primárias.

8.1 Ambiente Informativo

Essa prática de Extreme Programming tem como princípio básico a questão de que o ambiente onde está sendo feito o desenvolvimento deve ser um reflexo do projeto que está sendo implementado, ou seja, o ambiente tem que refletir o projeto de tal forma que qualquer um que entre na sala da equipe possa ter, em pouco tempo, ter uma noção básica, porém clara do que se trata e de como anda o projeto.

Para conseguir alcançar esse nível de reflexão as equipes XP se utilizam de instrumentos como, por exemplo, Cartões com histórias – que são como lembretes dos freqüentes diálogos com o cliente – que são colocados em murais na parede; Quadro branco, para que sejam feitas anotações rápidas ou explicações de idéias para o projeto ou qualquer outro tipo de apresentação; Gráficos colocados na parede com informações que sejam mais relevantes para a equipe; etc.

A utilização dessa prática XP gera um avanço considerável na comunicação de idéias entre os membros da equipe e os ajuda a ganhar tempo na execução de suas respectivas tarefas, pois todas as informações cruciais estão sempre ao alcance de todos. Vale lembrar também que obviamente essas informações devem estar sempre sendo atualizadas conforme a evolução do projeto.

8.2 Build de Dez Minutos

A prática do Build de Dez Minutos diz que é recomendável que se tente fazer o Build – que nada mais é do que a conversão do código de programação em

um programa executável – e realizar todos os testes automatizados – que servem para antecipar a detecção e correção de falhas no sistema – do projeto em um tempo máximo de dez minutos. É importante que se consiga essa rapidez na execução dos Builds, pois se essa execução se tornar algo demorado, é muito provável que a equipe deixe de fazê-lo com a frequência necessária. A principal importância da execução frequente dos Builds é que eles são uma boa oportunidade que a equipe tem para verificar o feedback sobre o funcionamento do sistema em desenvolvimento, conseqüentemente vão descobrir mais rapidamente alguma falha que esteja ocorrendo, e é sabido por todos que quanto mais tempo se demora a descobrir as falhas do sistema, mais caro se torna pra consertá-las.

8.3 Ciclo Semanal

A prática de Ciclo Semanal vem da idéia de se ter um desenvolvimento de software que seja iterativo e incremental, ou seja, onde se cumpram pequenos ciclos de desenvolvimento e a cada ciclo desse, se possa ter uma evolução notável do software. Nessa prática, os desenvolvedores procuram o cliente para que seja feita uma reunião semanal para se discutir e separar uma pequena quantidade de funcionalidades do sistema que será implementada e testada naquela semana, e depois disso, será entregue ao cliente para que este as utilize e avalie. Isso é feito toda semana, e a partir dos resultados obtidos na semana através daquelas funcionalidades, tem-se nova pauta para que seja discutida em nova reunião com o cliente.

8.4 Ciclo Trimestral

Essa prática tem as mesmas idéias e objetivos da prática anterior (Ciclo Semanal), com a diferença óbvia de tempo estimado de cada ciclo. Dessa vez o planejamento de desenvolvimento do software é dividido em trimestres, ou seja, as reuniões agora acontecem a cada começo de trimestre com o objetivo de discutir tudo o que foi feito no último trimestre e quais foram os resultados obtidos, e também para definir quais serão as prioridades a se implementar no período dos próximos três meses.

8.5 Desenvolvimento Orientado a Testes

Essa prática bate na tecla de um aspecto do desenvolvimento de softwares que é bastante discutido, mas geralmente é deixado de lado em alguns projetos, que é a aplicação de testes nos sistemas em implementação.

Resultados de pesquisas feitas nos EUA têm mostrado que há um prejuízo anual estimado de aproximadamente 60 Bilhões de dólares na economia americana causado por falhas de software, o que significa que essas falhas têm sido muito comuns e danosas. Essas pesquisas revelaram também que aproximadamente mais de um terço desses custos causados por falhas poderia ser eliminado através da utilização de uma infra-estrutura de testes bem mais robusta e eficiente, que pudesse permitir a identificação e remoção das falhas nos sistemas bem mais cedo. Calcula-se atualmente que cerca de metade dos erros e falhas sistemas só são identificados em fases finais dos projetos, ou até depois de serem colocados em produção.

8.6 Design Incremental

Novamente se fala em desenvolvimento iterativo e incremental em projetos XP, cujo objetivo principal é a criação de uma solução que seja o mais simples possível e que seja suficiente para a implementação das funcionalidades de cada iteração. A equipe só se preocupa com as características e funcionalidades que foram separadas e classificadas como prioridade para a iteração atual, qualquer característica que possa ser implementada para dar apoio a funcionalidades futuras, só são codificadas de fato se e quando tais funcionalidades forem priorizadas para uma iteração futura.

8.7 Equipe Integral

Uma equipe XP não é formada apenas pelos analistas e desenvolvedores, mas sim também pelos clientes e todas as outras pessoas que devam ser ouvidas e que acrescentam algo importante ao desenvolvimento do projeto, pois o sucesso de um projeto XP leva em conta opiniões diversificadas e também diferentes pontos de vista.

