UNIVERSIDADE DO SUL DE SANTA CATARINA

RODRIGO DOS SANTOS

ESTUDO DE CASO SOBRE A UTILIZAÇÃO DO *EXTREME PROGRAMMING* EM UMA PEQUENA EMPRESA DE DESENVOLVIMENTO PARA WEB

Florianópolis

2013

RODRIGO DOS SANTOS

ESTUDO DE CASO SOBRE A UTILIZAÇÃO DO *EXTREME PROGRAMMING* EM UMA PEQUENA EMPRESA DE DESENVOLVIMENTO PARA WEB

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Especialista.

Orientadora: Prof. Maria Inés Castiñeira. Dra.

Florianópolis

2013

RODRIGO DOS SANTOS

AVALIAÇÃO DA UTILIZAÇÃO DO *EXTREME PROGRAMMING* EM UMA PEQUENA EMPRESA DE DESENVOLVIMENTO PARA WEB

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Especialista em Engenharia de Projetos de Software e aprovado em sua forma final pelo Curso de Especialização em Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina.

Florianópolis, 29 de Janeiro de 2013.

_____

Profª e orientadora Maria Inés Castiñeira, Dra.
Universidade do Sul de Santa Catarina

_____

Prof. Jean Carlo Rossa Hauck, Dr.
Universidade do Sul de Santa Catarina

Dedico este trabalho a meu filho, Arthur Gabriel, que é minha fonte de motivação para tentar ser melhor a cada dia.

AGRADECIMENTOS

Primeiramente à minha esposa, Marta Catarina Araldi, pelo apoio e incentivo.

À minha mãe, meus sogros e cunhadas que se revesaram para tomar conta do meu filho quando precisei me ausentar para frequentar as aulas e estudar.

À Profª. Dra. Maria Inés Castiñeira pela orientação e por estar sempre à disposição para apontar meus erros e acertos.

A coisa mais difícil sobre o sucesso é que, para consegui-lo e mantê-lo, você precisa ultrapassar aquele ponto onde você normalmente desistiria. Se você conseguir ir em frente, às vezes esse esforço extra fará a diferença entre o sucesso e o fracasso.

José Aparecido Ferreira

RESUMO

O desenvolvimento de softwares é uma atividade complexa. Por este motivo torna-se indispensável a adoção de modelos de desenvolvimento de softwares que possibilitem às empresas reagirem rapidamente às demandas do mercado sem perder a qualidade ao final do projeto. O objetivo desta monografia foi analisar, através de um estudo de caso, a implantação do processo de desenvolvimento de softwares chamado *Extreme Programming* em uma pequena empresa desenvolvedora de softwares para internet. Primeiramente realizou-se uma pesquisa bibliográfica detalhando os valores e práticas do *Extreme Programming*. Em seguida foi feita a apresentação da empresa e do projeto que serviu de base para a implantação deste modelo. Assim, foram apontadas como cada uma destas práticas foram utilizadas pela empresa estudada durante a realização do projeto. Posteriormente o cliente e a equipe de desenvolvimento puderam avaliar a experiência de ter participado de um projeto utilizando o *Extreme Programming*.

Palavras-chave: *Extreme Programming*, Métologias Ágeis, Engenharia de Software

ABSTRACT

Software development is a complex activity. For that reason it is mandatory to adopt software development models that allow companies to react fast to the market demands without loosing the final project's quality. The aim of this thesis was to analyze, through a case study, the implementation of the software development process called Extreme Programming in a small web development software company. First, a literature review detailing the values and practices of Extreme Programming has been carried out. Then, a presentation of the company and the project that was the base to implement this model has been made. During the project each one of the Extreme Programming practicies used by the studied company has been analised. Later the client and the development team could evaluate the experience of having participated in a project using Extreme Programming.

Keywords: Extreme Programming, Agile Methodology, Software Engineering

SUMÁRIO

1 INTRODUÇÃO.....9
2 FUNDAMENTAÇÃO TEÓRICA: MODELO XP.....15
3 ESTUDO DE CASO.....34
4 CONCLUSÕES E TRABALHOS FUTUROS.....48
REFERÊNCIAS.....49
APÊNDICES.....51
APÊNDICE A – QUESTIONÁRIO DE AVALIAÇÃO.....52

1 INTRODUÇÃO

O desenvolvimento de software torna-se cada dia mais complexo a medida em que a tecnologia avança. Cada vez mais há computadores melhores, mais rápidos e mais acessíveis à população em geral o que faz aumentar a demanda por softwares cada vez mais modernos. Além disso, os softwares não atendem mais somente aos computadores convencionais, mas precisam também funcionar em *tablets*[1], *smartphones* e outras plataformas.

Por ser de natureza "abstrata e intangível" (SOMMERVILE, 2007 p. 3) um software pode se tornar muito complexo a medida que seus requisitos mudam muito rapidamente. Em decorrência disto, empresas que não adotam um processo definido de desenvolvimento de software estão sujeitas a enfrentar graves problemas. Entre eles, Engholm Júnior (2010, p. 21) cita os seguintes:

> Softwares difíceis de dar manutenção, tanto corretiva quanto evolutiva; Softwares difíceis de se implementar alterações; Reutilização de código mal elaborado e sujeito a geração/propagação de erros em outras partes do sistema em desenvolvimento. Sistemas com baixo desempenho e escalabilidade inadequada; Baixa eficiência no desenvolvimento; Falta de confiança no dados apresentados pelo sistema; Baixa qualidade de código.

A Engenharia de Software surgiu com o objetivo de resolver estes problemas, definindo um conjunto de princípios práticos para o desenvolvimento de software. De acordo com Bauer (1969 apud PRESSMAN, 2006 p.17), a Engenharia de software é "a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais". Estes princípios consistem em etapas ou passos que devem ser percorridos durante todo o processo de desenvolvimento de um software.

A grande maioria destes processos, conhecidos como métodos tradicionais, definem as seguintes etapas conforme afirma Pressman (2006): comunicação, planejamento, modelagem, construção e implantação. Estes processos podem acontecer de forma iterativa, ou seja, em vários ciclos de desenvolvimento conforme o Rational Unified Process (RUP) ou em apenas

---

[1] Um tablet é um computador em forma de prancheta eletrônica, sem teclado e com tela sensível ao toque. (Portal IG Tecnologia, 2011)

um, conforme o Modelo em Cascata. Ao final de cada uma destes ciclos ou etapas há a entrega de um artefato, que pode ser um documento, um diagrama, um protótipo ou um software.

O foco destas metodologias está no controle dos requisitos e no princípio que o planejamento feito anteriormente não será alterado. Oliveira (2004) cita que as metodologias tradicionais se baseiam na previsibilidade dos requisitos, que traz a grande vantagem de tornar os projetos completamente planejados, facilitando a gerência do mesmo, mantendo sempre uma linha, caracterizando o processo como bastante rigoroso.

Quando estas metodologias são aplicadas em pequenas e médias empresas e em projetos nos quais os requisitos mudam rapidamente, entretanto, não se obtém os resultados esperados. Sommerville (2007, p. 262), sobre a adoção de métodos tradicionais em empresas de pequeno e médio porte, cita que:

> Quando essa abordagem de desenvolvimento pesada e baseada em planos foi aplicada a sistemas de pequenas e médias empresas, o *overhead* envolvido foi tão grande que algumas vezes dominou o processo de desenvolvimento de software. O tempo gasto para determinas como o sistema deveria ser desenvolvido era maior do que o empregado no desenvolvimento do programa e em testes. À medida que os requisitos de sistema mudavam, o retrabalho era essencial e, em princípio pelo menos, a especificação e o projeto tinham de mudar com o programa.

Neste cenário percebemos que métodos tradicionais de desenvolvimento de software não são uma boa alternativa para pequenas e médias empresas que precisam responder rapidamente às demandas do mercado. Nesse contexto surgiram métodos de desenvolvimento de software mais adequados para situações nas quais os requisitos podem sofrer alterações frequentes, são os chamados modelos ágeis (PRESSMAN, 2006).

> O ambiente moderno de negócios que cria sistemas baseados em computador e produtos de software é apressado e sempre mutável. A engenharia ágil de software representa uma alternativa razoável para a engenharia de software convencional para certas categorias de software e certos tipos de projeto de software. Tem sido demonstrado que ela entrega rapidamente sistema bem-sucedidos (PRESSMAN, 2006, p. 58).

De acordo ainda com Sommerville (2007, p. 262):

> Os métodos ágeis contam com uma abordagem iterativa para especificação, desenvolvimento e entrega de software, e foram criados principalmente para apoiar o desenvolvimento de aplicações de negócios nas quais os requisitos de sistema mudam

> rapidamente durante o processo de desenvolvimento. Eles destinam-se a entregar um software de trabalho rapidamente aos clientes, que podem então propor novos requisitos e alterações a serem incluídos nas iterações posteriores do sistema.

Portanto, podemos concluir que a melhor alternativa para uma empresa de pequeno porte iniciar com um processo de desenvolvimento de software seria a adoção de métodos ágeis.

Sendo assim, este estudo aborda como o *Extreming Programming* (XP), um método ágil de desenvolvimento de softwares, foi utilizado em uma pequena empresa de desenvolvimento de software para web, e quais foram os resultados dessa implantação.

1.1 PROBLEMA

Sabe-se que a taxa de mortalidade entre as empresas de serviços no Brasil é de 26,9%, segundo pesquisa realizada pelo Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) em 2011. Ainda de acordo com o SEBRAE, 35% das empresas que são extintas alegam que a principal dificuldade em se manter no mercado é a forte concorrência. Por este motivo, as pequenas empresas devem sempre buscar um diferencial competitivo com o objetivo de se destacarem no mercado, mantendo seus clientes atuais satisfeitos e conquistando novos.

Como em qualquer outro ramo de negócio, as empresas de desenvolvimento de software procuram otimizar seus recursos, diminuindo seus custos e aumentando seus lucros. Este objetivo, porém, passa pela questão da qualidade do software produzido, visto que um software de baixa qualidade irá exigir um tempo maior em manutenção e correção. Como consequência, este tempo maior será traduzido como custos. Canfora & Cimitile (2000, p. 4) afirmam que a manutenção consome de 60% a 80% de tempo de todo o ciclo de vida de um software. Dessa maneira fica evidente que a adoção de um processo de desenvolvimento de software vem a contribuir com a diminuição deste tempo e, consequentemente, otimizar os recursos da empresa. Assim sendo o problema de pesquisa pode ser resumido como: é

possível adequar um modelo de software ágil para auxiliar o processo de desenvolvimento de uma pequena empresa?

1.2 OBJETIVOS

Os objetivos deste estudo são subdivididos em objetivo geral e objetivos específicos.

1.2.1 Objetivo Geral

Realizar um estudo de caso para adequar o modelo XP à realidade de uma empresa desenvolvedora de software.

1.2.2 Objetivos Específicos

- Realizar um levantamento bibliográfico sobre os conceitos do XP;
- Realizar um estudo de caso em uma empresa de desenvolvimento de software para web que não possui um processo de desenvolvimento definido;
- Descrever o processo de implantação do XP na empresa;
- Realizar uma avaliação após a implantação do processo.

1.3 METODOLOGIA DA PESQUISA

Inicialmente este estudo consiste em uma pesquisa bibliográfica, uma vez que é "um apanhado geral sobre os principais trabalhos já realizados, revestidos de importância, por serem capazes de fornecer dados atuais e relevantes relacionados com o tema" (LAKATOS e MARCONI, 2010, p 142).

Ainda sobre a pesquisa bibliográfica, Almeida Júnior (2007, p.100) nos diz que "é a atividade de localização e consulta de fontes diversas de informação escrita, para coletar dados gerais ou específicos a respeito de um determinado tema". Rodrigues (2006, p. 61) cita também que "a pesquisa bibliográfica tem por fim a aprendizagem e a aquisição de produção de conhecimento."

Este trabalho também pode ser caracterizado como um estudo de caso, pois tem como objetivo estudar um único exemplo dentro de um contexto, sendo neste caso uma empresa. Gil (2010 p. 37), explica que o estudo de caso "consiste no estudo profundo e exaustivo de um ou poucos objetos, de maneira que permita seu amplo e detalhado conhecimento. Creswell (2010, p. 269) afirma que "estudos de caso são uma estratégia em que o pesquisador explora em profundidade um programa, um evento, uma atividade, um processo ou um ou mais indivíduos".

Por ter um foco prático e com o objetivo de resolver um problema de uma empresa específica, pode-se afirmar que esta também é uma pesquisa aplicada, concordando assim com o que Barros e Lehfeld (2000, p. 78) dizem: "a pesquisa aplicada tem como objetivo a contribuição para fins práticos, visando à solução mais ou menos imediata do problema encontrado na realidade."

Uma vez que a pesquisa também envolve coleta e interpretação de dados de modo qualitativo, podemos dizer que possui também as características de uma pesquisa qualitativa. Creswel (2010, p. 271) diz que "o processo de pesquisa qualitativa envolve questões e procedimentos emergentes; coletar dados no ambiente dos participantes; analisar os dados indutivamente, indo dos temas particulares para os gerais".