A prática de Equipe Integral não prega que os clientes devam passar todo o tempo de implementação junto com a equipe, até porque sabe-se que isso não é possível nem viável, mas o que se quer dizer é que o cliente e todos que estejam de alguma forma envolvidos no projeto tenham a maior disponibilidade possível para ajudar a equipe sempre que surgirem dúvidas sobre o projeto.

Essa proximidade entre equipe e cliente é muito importante para a

comunicação no XP, pois a comunicação é imprescindível para que a equipe fique totalmente por dentro de todas as idéias do cliente para o projeto e assim evitar transtornos futuros na implementação.

8.8 Folga

As equipes comuns de desenvolvimento de software normalmente trabalham com prazos apertadíssimos e acabam por ocorrer atrasos no processo de implementação dos projetos. Isso é ruim pois se a equipe não consegue cumprir os prazos estabelecidos, os clientes tendem a perder a confiança nela, o que pode acabar gerando diversos desperdícios como pressões de tempo maiores, baixa motivação e relacionamentos conflituosos.

A prática de Folga em XP diz que as equipes devem estabelecer um certo nível de folga em todos os seus planejamentos. Isso pode fazer com que a equipe não atrase seus prazos junto ao cliente por conta de imprevistos ocorridos no processo de desenvolvimento.

8.9 Integração Contínua

Em uma equipe com vários desenvolvedores, todos trabalhando na elaboração de um mesmo sistema, qual a melhor forma de unificar as diversas alterações feitas na base de código? Processos ágeis como o Extreme Programming utilizam a prática conhecida como Integração Contínua para solucionar essa questão.

Integração contínua consiste em integrar o trabalho diversas vezes ao dia,

assegurando que a base de código permaneça consistente ao final de cada integração. Nesse artigo, você conhecerá os passos necessários para usar essa prática em seu dia-a-dia, primeiro de forma manual, com foco nos conceitos, e depois de forma automatizada.

8.10 Programação em Par

A prática de programação em par é uma das mais conhecidas pelas equipes de Extreme Programming. Ela sugere que toda a implementação de código feita no projeto seja sempre feita por duas pessoas no mesmo computador, se revezando no teclado.

Por se utilizar de duas pessoas em cada máquina, sendo que esta só pode ser usada por uma pessoa de cada vez, essa prática acaba gerando polêmica e parece ser fadada ao fracasso, mesmo que se possa notar claramente os benefícios.

Benefícios como, por exemplo, a maior eficiência na detecção de bugs em tempo de implementação. Isso se deve principalmente pelas diferentes visões que os desenvolvedores têm durante a implementação com o uso dessa prática. Quando se utiliza a prática de programação em par, um dos desenvolvedores está “com a mão na massa”, ou seja, operando teclado e mouse, enquanto o outro está sentado ao lado e olhando com mais atenção o código que está sendo produzido. Assim eles estão trabalhando juntos na mesma solução e conversam o tempo todo discutindo e trocando idéias sobre a implementação. O desenvolvedor que está apenas observando tem o importante papel de estar sempre corrigindo o código.

8.11 Sentar-se Junto

Essa prática diz basicamente se deve evitar o trabalho em cubículos lugares pequenos e apertados. Membros de uma equipe XP devem procurar se sentar juntos em uma sala aberta, na qual todos possam trabalhar em conjunto e se comunicar da forma mais rápida e eficaz possível.

Mas apesar disso sabemos que os desenvolvedores também precisam de privacidade pra resolver problemas particulares que possam vir a surgir. É possível atender essa necessidade com pequenos cubículos próximos à área comum da equipe, ou limitando as horas de trabalho.

A utilização dessa prática no processo de desenvolvimento não significa que um projeto XP não possa também ser distribuído, mas apenas mostra que é necessário que os membros da equipe busquem sempre que possível um contato entre si para que possa fluir uma melhor comunicação.

8.12 Trabalho Energizado

Projetos XP seguem a filosofia de que o mais importante não é trabalhar mais e sim trabalhar de forma mais inteligente, em um período de tempo semanal que as pessoas sejam capazes de sustentar sem ficarem esgotadas e sem prejudicarem o trabalho com o déficit de atenção decorrente da fadiga. Existe uma distinção importante entre movimento e progresso. Uma equipe que se movimenta muito, que trabalha muitas horas, que "dá um gás" pelo projeto, mas produz várias coisas erradas não está progredindo. Está apenas consumindo energia. O importante não é se movimentar muito, e sim se mover na direção certa, de forma

inteligente e sustentável.

9 Práticas Corolárias

Antes de implementar as práticas corolárias deve-se adotar primeiramente as práticas primárias para tornar mais fácil e segura sua implementação. Confie na sua intuição sobre o que é a próxima coisa que precisa ser aprimorada. As práticas a seguir precisam ser analisadas para saber se é apropriada ou não, se for tente-a senão há mais trabalho a ser feito antes de utilizá-las para aprimorar o processo de desenvolvimento.

9.1 Análise da Raiz do Problema

A técnica dos cinco porquês, sugerida por Taiichi Ohno, pai do Sistema de Produção da Toyota, consiste em diante de um problema, perguntamos por que ele ocorreu cinco vezes, para assim, conseguimos passar do sintoma à raiz do problema.