Por fim, pode-se dizer também que esta é uma pesquisa exploratória, uma vez que "visa desenvolver, esclarecer e modificar conceitos e ideias. Envolve levantamento bibliográfico e documental, entrevistas e estudos de caso" (ALEXANDRE, 2003, p. 66).

1.4 ESTRUTURA DO TRABALHO

Este trabalho está subdividido em 4 capítulos.

O capítulo 1 - Introdução - apresenta a introdução, apresentação do problema, objetivos gerais e específicos e a metodologia utilizada.

O capítulo 2 – Fundamentação Teórica - traz informações e textos relevantes produzidos por especialistas em XP, que servirão como base teórica deste estudo.

O capítulo 3 – Estudo de Caso – aborda a descrição da empresa estudada, seu processo de desenvolvimento de software, o processo de implantação do XP e a avaliação final.

O capítulo 4 – Considerações finais – trata das conclusões finais do trabalho e sugestões para futuros estudos.

2 FUNDAMENTAÇÃO TEÓRICA: MODELO XP

O XP é um modelo de desenvolvimento de software ágil que procura ajudar as empresas a sanar diversos problemas como o atraso na entrega dos projetos, desperdício de recursos e falta de organização e comunicação. Beck (2004, p. 21) o criador do XP, ratifica que "o desenvolvimento de software tem falhas na entrega e falha nos valores entregues. Essas falhas têm impactos econômicos e humanos enormes". O conjunto de práticas do XP procura sanar estas falhas.

Sabe-se que as atividades que envolvem o processo de desenvolvimento de um software se repetem diversas vezes. Este ciclo de atividades é chamado de iteração. Sommerville (2007, p. 47) cita que "o processo de software não é de execução única; pelo contrário, as atividades de processo são repetidas regularmente à medida que o sistema é retrabalhado, em resposta às solicitações de mudança.". O XP procura, através de iterações curtas, responder rapidamente às mudanças de requisitos proporcionando resultados de alta qualidade de maneira econômica e flexível.

O XP se destaca também pelo seu foco nos requisitos do cliente. De acordo com Kniberg (2008), o foco da equipe que trabalha com XP é completar alguma parte do trabalho que seja entregável ao fim de cada iteração, deixando de lado o uso de uma pesada documentação e diagramas feitos em ferramentas de CASE *(Computer Aided Software Engeneering)*.

> Na *extreme programming*, todos os requisitos são expressos como cenários (chamados histórias do usuário), que são implementados diretamente como um série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes da escrita do código. Todos os testes devem ser executados com sucesso quando um novo código é integrado ao sistema. Há um pequeno espaço de tempo entre os *releases* do sistema. (SOMMERVILLE, 2007 p. 264)

O XP envolve e é formado por um conjunto de valores e práticas que serão abordados com mais detalhes a seguir. De acordo com Beck (2004), são quatro os valores do XP: comunicação, simplicidade, *feedback* e coragem. As práticas adotadas pelo XP são: O jogo do planejamento, entregas frequentes, metáfora, projeto simples, testes, refatoração,

programação em pares, propriedade coletiva, integração contínua, semana de 40 horas, cliente presente e padrões de codificação.

2.1 VALORES

Os quatro valores do XP são descritos a seguir.

2.1.1 Comunicação

A comunicação é essencial em qualquer método de desenvolvimento de software. Tudo começa com o cliente comunicando à um membro da empresa qual é o seu problema e o que precisa ser feito. Depois disto, esta pessoa comunica aos demais membros da equipe o que foi conversado com o cliente. Então são criados documentos, histórias, diagramas e demais artefatos que têm como objetivo servir de comunicação entre o cliente e a empresa assim também como entre os diversos departamentos e desenvolvedores envolvidos no projeto. Qualquer falha neste processo de comunicação irá resultar em algo que não era esperado pelo cliente, causando frustração, retrabalho e, consequentemente, prejuízo.

O XP reconhece que a comunicação é o coração do processo de desenvolvimento de um software e por este motivo recomenda que o cliente esteja presente, trabalhando no mesmo ambiente que a equipe de desenvolvimento. Muitas das falhas de comunicação que ocorrem durante um projeto de software estão justamente na falta de capacidade do comunicador expressar exatamente o que deseja transmitir e também do receptor em interpretar a mensagem. Isto acontece tanto através de documentos escritos, diagramas ou diálogos, sendo atenuado em projetos mais complexos.

> Projetos XP procuram envolver ativamente seus usuários (ou ao menos um representante dos mesmos) fazendo com que se tornem parte integrante da equipe de desenvolvimento. Na prática, isso significa que o usuário (ou seu

> representante) está presente no mesmo local onde os desenvolvedores trabalham, possibilitando que eles tenham acesso rápido e direto a um ou mais especialistas no domínio do negócio. Isso ajuda a acelerar o fluxo de informações e permite que a comunicação se baseie prioritariamente em diálogos presenciais (TELES, 2005 p. 63).

Ao trabalhar junto com a equipe, o cliente fornece *feedback* constante, respondendo em tempo real as perguntas da equipe, fazendo com que a comunicação seja mais eficaz e rápida.

2.1.2 Simplicidade

O XP sempre visa a simplicidade. Fazer a coisa mais simples que possa funcionar e ser entregue. Isto também vai de encontro à prática de *realeases*[2] curtos, a qual será discutida adiante. Beck (2004, p. 109) cita que:

> Um projeto complicado é mais difícil de ser comunicado do que um simples. Devemos, portanto, criar uma estratégia de projeto que resulte no projeto mais simples possível, consistente com nossos demais objetivos. Por outro lado, devemos criar uma estratégia de projeto que gere projetos comunicativos, em que os elementos do projeto comuniquem ao leitor aspectos importantes do sistema.

Isso não significa que o XP só é recomendado para projeto simples, muito pelo contrário. Em projetos complexos, costuma-se subdividir o projeto em tarefas simples. "Se o projeto for simples e limpo, ele permanece ágil e maleável. Retardando o projeto até que ele seja requerido, podemos fazer com que algo valioso funcione agora" (ASTELS et al., 2002, p. 9).

Esta flexibilidade em manter o projeto simples através de pequenas entregas rápidas é também destacada por Beck (2004, p. 47): "é melhor fazer uma coisa simples hoje e pagar um pouco amanhã para fazer alguma modificação nela se for necessário do que fazer uma coisa complicada hoje que talvez nunca será usada".