Resolver o sintoma não soluciona o problema, na maioria dos casos. É preciso identificar as verdadeiras causas para dessa forma, corrigi-lo.

No desenvolvimento de software também se deve utilizar essa técnica. Quando encontramos um erro no software, em vez de corrigi-lo com a primeira idéia que vier na cabeça, é melhor nos perguntarmos alguns porquês.

Para lidar com um defeito, escreva um teste funcional que demonstre o defeito, incluindo o comportamento desejável, em seguida, escreva um teste de

unidade com o menor escopo possível que também seja capaz de reproduzir o defeito, corrija o sistema, de modo que o teste de unidade passe a executar com sucesso, isso deve fazer com que o teste funcional também passe com sucesso, se não, refaça o teste de unidade. Uma vez que o defeito tenha sido corrigido, tente descobrir por que o defeito foi criado e não foi detectado antes. Inicie as mudanças necessárias para evitar que este tipo de defeito volte a ocorrer no futuro. Envolve a equipe nesse processo e dissemine a informação sobre esse defeito e o que pode ser feito para ele não acontecer novamente no futuro.

A análise da raiz de um problema ajuda a corrigir e prevenir problemas, buscando onde o problema começa e como trabalhar para que ele não mais aconteça, pois apenas corrigir os sintomas sem tratar as causas reais geralmente acarreta na volta dos sintomas.

9.2 Base de Código Unificada

O ideal é ter apenas uma base de código. Pode-se até desenvolver temporariamente em outra base, porém depois de algumas horas, deve-se desfazer dela. Múltiplas bases de código causam problemas na hora da manutenção, pois para cada problema que ocorrer deve-se fazer a alteração em todas as versões do software, e esta alteração poderá trazer erros nas outras bases.

Existem motivos válidos para se manter múltiplas versões do código ativas ao mesmo tempo. Mas em muitos casos, o que está em andamento é simplesmente uma pequena melhoria levada adiante sem se preocupar com as conseqüências que podem ocorrer.

Tendo várias versões de código, faça um planejamento de como reduzi-

las gradativamente. Melhore o sistema de build, pois assim criará vários produtos a partir de uma única base de código. Verifique o se altera de uma versão para outra e coloque em arquivos de configuração. Ou seja, aprimore o seu processo até que você não precise mais ter múltiplas bases do código.

Havendo motivos válidos para ter múltiplas versões de código fonte, olhe para esses como sendo proposições a serem desafiadas, e não como verdades definitivas. Desfazer-se dessas proposições, pode levar tempo, mas desfazendo talvez abra a porta para a próxima rodada de melhorias.

9.3 Código Coletivo

A propriedade do código é coletiva, todos têm acesso e autorização para editar qualquer parte do código do software, a qualquer hora e a responsabilidade de todas as partes do código é dividida igualmente para todos. Com isso, o desenvolvedor ganha tempo, pois não precisa de autorização para editar uma área do código e há maior disseminação de conhecimento. Quando várias pessoas olham uma mesma base de código, fica mais fácil identificar erros e apontar melhorias.

9.4 Código e Testes

"Mantenha apenas código e testes como artefatos permanentes. Gere outros documentos que se façam necessários a partir do código e dos testes. Utilize mecanismos sociais para manter viva a história do projeto". (TELES, 2006).

Os artefatos que contribuírem para que o sistema faça hoje e o que a equipe pode fazer com que o sistema faça futuramente, são valiosos, o restante é

desperdício.

Aos poucos essa técnica é fácil de ser entendida a medida que a equipe se torna mais habilidosa. Se a equipe for boa em design incremental, menos decisões sobre estes precisarão ser tomadas com antecedência. Se o ciclo trimestral se tornar mais claro, em termos de expressar prioridades de negócio, o documento de requisitos será mais fino.

Ultimamente, os softwares vêm sendo desenvolvidos de maneira oposta a está prática, pois o formalismo interfere no fluxo de valor. Para suavizar este fluxo de valor, devemos desfazer dos artefatos que estão atualmente obsoletos. Devemos decidir o que iremos fazer o que não iremos fazer e como iremos fazer aquilo que vamos fazer, de modo que essas decisões se alimentem umas das outras para suavizar o fluxo de valor.

9.5 Continuidade da Equipe

A maioria das grandes empresas abstrai as pessoas ao invés das coisas. O valor do software está também no relacionamento das pessoas e pelo que as pessoas são capazes de alcançar unidas, e não só pelo que as pessoas sabem e fazem. Não ignore o valor dos relacionamentos e da confiança para simplificar o problema de agendamento, pois assim terá uma falsa economia.

Já as pequenas empresas não possuem esse problema, já que tem apenas uma equipe. Com essa integração o administrador ganhará e passará uma forte confiança da equipe. Nas grandes empresas, quando o projeto encerra-se, o programador já recebe outro projeto, com o objetivo de fazer com que todos sejam utilizados integralmente. Essa estratégia de agendamento nada mais é do que uma

eficiência ilusória onde os programadores ficam ocupados digitando, mas o valor é ignorado, pois impede que os indivíduos trabalhem na maior parte do tempo com as pessoas que conhecem e confiam.

Integrar equipes não significa dizer que as equipes ficarão completamente estáticas. Novos membros contribuem rapidamente em um projeto XP, pois implementam tarefas de desenvolvimento, já nas primeiras dias e após alguns meses contribuem de maneira independente. “Preservando equipes bem integradas e encorajando um nível razoável de rotação, a organização ganha os benefícios de um equipe estável e consistência em termos de dispersão de conhecimento e experiência”. (TELES, 2006)

9.6 Contrato de escopo negociável

Como qualquer outro projeto na área de software, projetos XP, possuem um escopo que deve existir antes do projeto iniciar e continuar existindo até o fim do projeto, este escopo define o que deve ser feito no projeto. Este escopo deve ser fixado em contrato, mas normalmente isto não é feito, e com isto possibilita que o cliente faça ajustes nele para que o programa leve em conta seu aprendizado ao longo do projeto, ou faça mudanças nas circunstâncias. O escopo deve ser revisado freqüentemente para que a cada etapa do projeto garanta que a equipe se esforçará no que for mais prioritário.

Para lidar com a não fixação do escopo no contrato utiliza-se a previsibilidade que faz com que o cliente tenha a ilusão de que contará com custo previsível, prazo previsível e escopo previsível, ou seja, ele terá uma previsão do que irá receber, quando irá receber, e quanto vai gastar com o projeto. Do lado da

empresa, ela terá perspectivas interessantes como, receita previsível, prazo previsível, demanda previsível, ou seja, ela terá uma previsão do que irá fazer quanto vai ganhar e quando terá folga para colocar os programadores em outro projeto.

Ambos aparentemente estão em situação confortável, mas o problema está na premissa da previsibilidade, pois o problema do cliente geralmente se mantém estável, mas o escopo da solução pode e normalmente é alterado ao longo do projeto, por isso o escopo não é previsível, portanto não se deve fixá-lo freqüentemente. Aprender e aprimorar a solução pode trazer economia e tempo tanto para o cliente, quanto para a empresa.

O cliente geralmente não conhece todos os detalhes que software precisa ter, dificultando a estimativa de escopo perfeito. Mesmo o cliente contando todos os detalhes, a equipe pode compreender incorretamente as especificações do projeto, acarretando numa estimativa de escopo errada.

Resumindo, previsibilidade é inviável na maioria dos casos, uma ilusão na área de software, onde o cliente e a empresa fingem acreditar naquilo que está sendo proposto.

Optando por um escopo fixo, corre-se o risco, bem alto, de que o programa final não atenda às reais necessidades do cliente, e não é só isto, segundo estatísticas. Mais de 60% das funcionalidades programa nunca são usadas quando colocadas em produção, ou seja, 60% do investimento do cliente será perdido.

Para administrar bem um projeto de software deve-se rever as prioridades do cliente para que apenas as funcionalidades que serão realmente utilizadas, sejam colocadas no software. Em XP o desenvolvimento é iterativo possibilitando que

essas funcionalidades sejam descobertas ao longo do projeto, pois a medida que o cliente vai interagindo com o sistema, ele vai descobrindo as funcionalidades que ele quer que o software possua. Resumindo, fixar um escopo fixo no início do projeto é prejudicial para o cliente.

Escopo negociável é um contrato que supõe que não existirá previsibilidade do que será feito no programa. Porém pode-se os gastos e o tempo podem ser previstos. Ou seja, é uma maneira de equilibrar os interesses do cliente e da empresa.

O contrato de escopo fixo determina o custo, prazo, escopo e qualidade. Geralmente a qualidade acaba sendo sacrificada devido ao prazo mal estipulado. No contrato de escopo negociável o cliente continua sabendo quanto irá gastar e quando irá receber o software, mas não saberá com exatidão o que irá receber, isto também acontece no escopo fixo, a diferença é que neste o cliente tem a ilusão de que sabe o que irá receber. Resumindo o cliente não irá perder nada apenas receberá um contrato mais honesto e realista.

Essa filosofia é boa para o cliente, pois no final do projeto ele terá um software que atenderá suas necessidades e prioridades reais, e não um com funcionalidades que ele achava que o software deveria possuir no início do projeto.

No contrato de escopo fixo normalmente o cliente só descobre se fez uma boa escolha da equipe após o projeto ter tido avanço grande, já que o código demora a ser implementado, portanto o feedback demora a aparecer. No XP, o cliente recebe funcionalidades prontas já na primeira semana, na semana seguinte receberá outras novas funcionalidades e assim sucessivamente até o final do projeto, com isso o cliente decidirá melhor se deve ou não continuar com a equipe.

O contrato de escopo negociável deverá estimar o tempo necessário e o

número de integrantes da equipe, antes do projeto começar, fazendo um levantamento das funcionalidades, a partir de conversas entre o cliente e a empresa. O custo do projeto será proporcional ao tempo necessário e o número de integrantes da equipe. Não adianta buscar previsibilidade e estimativa perfeita, ao invés disto deve-se buscar valores razoáveis para o tempo, custo e número de pessoas. XP procura evitar que o erro demore a ser descoberto, através das iterações semanais. Com esta filosofia, o cliente terá opções de saída, ou seja, poderá cancelar o contrato sem ter que pagar multas contratuais.