---

[2] Um release representa um marco no tempo no qual um conjunto coeso de funcionalidades é finalizado e lançado para consumo de seus usuários. (Teles, 2005 p. 73)

Mantendo esta simplicidade o XP proporciona também a possibilidade de se criar um software o mais próximo possível daquilo que é almejado pelo cliente. Em uma pesquisa, The Standish Group International (2001) revelou que 45% das funcionalidades que são incluídas em um software nunca são usadas. Este mesmo estudo revela ainda que 19% das funcionalidades são raramente utilizadas. Sendo assim, chegamos a conclusão que 64% dos recursos de um software poderiam ser eliminados porque nunca ou raramente são usados. O fato de pensarmos que, mais da metade do tempo de um projeto de software é gasto em funcionalidades inúteis, é realmente assustador. Isto é sem dúvida um grande impacto financeiro para qualquer empresa e qualquer projeto.

Sobre a necessidade de melhorar a comunicação e manter as coisas simples, Beck (2004, p. 47) afirma ainda que "simplicidade e comunicação têm uma maravilhosa relação de suporte mútuo. Quanto mais você se comunica, mais claramente você vê o que precisa ser feito e tem mais certeza sobre o que não precisa".

2.1.3 Feedback

O *feedback* possibilita ao cliente e ao desenvolvedor conduzirem melhor o desenvolvimento de seu produto, focando no que é mais importante e estabelecendo prioridades. O desenvolvedor sabe se está ou não no caminho certo, enquanto o cliente tem a certeza de que está sendo compreendido.

Percebe-se que o *feedback* complementa a comunicação e vice-versa. De acordo com Beck (2004 p. 48), "o *feedback* concreto trabalha juntamente com a comunicação e a simplicidade. Quanto mais *feedback* você tiver, mais fácil irá se comunicar." Este processo de comunicação e *feedback* constante, além dos *releases* curtos, evitam que se passem longos períodos de desenvolvimento e planejamento sem uma entrega real ao cliente.

Desta maneira, o cliente tem a chance de dar sua opinião sobre a funcionalidade entregue, apontando erros, falhas ou melhorias. A equipe de desenvolvimento, por sua vez, apenas irá seguir adiante após o *feedback* positivo do cliente. Astels et al. (2002, p. 3), afirmam que os clientes, "quanto trabalham junto com a equipe de desenvolvimento,

fornecem *feedback* constante, bem como respondem todas as perguntas da equipe sobre o que realmente queriam dizer".

2.1.4 Coragem

Praticar o XP requer coragem por conter várias premissas que se opõem aos modelos de desenvolvimento de software tradicional. Kunh e Pamplona (2009) apontam que a equipe de desenvolvedores precisa ter 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 estórias 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.

Percebe-se, então, que os valores do XP se complementam entre si.

> A comunicação dá suporte à coragem porque abre a possibilidade para mais experiências de alto risco e alta recompensa. Coragem dá suporte à simplicidade porque, assim que a oportunidade de simplificar o sistema é percebida, você a experimenta. O *feedback* concreto dá suporte à coragem porque você se sente muito mais seguro experimentando algo extremo no código se você puder pressionar um botão e ver resultados positivos nos testes. (BECK, 2004 p. 49)

Comunicando-se melhor e tendo um *feedback* constante do cliente, a equipe de desenvolvimento sente coragem e segurança para seguir em frente.

2.2 PRÁTICAS

As práticas do XP são descritas a seguir.

2.2.1 Cliente presente

A prática de ter o cliente presente durante todo o processo de concepção do software ratifica os valores de comunicação e *feedback* promovidos pelo XP. O cliente é o usuário final do software e conhece bem quais são os problemas que o software deve resolver. Sendo assim, é de fundamental importância ter o cliente presente fisicamente, seja fazendo-o deslocar-se até a equipe de desenvolvimento ou fazendo a equipe de desenvolvimento deslocar-se até o cliente. "O envolvimento do cliente faz uma enorme diferença no sucesso do produto". (SHORE & WARDEN, 2008 p. 30, tradução nossa)

O XP enfatiza que é o cliente o responsável por definir o rumo que o software irá tomar, porém ele não faz isto sozinho. O cliente tem à disposição a equipe de desenvolvimento para sugerir melhorias, ajustes, soluções alternativas e vice-versa.

> Um cliente real deve ficar com o time, estar disponível para responder questões, resolver disputas e definir prioridades de menor escala. Por "cliente real", eu quero dizer alguém que vá realmente usar o sistema quando ele estiver em produção. Se você está construindo um sistema para o serviço de atendimento ao consumidor, então o cliente será um representante do serviço de atendimento. Se você está construindo um sistema de negociação de penhores, o consumidor será um negociador de penhores. (BECK, 2004 p. 71).

Ao aplicar esta prática, o XP garante que o tempo entre uma solução pensada por um membro da equipe para uma nova funcionalidade e a avaliação pelo cliente seja o mais curto possível. O tempo de *feedback* é reduzido pela proximidade física entre cliente e desenvolvedores, facilitando assim a comunicação e reduzindo falhas de interpretação.

Coffin (2006) descreve uma experiência com duas equipes de desenvolvimento semelhantes, em que uma possuía o cliente presente e outra não. A equipe que não tinha o cliente presente levou quinze meses para produzir um software que foi avaliado posteriormente pelo cliente como sendo de valor insignificante para o negócio. Do outro lado, a equipe que teve o cliente presente, mesmo contando com o mesmo número de desenvolvedores, da mesma empresa, usando o mesmo processo, produziu o software em três meses e obteve grande aprovação por parte do cliente. Percebe-se com esta experiência a importância de manter equipe e cliente próximos.

> A redução da distância entre cliente e desenvolvedores também produz outro efeito benéfico: a melhoria na relação de confiança entre as partes envolvidas. Estar próximo fisicamente permite que o cliente perceba mais facilmente os esforços da equipe de desenvolvimento. Além disso, a redução no tempo para a obtenção de resultados também ajuda a elevar a satisfação do cliente, o que também melhora os relacionamentos (TELES, 2004 p. 73)

2.2.2 Jogo do planejamento

O processo de desenvolvimento de software, independente do tamanho, precisa ser planejado. O planejamento possibilita tanto ao cliente quanto à equipe de desenvolvimento ter uma estimativa de tempo para conclusão do projeto, fazendo-se um balanço entre o que é desejável e o que é possível. Também através do planejamento é possível ter uma meta a seguir, um ponto de referência para se saber se o projeto está evoluindo no caminho certo.