A equipe só tem garantia de que irá faturar no tempo inicial do contrato, pois ao final deste o cliente poderá cancelar o contrato se não estiver satisfeito. Por outro lado a equipe terá interesse em continuar até o final do projeto para que seu faturamento aumente, portanto terá razões para trabalhar bem e com agilidade, para que o cliente renove o contrato. Entretanto ela não tem nenhuma razão para não aceitar as mudanças exigida pelo cliente ao longo do projeto, pois não está fixado no escopo e o pagamento não está vinculado a ele. Com isto o cliente poderá fazer alterações nas funcionalidades do software sem atrapalhar a equipe de cumprir o contrato. Ou seja, todos irão querer que o software atenda da melhor maneira possível às necessidades do cliente, para que ambos sejam beneficiados.

No contrato de escopo fixo, alterações sugeridas pelo cliente ao longo do projeto costumam ser rejeitadas pela equipe ou cobradas a preços elevados, pois essas alterações interferem na capacidade da equipe cumprir o contrato. Esse tipo de contrato faz com que o cliente e a equipe se tornem adversários, onde o primeiro deseja ter funcionalidades a mais, e o segundo menos esforços e funcionalidades. Pra completar, esses contratos são mais caros porque a empresa já prever que o cliente vai querer fazer alterações no escopo e, por isso, cobra um valor acima do

seu custo real para cobrir os custos que surgirão com essas alterações.

No modelo de contrato de escopo negociável se o cliente não estiver satisfeito, ele poderá interromper o contrato perdendo apenas tempo e o valor investido neste intervalo de tempo. No modelo de contrato de escopo fixo, o cliente só percebe que não está satisfeito perto do fim do projeto, onde o custo para interromper o contrato é bem maior.

Quanto mais confiança o cliente tiver na empresa, menor a probabilidade dele interromper o contrato, portanto deve-se criar um histórico de credibilidade e aproximar ao máximo o cliente com a equipe. O cliente deve participar do desenvolvimento, para conhecer melhor o trabalho, o ritmo, as deficiências, os pontos fortes, o empenho e o comprometimento da equipe. Ou seja, se a equipe estiver realmente dedicada a trabalhar bem, o cliente conseguirá notar isso. Quando a equipe cumpre o contrato a cada iteração entregando as funcionalidades acordadas, cria-se uma relação de confiança com o cliente.

O custo no modelo de contrato de escopo negociável é menor do que o de escopo fixo, pois o escopo não está vinculado ao contrato, portanto, elimina-se o risco da empresa deixar de cumprir o contrato por interpretação equivocada da equipe ou por alterações feitas pelo cliente no escopo ao longo do projeto.

Para adotar o modelo de contrato de escopo negociável, deve-se compreendê-lo e propô-lo aos clientes. Pode ser que não seja fácil, mas poderá reduzir custos, riscos, e melhorar a relação com o cliente. Esse tipo de contrato representa uma mudança cultural, para adotá-lo deve-se apresentar todas as questões abordadas anteriormente, em especial a falta de previsibilidade e o aprendizado do cliente no decorrer do projeto. Deve-se apresentar o custo do contrato de escopo negociável comparando com o do escopo fixo para convencer o

cliente que ele de mudar o modelo de contrato. Se o cliente perceber que deve alterá-lo fica mais fácil implementar o modelo de contrato de escopo negociável.

9.7 Envolvimento do Cliente Real

No XP, desenvolvedores tomam decisões técnicas, já o pessoal de negócios, que em XP são chamados de requerentes, tomam decisões de negócio. Ou seja, deve-se colocar um ou mais requerentes, para definir e priorizar as funcionalidades que deverão ser implementadas.

"A prática equipe integral sugere que haja requerentes participativos na equipe, representando os mais diversos pontos de vistas dos usuários. Mas, isso não é suficiente. A prática de envolvimento do cliente real vai além. Ela sugere que os usuários finais sejam envolvidos diretamente no processo de desenvolvimento". (TELES, 2006)

No decorrer do projeto, podemos trazer usuários finais para testar as funcionalidades que já foram implementadas, para saber se o que está sendo pedido pelos requerentes realmente reflete as necessidades reais do usuário final e também para testar a usabilidade do software.

9.8 Equipes que Encolhem

Em Xp, deve-se liberar as pessoas da equipe para que se forme novas equipes. Se tiver equipes com poucas pessoas, misture-as. Ao invés de fazer com que todas as pessoas tenham a mesma carga de trabalho, deve-se assegurar que o máximo de integrantes da equipe esteja trabalhando de forma integral. Com isso

eles trocam idéias até que alguém da equipe fique ocioso e seja liberado.

9.9 Implantação Diária

Todos os dias, deve-se colocar novas funcionalidades no software, pois qualquer diferença entre o código que está no computador do desenvolvedor e o código que está em produção é perigosa. Se o programador não estiver em sintonia como o software em produção, poderá tomar decisões sem receber feedback preciso sobre suas decisões.

"Implantação diária é uma prática corolária porque tem inúmeros pré-requisitos. A taxa de defeitos tem que ser da ordem de apenas um punhado por ano. O processo de build tem que ser bem automatizado. As ferramentas de deploy precisam ser automatizadas, incluindo a capacidade de colocar em produção de forma incremental e voltar atrás (roll back) em caso de falhas. Sobretudo, a confiança entre a equipe e os clientes precisa estar altamente desenvolvida." (TELES, 2006)

Atualmente pode-se observar que várias aplicações estão sendo atualizadas freqüentemente a cada poucos dias. A implantação diária vai além desta tendência. Portanto se a empresa no momento atual não consegue atualizar seu software mais que uma vez por ano, será difícil conseguir implantar essa prática.

Normalmente o desenvolvedor consegue implementar novas funcionalidades ao software mas ao invés de lançar uma nova versão ele cria correções (patches), pois vêem isso como algo vergonhoso ao invés de oportunidade.

Para efetuar o deploy existem barreiras técnicas como a quantidade de

defeitos e o custo para fazê-lo, e também existem barreiras psicológicas ou sociais como um processo de implantação tão estressante que faz com que as pessoas não queiram fazer atualizações. Essas barreiras devem ser eliminadas para que as implantações diárias venham naturalmente melhorando o desenvolvimento.

9.10 Implantação Incremental

As partes do sistema devem ser substituídas gradativamente desde o início do projeto. Normalmente são feitas grandes implementações onde os desenvolvedores passam dias programando até que decidam colocar a implementação em produção. Poderá dar certo, mas a equipe ficará cansada e dificilmente irá manter o ritmo de desenvolvimento, e se não der certo o sistema sairá de produção e os prejuízos serão maiores ainda.

Portanto, as funcionalidades ou conjunto de dados que poderem ser implementadas no momento devem ser implementadas.

9.11 Pagar por uso

Deve-se cobrar por cada vez que o sistema for utilizado, pois conectar o fluxo de dinheiro ao desenvolvimento de software provê informação precisa e em tempo que pode ajudar a aprimorar o processo de desenvolvimento de software.

O que ocorre normalmente é a cobrança por cada release do software. Esta prática coloca em oposição os interesses da empresa com os do cliente. O primeiro quer lançar vários releases contendo uma quantidade mínima de funcionalidades necessárias para que o segundo pague por elas, estes por sua vez,

desejam menos releases com mais funcionalidades cada. Esta oposição causa redução de comunicação e feedback.

Não podendo implementar pay-per-use, utilize um modelo de subscrição onde o software é comprado mensalmente ou trimestralmente, pois assim a empresa pelo menos terá informação da quantidade de clientes que continuam subscritos para saber como está o andamento da equipe. Outra solução seria colocar no contrato, um valor de pagamento inicial mais baixo e taxas de suporte mais altas.

Clientes desejam custos previsíveis, portanto se o preço do pay-per-use for baixo o suficiente talvez ele não se importe em utilizar este modelo. Se a equipe souber utilizar a informação gerada pelo modelo pay-per-use ela poderá fazer trabalhos mais eficazes que uma equipe que utiliza feedback apenas com informações geradas pelas receitas de licença.

10 Outras Práticas

10.1 Reunião em Pé

Reunião em pé é uma breve reunião realizada diariamente, normalmente de manhã, pela equipe de desenvolvimento com o objetivo de compartilhar informações sobre o projeto e priorizar suas atividades. Trata-se de um diálogo entre todos os membros da equipe, se possível envolvendo também a presença do cliente. Em qualquer diálogo presencial, existem pelo menos duas coisas que são compartilhadas de maneira bastante rica: informações e emoções.

10.2 Informações

Cada membro da equipe tem acesso a toda e qualquer parte do código e, acima de tudo, tem o direito de alterar qualquer parte do sistema, a qualquer momento, sem ter que pedir permissão a ninguém. Essa é uma prática conhecida como Código Coletivo. A parte boa dessa prática é que o desenvolvedor consegue avançar muito rapidamente, pois nunca fica dependendo de outra pessoa, ou da autorização de alguém, para editar uma parte do código. Por outro lado, isso exige muita troca de informações, visto que tudo o que está acontecendo, em qualquer parte do sistema, interessa a todos os membros da equipe.

O diálogo diário, realizado nessa reunião, permite que cada desenvolvedor descreva brevemente o que fez no dia anterior, eventuais problemas que detectou, soluções interessantes que foram criadas etc. A idéia é que as pessoas saibam o que está acontecendo e quem fez o que. Se um desenvolvedor se interessar bastante por um assunto a respeito do qual outra pessoa trabalhou no dia anterior, ele pode, após a reunião, conversar com essa pessoa e discutir a solução com maior nível de detalhe. Pode até resolver fazer par com aquela pessoa durante o dia de trabalho que está começando.

À primeira vista, essa reunião parece ser demorada. Afinal, se cada desenvolvedor tiver que explicar tudo o que fez no dia anterior, certamente isso pode consumir muito tempo. Entretanto, a idéia é fazê-lo diariamente e há várias razões importantes para ser assim. A primeira é que um dia de trabalho é um período relativamente curto. Descrever o que aconteceu em um dia é diferente de descrever os acontecimentos de uma semana, um mês ou um ano. Um dia é muito pouco

tempo e normalmente não se produz uma quantidade enorme de coisas interessantes em um único dia. Sendo assim, existe a tendência de que cada desenvolvedor tenha pouco a dizer se essa reunião realmente for conduzida com frequência diária. Apesar de pouco, o que ele tiver a dizer interessa a todos, porque o código é coletivo. De tempos em tempos, haverá alguma coisa realmente muito significativa que tenha sido feita por um par no dia anterior. Nestes casos, a reunião pode acabar tomando mais tempo, porém o valor deste tempo é mais alto devido à utilidade da informação que se está transmitindo.