> O planejamento não significa passar seis ou mais meses em uma análise detalhada e um plano semana a semana para os próximos cinco anos. Entretanto, você precisa fazer algum planejamento mínimo para saber onde está indo, como você chegará lá e quais riscos podem ser encontrados ao longo do caminho (ASTELS, 2002 p. 5).

O XP também considera o planejamento importante e não o deixa de lado. Porém o planejamento é feito de forma constante. Este trabalho é feito de forma colaborativa, com a participação do cliente e da equipe de desenvolvimento.

De acordo com Beck (2004), para cada release o cliente precisa decidir sobre quatro questões: escopo, prioridade, conteúdo do release e datas de entrega. O escopo diz respeito ao que o sistema precisa fazer, qual problema ele deve resolver. A prioridade define o que é mais importante, o que deve ser feito primeiro. O conteúdo do *release* refere-se a quanto precisa ser feito para que esta entrega agregue valor ao negócio. E por fim, a data de entrega informa quando a presença do software ou parte do software faria diferença. A data de entrega informada pelo cliente muitas vezes está relacionada a questões de marketing, como por exemplo o lançamento de uma nova campanha para o natal.

Estas decisões, entretanto, não devem ser tomadas somente pelo cliente. Este deve ter o suporte e apoio da equipe de desenvolvimento, que informará o cliente sobre questões técnicas. Além disso, a equipe também decidirá sobre estimativas, consequências, processo e cronograma detalhado. A estimativa informará quanto tempo será gasto para implementar as novas funcionalidades. As consequências são questões técnicas relacionadas às decisões tomadas, como por exemplo alterações de hardware e aquisição de licenças. O processo irá revelar de que forma o trabalho será organizado. E por último, o cronograma detalhado definirá a ordem de execução das tarefas, dando prioridade aos itens mais arriscados com o objetivo de reduzir o risco total do projeto.

O planejamento no XP é feito visando o curto prazo, deixando assim a equipe preparada para eventuais mudanças nos requisitos. Planejamentos de longo prazo normalmente são imprecisos, pois de acordo com Astels et al. (2002, p. 5), "quanto mais longe no futuro for o planejamento, maiores as chances de que ele seja impreciso e maior a probabilidade da imprecisão ser grande. É muito difícil prever com antecedência". Os valores de comunicação e *feedback*, juntamente com a prática de manter o cliente dentro da equipe torna possível manter o planejamento simples, reduzindo assim os riscos do projeto.

2.2.3 Stand up metting

As *stand up meetings*[3] são reuniões rápidas realizadas no início do dia de trabalho com os participantes em pé nas quais se discute tudo o que foi realizado no dia anterior e

[3] O termo em inglês significa "Reunião Em Pé"

planeja-se o que será realizado naquele dia, também definindo os responsáveis por cada uma das tarefas.

O fato de a reunião ser realizada com os participantes em pé faz com que sejam curtas, rápidas e objetivas. Todos falam, fazendo com que os valores de comunicação e *feedback* sejam aplicados. Astels et al. (2002), recomenda que reunião seja focalizada e que não se permita que discussões sejam iniciadas.

A objetividade destas reuniões faz com que a mesma não se torne cansativa. Além disto, esta prática ajuda no entrosamento e na comunicação da equipe pois cada membro poderá saber como anda cada parte do projeto.

> O *stand up meeting* é uma reunião que força uma aproximação dos desenvolvedores de forma diária e contínua. Ele diminui os tempos de *feedback*, na medida em que cada desenvolvedor reporta as atividades executadas no dia anterior. Isso permite que toda a equipe tenha conhecimento rapidamente dos desafios que foram enfrentados por cada membro, das soluções criadas, das idéias colocadas em prática e dos problemas que precisam ser tratados com urgência. (TELES, 2004 p. 78)

Conforme o projeto segue todos dentro da equipe, inclusive o cliente, ficam atualizados sobre tudo que está acontecendo, podendo então fazer qualquer ajuste necessário em tempo hábil.

2.2.4 Programação em Pares

A programação em pares consiste em juntar dois programadores em apenas um computador. Cada um dos programadores representa um papel diferente. O que está de posse do teclado e mouse estará digitando o código, enquanto o outro estará analisando o código construído, contribuindo com a busca por erros e oportunidades de melhoria. Em um projeto XP todos os desenvolvedores programam em pares. “Duas cabeças pensam melhor do que uma, mas na XP duas cabeças juntas são melhores do que duas cabeças separadas.” (ASTELS et al., 2002 p. 9).

À princípio esta ideia pode parecer muito extrema. Deixar duas pessoas usando um mesmo computador quando se pode tê-las trabalhando simultaneamente em dois

computadores. Quanto a isto, Beck (2004 p. 76) menciona que “é mais provável a ocorrência de erros, reprojeto e o desrespeito às práticas quando as pessoas programam sozinhas, principalmente sob pressão.”

Esta prática também contribui para a comunicação da equipe, que é um dos valores fundamentais do XP. Os pares estão constantemente se comunicando com o objetivo de encontrar a melhor solução para o problema que está sendo trabalhado. De acordo com Pressman (2006, p.65) “isso fornece um mecanismo de solução de problemas em tempo real (duas cabeças são frequentemente melhores do que uma) e de garantia de qualidade em tempo real. Também mantém os desenvolvedores focados no problema em mãos”.

A programação em pares também serve como uma ferramenta de aprendizado técnico dentro de uma equipe, uma vez que haverá membros com mais experiência e conhecimento que outros.

> A programação em pares funciona para a XP porque encoraja a comunicação. Eu gosto da analogia com um tanque de água. Quando uma nova e importante informação é aprendida por alguém do time, é como colocar uma gota de corante na água. Pelo fato de os pares serem embaralhados o tempo todo, a informação se difunde rapidamente pelo time, assim como o corante se espalha na água. No entanto, ao contrário do corante, a informação torna-se mais rica e intensa quando se espalha e é enriquecida pela experiência e perspicácia de todos no time. (BECK, 2004 p. 106)

A rotatividade é outro aspecto importante da programação em pares. Os pares se revezam constantemente, fazendo com que o conhecimento seja nivelado entre os membros da equipe. Esta é uma grande vantagem para a empresa que adota o XP, pois como os pares fazem o conhecimento se espalhar pela equipe, a eventual saída de um membro não terá um impacto muito grande. Desta maneira, a empresa pode ainda se beneficiar da programação em pares como forma de treinar novos membros da equipe.