Se alguma coisa muito grave é detectada nessa reunião, o gerente do projeto e o próprio cliente ficam sabendo rapidamente. Pois qualquer problema terá sido detectado há no máximo um dia de trabalho. Assim, o gerente, o cliente e os desenvolvedores podem criar soluções enquanto esse problema ainda não deu origem a outros.

A reunião em pé também é um mecanismo para expressar e treinar a coragem dos membros da equipe. Em muitas empresas, os desenvolvedores escondem problemas potenciais com medo de sofrerem uma punição do chefe, do cliente ou de qualquer pessoa com poder na organização. De fato, muitas organizações são regidas pela cultura do medo. A reunião em pé é um espaço aberto onde as pessoas são incentivadas a falar tudo o que está acontecendo e são valorizadas por fazer isso.

Finalmente, equipes XP costumam utilizar radiadores de informações como o Quadro de Acompanhamento Diário. Trata-se de uma tabela, desenhada em um quadro branco, contendo informações sobre todas as histórias da iteração. Nela, colocam-se as estimativas de cada história, quanto tempo foi gasto em cada uma por dia da iteração, que tarefas extras foram executadas etc. É um quadro

importante porque gera muita visibilidade para todos os membros da equipe, incluindo o cliente. Com esse quadro, não há necessidade de perguntar nada para o gerente para saber o estado do projeto em um iteração. Basta olhar para a parede. As informações estão lá. Assim, nenhum desenvolvedor precisa preencher uma folhinha dizendo quantas horas trabalhou por dia, por exemplo.

10.3 Emoções

É possível transmitir informações através de ferramentas. Porém, emoções também informam e estas são muito difíceis de serem compartilhadas através de ferramentas. Durante uma reunião em pé, os membros da equipe conseguem avaliar e sentir como está o clima da equipe. Olhando para cada pessoa, é possível observar se estão satisfeitas com o trabalho que está em andamento, dá para ver se estão cansadas ou energizadas, se estão entediadas ou empolgadas, se estão de acordo com os rumos do projeto etc. Muitas informações importantes jamais são verbalizadas, mas freqüentemente basta você olhar para a expressão facial de uma pessoa para saber que ela discorda plenamente de alguma coisa ou para notar que ela está cansada, ou aborrecida. Nestes momentos, um bom coach (bem como qualquer membro da equipe) deve atuar pedindo à pessoa que expresse seus sentimentos. Muitas vezes vivenciamos grandes viradas em projetos, viradas extremamente positivas, porque alguém notou um problema na expressão de um dos desenvolvedores.

Para finalizar, precisamos compreender que a reunião em pé é uma forma de priorizar o que será feito em cada dia de trabalho. Nós priorizamos para assegurar que estamos fazendo o que é mais importante a cada momento do projeto

e isso é feito porque o maior objetivo do XP é entregar um fluxo constante de valor para o cliente. Isso só é possível quando planejamos cuidadosamente o que será feito, isto é, quando escolhemos a cada dia fazer o que mais tenha potencial de gerar valor naquele momento. Sem essa priorização diária, é fácil um desenvolvedor acabar gastando tempo com algo que pode até ser útil, mas não era a coisa mais importante naquele ponto do projeto.

Em princípio, é fácil decidir o que deve ser feito a cada dia de trabalho. Basta olhar para o mural, onde a equipe coloca os cartões contendo as histórias da iteração, e verificar que histórias ainda precisam ser finalizadas. Entretanto, às vezes surgem problemas inesperados no dia anterior. Nesses casos, a equipe usa essa reunião para decidir se esses problemas devem ser tratados já no dia que está sendo iniciado, se devem ser deixados para uma iteração futura etc.

A priorização em equipe só pode ser feita de forma adequada quando temos a noção do todo, isto é, quando sabemos o que está acontecendo em cada parte do projeto, com cada pessoa da equipe e não apenas conosco. Pois, algo que eu possa considerar muito sério e prioritário no momento, pode ser absolutamente irrelevante quando somos informados de um problema bem mais significativo que está acontecendo em outra parte do projeto. Nos dias em que isso acontece, talvez faça mais sentido deixar o meu problema de lado e ir ajudar outra pessoa a resolver o problema mais sério que está afetando toda a equipe. Sem stand up meeting diário, é mais difícil notar e atuar rapidamente sobre situações como essas.

10.4 Refatoração

Quando deixamos de cuidar de nossa casa, a poeira se acumula, a pia

fica cheia de louça suja, as lixeiras ficam entupidas e aquilo que se costumava chamar de lar se transforma em um lugar pouco agradável. Para evitar que isso ocorra, investimos tempo freqüentemente para varrer, lavar a louça, arrumar a cama, retirar o lixo, entre outras atividades. Isso nos ajuda a ter mais conforto e permite que utilizemos a nossa casa para o que desejarmos, como por exemplo, convidar amigos para assistir a um jogo conosco.

Assim como a nossa casa, sistemas também se deterioram quando não investimos continuamente na limpeza e na organização dos mesmos. A estrutura de qualquer sistema tende a se degradar ao longo do tempo, à medida que novas funcionalidades são inseridas, alterações são feitas, erros são corrigidos e mais código é introduzido. Para evitar que a aplicação se transforme em uma casa suja, desorganizada e difícil de manter, equipes XP utilizam a prática de refatoração. Elas alteram pequenas partes do sistema, freqüentemente, sempre que encontram uma oportunidade para melhorar o código, tornando-o mais limpo, mais claro e mais fácil de ser compreendido. Tais alterações não mudam o comportamento das funcionalidades, apenas melhoram a estrutura do código. Agindo assim de forma sistemática e com freqüência, as equipes investem para que o software se mantenha sempre fácil de alterar, gerando a velocidade de desenvolvimento que os clientes de projetos XP costumam vivenciar e apreciar.

10.5 Metáfora

Metáforas são usadas frequentemente durante o desenvolvimento de sistemas, na medida em que os desenvolvedores criam elementos dentro do computador para simular outros que existem regularmente fora dele, no mundo

físico. A lixeira, a mesa de trabalho, janelas, pastas e outros itens que estamos habituados a encontrar no computador, simulam elementos do mundo físico e seus respectivos comportamentos. XP procura explorar ao máximo a utilização de metáforas, para que clientes e desenvolvedores sejam capazes de estabelecer um vocabulário apropriado para o projeto, repleto de nomes representando elementos físicos com os quais os clientes estejam habituados em seu dia-a-dia, de modo a elevar a compreensão mútua.

CONCLUSÃO

De todos os padrões de projetos o XP é o único que é 100% voltado para projetos de software, enquanto os outros tem uma visão ampla, tentando ser útil para todos os tipos de projetos que podem vir a surgir. Deste modo, para nós desenvolvedores é muito mais simples de entender e de por em pratica um modelo que foi feito por desenvolvedores, e na medida para desenvolvedores do que outros modelos que somente foram adaptados para nós.

O que temos que ter sempre em mente é que a estrutura atual da organização acarreta um processo de reformulação e modernização dos índices pretendidos. Evidentemente, o comprometimento entre as equipes nos obriga à análise de alternativas às soluções ortodoxas.

O XP é essa alternativa, pois propõe uma mudança radical na forma de encarmos o projeto. Em vez de focar nas etapas do desenvolvimento do projeto, ele tenta fazer com que as atitudes e comportamento dos envolvidos no mesmo sejam diferentes, eliminando etapas consideradas desnecessárias e aumentando a produtividade por hora trabalhada.

Em vez de documentar no papel cada mudança que foi idealizada ou realizada durante o decorrer do projeto, eles propõem que o código seja a documentação, onde ele seja auto-explicativo, e o que não for possível descrever a traves de código seja realizado nos comentários, sendo que estes devem ser reduzidos ao máximo possível, para evitar a poluição desnecessária de código.

É claro que a necessidade de renovação processual agrega valor ao estabelecimento das novas proposições. Do mesmo modo, o novo modelo estrutural aqui preconizado pode nos levar a considerar a reestruturação do fluxo de

informações. A complexidade dos estudos efetuados promove a alavancagem das diretrizes de desenvolvimento para o futuro. Pensando mais a longo prazo, o acompanhamento das características deste modelo possibilita uma melhor visão global das posturas dos órgãos dirigentes com relação às suas atribuições.

REFERÊNCIAS

OSHIRO, Adriane; NOVELLI, Andreia; et all. **Extreme Programming, um novo modelo de processo para o desenvolvimento de software**. Disponível em: <http://www.dc.ufscar.br/~rosangel/mds/Aula-09-XP/ArtigoXP.pdf>. Acesso em: 26/05/2007.

CAVALCANTI, Ricardo. **O TOYOTA WAY E O DESENVOLVIMENTO ÁGIL DE SOFTWARE**. 2006. Disponível em: <http://www.cin.ufpe.br/~tg/2006-1/roc3.pdf>. Acesso em: 26/05/2007.

AMBROSI, Cleison Vander; GRAHL, Everaldo Artur. **Extreme Programming: Um modelo de processo para o desenvolvimento de software**. Blumenau ? SC: Instituto Catarinense de Pós-Graduação.

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. **Extreme Programming: Guia prático**. Rio de Janeiro ? RJ: Campus, 2002.

BECK, Kent. **Programação extrema (XP) explicada: acolha as mudanças**. Porto Alegre ? RS: Bookman, 2004.

POHREN, Daniel. **XP Manager: Uma Ferramenta de Gerência de Projetos Baseados em Extreme Programming**. Novo Hamburgo ? RS: Centro Universitário Feevale, 2004.

TELES, Vinícius Manhães. **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. São Paulo - SP: Novatec Editora Ltda, 2004. Disponível em: <http://www.improveit.com.br/xp/>. Acesso em: 26/05/2007.