> A natureza conversacional da programação em pares também melhora o processo de desenvolvimento de software. Você aprende rapidamente a falar em muitos níveis diferentes – este código em especial, códigos como este em outros lugares do sistema, episódios de desenvolvimento parecidos com este que ocorreram no passado, sistemas parecidos com este já feitos, as práticas que você está usando e como elas podem ser melhoradas. (BECK, 2004 p. 107)

2.2.5 Código Coletivo

Além da programação em pares, todo o código produzido em um projeto XP é de propriedade coletiva. Isso significa que qualquer membro da equipe pode editar e melhorar qualquer trecho de código do projeto, sem pedir autorização para o programador que criou aquele código. Isto garante que os membros da equipe possam conhecer todo o código do sistema e também fazer modificações caso haja necessidade, evitando situações nas quais somente um desenvolvedor tem a propriedade de uma classe e todas as pessoas que precisam fazer uma alteração têm que solicitar para o proprietário (ASTELS et al., 2002).

A ideia de ter várias pessoas mexendo no mesmo código pode parecer estranha, ainda mais se for considerada a possibilidade de uma pessoa danificar uma parte do código, ou alterar uma parte e gerar defeito em outra. A equipe XP tem liberdade para trabalhar desta forma pois o código produzido é orientado a testes, o que significa que cada classe e cada método tem seus testes automatizados, o que garante o seu funcionamento. Escrevendo e executando testes diminuem-se as chances de estragar algo acidentalmente (BECK, 2004).

Com a propriedade coletiva a equipe também passa a ser menos afetada pela ausência de um membro da equipe, por motivo de doença ou férias, por exemplo. Todos estão habituados a realizar alterações em diversas partes do código.

> A propriedade coletiva também tende a espalhar o conhecimento do sistema por todo o time. É improvável que exista uma parte do sistema que apenas duas pessoas conhecem (precisa ser ao menos um par, o que já é melhor do que a situação usual, em que um programador esperto mantém a todos como reféns). Isso reduz ainda mais o risco do projeto. (BECK, 2004 p. 105)

2.2.6 Código Padronizado

Em um projeto XP, todo código produzido deve seguir um padrão. Este padrão consiste em um conjunto de regras comuns, como nomenclatura de classes, métodos, variáveis e formatação do código.

> Se você terá todos esses programadores trabalhando em tantas partes diferentes do sistema, trocando de dupla várias vezes ao dia e refatorando os códigos uns dos outros constantemente, você não pode ter conjuntos diferentes de práticas de codificação. Com um pouco de prática, será impossível dizer qual pessoa do time escreveu que código. (BECK, 2004 p. 72)

Além de ajudar na comunicação entre os desenvolvedores e facilitar a programação em pares, o adoção de um padrão fornece meios para manter o código coletivo. Outra vantagem desta técnica é o ganho de tempo na produção de código novo.

> Você pode trabalhar com velocidade total porque não precisa reformatar o código durante o trabalho, nem precisa estar constantemente alternando de um estilo para outro ou tentando entender a formação e o estilo, bem como o próprio código. (ASTELS, 2002 p. 10)

2.2.7 Design Simples

Os projetos XP sempre são baseados na simplicidade. Fazer a coisa mais simples que possa funcionar, em iterações curtas. Desta forma, qualquer alteração ou correção que precise ser feita no software irá gerar um baixo custo. Isto se opõe às premissas dos métodos tradicionais de desenvolvimento de software, que sugerem que o custo de se fazer uma alteração em um software cresce exponencialmente ao longo do tempo.

> O custo de consertar um problema em um software qualquer aumenta exponencialmente ao longo do tempo. Um problema que talvez custe um dólar para ser consertado se você o encontrou durante a análise de requisitos pode custar milhões de dólares quando o software estiver em produção (BECK, 2004 p. 39)

Neste cenário os programadores tentam resolver os problemas sendo generalistas, criando códigos genéricos pensando em funcionalidades que podem vir a ser solicitadas no futuro, pois assim o impacto destas mudanças seriam amenizadas. Ao aderir a esta prática, no entanto, “erramos com mais frequência que acertamos” (TELES, 2006 p. 152), isto porque é

pouco provável que se possa adivinhar o futuro. “Sendo assim, é comum criarmos soluções que jamais serão necessárias, ou deixamos de criar soluções para os problemas que realmente ocorrerão no futuro” (TELES, 2006 p. 152).

No XP, o custo de uma alteração cresce lentamente ao longo do tempo. Por buscar sempre a entrega rápida da solução mais simples que possa funcionar, evita-se que os programadores pensem em soluções genéricas e que provavelmente nem sejam usadas. O foco está em resolver um problema de cada vez, sem a preocupação de generalizar pensando no futuro. O *feedback* do cliente e os *releases* curtos garantem isso. A equipe trabalha em um ciclo constante de *feedback*, sempre tirando dúvidas com o cliente, e evitando assim a necessidade de se pensar em generalizações.

> Se o custo das modificações aumentasse vagarosamente ao longo do tempo, você agiria de maneira completamente diferente […] Você tomaria grandes decisões o mais tarde possível, durante o processo, para adiar o custo de tomar as decisões e para ter a maior chance na esperança de que as necessidades do amanhã por você antecipadas não se tornassem realidade. Você introduziria elementos no projeto apenas se eles simplificassem o código existente e fizessem com que a escrita do próximo trecho de código fosse mais simples. (BECK, 2004 p. 41)

2.2.8 Testes

Sabe-se que o custo para consertar ou alterar um software cresce exponencialmente ao longo do tempo (BECK, 2004). Isso significa dizer que consertar um software custa caro, principalmente quando este software já está em uso. Entre os motivos, pode-se citar que passados alguns dias ou até mesmo horas os programadores irão se esquecer do código que escreveram. Caso haja necessidade de mexer no código, o programador terá que ler novamente tudo o que foi escrito para tentar entender o problema. Além disso, há grande possibilidade de que novos erros sejam introduzidos. Astels et al (2002, p. 7), afirma que “muito código é distribuído sem ser totalmente testado […]. Este é um procedimento caro e instável.”

Em um projeto XP todo o código produzido é orientado a testes, ou seja, cada pequeno trecho de código está coberto por um teste automatizado que pode ser executado rapidamente, chamado de teste unitário. “Os testes unitários que são criados devem ser implementados usando um arcabouço que lhes permita ser automatizados (consequentemente, eles podem ser executados fácil e rapidamente” (PRESSMAN, 2006 p. 65).

O TDD (*Test Driven Development*[4]) no XP orienta que a cada estória ou funcionalidade introduzida no sistema deve-se também escrever um mecanismo de teste automatizado como uma técnica preventiva durante todo o projeto e manutenção do sistema (BECK, 2000). Esta prática fornece confiança para a equipe de desenvolvimento, pois os testes são executados várias vezes ao dia. Ao testar uma funcionalidade, testa-se também todas as outras anteriores, tendo-se assim a certeza de que tudo está funcionando e que o código adicionado não afetou o restante do sistema. Consequentemente, o efeito “conserta aqui, quebra lá” é totalmente eliminado. “Essa coleção abrangente de teste dá uma ótima sensação de confiança. Essa confiança permite que você seja agressivo porque, se quebrar alguma coisa, descobrirá na próxima vez que executar os testes.” (ASTELS et al., 2002 p. 8).

Os testes no XP não se limitam apenas ao código e não são apenas responsabilidades dos programadores. Além dos testes unitários há também os testes de aceitação, feitos pelo cliente com ou sem o auxílio da equipe. Para cada nova funcionalidade que é completa, o cliente realiza um teste de aceitação, para ver se tudo está saindo como o esperado do ponto de vista do usuário final. Os testes de aceitação também permitem que a nova funcionalidade seja testada juntamente com as demais existentes.

> Podemos entender o teste de aceitação como sendo um roteiro que tem vinculado a ele um conjunto de respostas esperadas. Dada uma estória, é importante escrever um passo-a-passo que o usuário deverá executar. Para cada ação, existe uma resposta esperada que está escrita no passo-a-passo. O usuário deve executar as ações e comparar os resultados com aqueles esperados. Se o sistema apresenta um resultado diferente é porque existe um erro nele ou o próprio teste está incorreto. (TELES, 2006 p. 138)

Com os testes, tanto de unidade quanto de aceitação, o XP garante que cada nova funcionalidade está seguindo no rumo correto e que está funcionando.

> Em resumo, pense nesta prática como o caminho da prevenção. Existe um custo associado, sem dúvida. Mas, esteja absolutamente seguro de que este

[4] O termo em inglês significa “Desenvolvimento Guiado por Testes”

> custo é infinitamente menor que o custo de remediar, exatamente como acontece com nossas vidas. Portanto, teste tudo aquilo que você produzir, pois somente assim você estará totalmente coberto e poderá assegurar que a maior parte do tempo do projeto é dedicada ao desenvolvimento de software e não à depuração de *bugs*. (TELES, 2006 p. 108)

2.2.9 Refatoração

A refatoração (*refactoring*) é uma prática constante em um projeto XP e consiste na melhoria contínua do código com o objetivo de torná-lo mais fácil de ler e de utilizar.

De acordo com Teles (2006, p. 25):

> O *refactoring* é o ato de alterar um código sem afetar a funcionalidade que ele implementa. É utilizado para tornar o software mais simples de ser manipulado e se utiliza fortemente dos testes descritos anteriormente para garantir que as modificações não interrompam o seu funcionamento.

Ao se deparar com um trecho de código de difícil leitura ou que não deixa claro quais seus objetivos, o programador deve, antes de mais nada, realizar o *refactoring* para tornar aquele código compreensível, para então prosseguir com a nova funcionalidade que deseja implementar. O *refactoring*, portanto, é parte constante do desenvolvimento em um projeto XP, ajudando a manter o projeto e o código o mais simples possível.

> Você deve fazer o *refactoring* o tempo todo. Quando escreve testes, você pode fazer o *refactoring* para tornar alguma coisa mais fácil de ser testada. Quando codifica alguma funcionalidade, você pode fazer o *refactoring* para facilitar a inclusão ou remoção de duplicatas. Outra ótima maneira de usar o *refactoring* é tomar código existente e reestruturá-lo para que sej compatível com um padrão de projeto. (ASTELS et al., 2002 p. 13)

De acordo com Beck (2004, p. 69) “ao trabalhar desta forma, você garante que poderá adicionar a próxima função com uma quantidade aceitável de esforço, e todas as seguintes também”.

A ideia de alterar um trecho de código sem alterar sua funcionalidade soa arriscada no início. Se uma determinada parte de um software está funcionando perfeitamente,

o simples fato de mexer no código pode introduzir diversos erros. Mas é neste ponto que percebemos como as práticas do XP se completam. Ao adotar o desenvolvimento orientado a testes garante-se que o *refactoring* pode ser feito de maneira segura. A cada alteração, basta rodar os testes automatizados para verificar se o código alterado ainda possui o mesmo comportamento. Desta maneira o *refactoring* pode ser feito sem receio de quebrar alguma funcionalidade no software, visto que os testes garantem que tudo continua funcionando.

2.2.10 Integração Contínua

Em um projeto de desenvolvimento de software, costuma-se dividir a equipe de modo que cada uma trabalhe em uma parte específica do sistema. Normalmente esta divisão é feita por módulos. Equipe A trabalha no módulo Cliente, Equipe B trabalha no módulo Financeiro, e assim por diante. De fato esta é uma boa escolha pois limita o nível de complexidade a que cada equipe fica exposta. Por outro lado, há o problema de compatibilidade entre estes módulos. Uma função que seja alterada ou adicionada no Módulo A poderá interferir no Módulo B, por exemplo. Também corre-se o risco de um programador sobrescrever o código feito por outro.

O XP resolve este conflito através da integração contínua.

> Quando uma nova funcionalidade é incorporada ao sistema, ela pode afetar outras já implementadas. Para assegurar que todo o sistema esteja sempre funcionando de forma harmoniosa, a equipe pratica a integração contínua que leva os pares a integrarem seus códigos com o restante do sistema diversas vezes ao dia. (TELES, 2006 p. 27)

A resolução de conflitos no código é resolvida através dos testes unitários. A cada integração, o par de programadores deve executar todos os testes unitários para ter certeza que nada foi quebrado. Quando os testes rodam com 100% de sucesso, a integração é feita.

> Os testes permitem que você saiba com confiança que, se tudo (ou todo o sistema) for executado com sucesso, você pode liberar o novo *release* do código-base. Integrando com essa frequência, a chance de as alterações entrarem em conflito com as alterações de outra pessoa é minimizada. Se ainda houver um conflito, ele será mínimo. Esses ciclos curtos de integração também mantém a convergência entre

> todas as pessoas. O sistema não tem tempo para divergência entre as integrações. (ASTELS et al., 2002 p. 13)

2.2.11 Releases Curtos

Dando ênfase aos valores de comunicação e *feedback* constante, o XP trabalha com *releases* curtos. O objetivo é terminar uma pequena funcionalidade no menor tempo possível e liberar para que o cliente possa usar em produção. Beck (2004, p. 67) menciona que "cada versão entregue deve ter o menor tamanho possível, contendo os requisitos de maior valor para o negócio".

Com os *releases* curtos o XP pretende dar ao cliente o máximo de valor econômico em um curto espaço de tempo, fazendo com que o cliente consiga uma versão utilizável do seu software no menor tempo possível e a partir daí o software vai sendo incrementado a cada nova iteração. Percebe-se que esta prática também ajuda a eliminar falhas na concepção do software, pois o cliente estará acompanhando cada novo *release* do sistema, tornando mais fácil efetuar qualquer alteração caso seja necessário.

> O XP tem como objetivo gerar um fluxo contínuo de valor para o cliente. Sendo assim, ele trabalha com *releases* curtos, ou seja, 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. Durante todo o projeto, a equipe colocará o sistema em produção diversas vezes, cada vez incorporando mais funcionalidades e gerando mais valor. (TELES, 2006 p. 27)

A prática de lançar releases curtos, no entanto, não pode ser confundida com entregar funcionalidades incompletas. De fato, Beck (2004) afirma que o *release* precisa fazer sentido e não se pode implementar meia função e entregá-la apenas para tornar menor o ciclo de entrega. Para Astels e outros (2002, p. 14) "um *release* com um recurso implementado pela metade não tem muita utilidade para ninguém. A regra para uma equipe XP é que o *release* não pode ser liberado enquanto não estiver completamente acabado".

2.2.12 Metáfora

A metáfora é utilizada no XP para que todos os participantes da equipe possam ter a mesma ideia conceitual sobre o projeto e os problemas a serem resolvidos. O cliente, por fazer parte da equipe, muitas vezes poderá ter dificuldades em entender um termo técnico ou um conceito sobre programação, design ou arquitetura. Do outro lado, os programadores também podem ter dificuldades em entender uma regra de negócio que o cliente queira transmitir. Para ambos os casos usam-se metáforas para que todos possam compreender e serem compreendidos.

De acordo com Teles (2006, p. 26), "a equipe de desenvolvimento utiliza metáforas, já que elas têm o poder de transmitir ideias complexas de forma simples, através de uma linguagem comum que é estabelecida entre a equipe de desenvolvimento e o cliente". Beck (2004, p. 68) cita que a metáfora "ajuda todos os envolvidos no projeto a entenderem os elementos básicos e seus relacionamentos".

O objetivo final da metáfora é servir de apoio à comunicação, garantindo que as pessoas envolvidas no projeto compartilhem da mesma compreensão do problema. Entretanto, a metáfora não tem a pretensão de ser precisa. "As metáforas não significam ser preciso em todos os aspectos. Em vez disso, elas fornecem um contexto para discutir os problemas e soluções." (ASTELS et al., 2002 p. 4).

2.2.13 Ritmo Sustentável

Uma prática comum das empresas de desenvolvimento de software, principalmente quando se aproxima o prazo limite para entrega de um projeto, é que se adote uma jornada de trabalho maior, fazendo os desenvolvedores trabalharem até mais tarde ou em feriados e finas de semana. O XP proíbe tal prática e recomenda que a equipe trabalhe no máximo 40 horas semanais, respeitando a individualidade de cada membro da equipe. Para Astels e outros (2002, p. 14) "ninguém faz seu melhor trabalho quando está estressado, sob

pressão e cansado. É curioso que essas sejam exatamente as condições que prevalecem em nosso setor".

Beck (2004) afirma que horas extras de trabalho são sintomas de um problema sério no projeto e que isto não será resolvido fazendo as pessoas trabalharem mais.

> A qualidade do design, do código, das metáforas e do sistema é determinada diretamente pela qualidade dos desenvolvedores e a capacidade que eles têm de se manter atentos, criativos e dispostos a solucionar problemas. Para garantir que a equipe tenha sempre o máximo de rendimento e produza software com melhor qualidade possível, o XP recomenda que os desenvolvedores trabalhem apenas oito horas por dia e evitem fazer horas extras, visto que é essencial estar descansado a cada manhã, de modo a utilizar a menta na sua plenitude ao longo do dia. (TELES, 2006 p. 27)

Trabalhar oito horas por dia e quarenta horas por semana são valores conceituais. "O importante é não trabalhar mais do que funciona para você" (ASTELS et al., 2002 p. 14).

2.3 CONSIDERAÇÕES FINAIS SOBRE O CAPÍTULO

Neste capítulo foram abordados as práticas e valores do XP e a maneira como se completam e apoiam uns aos outros. Foram apresentados tantos os aspectos técnicos quanto os aspectos de relacionamento pessoal e comunicação. Destacou-se também a importância da participação do cliente durante a fase de desenvolvimento e como a comunicação é um fator importante durante o projeto, pois proporciona tanto para o cliente quando para a equipe confiança e convicção de que o projeto está sendo guiado em direção ao caminho certo.

O próximo capítulo apresentará um estudo de caso onde será abordado o processo de desenvolvimento de software de uma empresa e a experiência obtida ao utilizar o XP pela primeira vez em um de seus projetos.

3 ESTUDO DE CASO

O estudo de caso visa descrever a aplicação das práticas do *Extreme Programming* em um projeto de desenvolvimento de software, o qual teve duração de seis meses.

Devido ao projeto ser de teor comercial e estar protegido por um termo de sigilo, o nome do software e do cliente foram substituídos por nomes fictícios. Todas as informações foram obtidas com a participação do autor da monografia neste projeto, que teve oportunidade de participar da equipe de desenvolvimento.

3.1 DESCRIÇÃO DA EMPRESA

O objeto de análise deste estudo é uma pequena empresa voltada para o desenvolvimento de sistemas para internet chamada Ilha Web. Esta empresa nasceu como um *hobbie* do fundador, que iniciou em 2004 fazendo pequenos sites para uso pessoal e para alguns amigos, mesmo tendo formação em uma área do conhecimento não relacionada com Tecnologia da Informação (TI). Com o decorrer do tempo seu trabalho foi ganhando destaque e mais e mais oportunidades foram surgindo, chegando então o momento em que se tornou viável a criação da empresa no final do ano de 2006.

Atualmente, com uma equipe de quatro desenvolvedores e um analista de suporte, a Ilha Web atende projetos de pequeno e médio porte, tendo como seu foco principal o desenvolvimento de sistemas para empresas que adotam o modelo de vendas em rede, chamado Marketing Multinível.

A empresa trabalha com tecnologias voltadas para a web e utiliza, em sua maioria, ferramentas *open source*. A principal linguagem de programação utilizada na empresa é o PHP (*Hypertext Preprocessor*), sendo assim, a equipe de desenvolvimento é formada por especialistas nesta linguagem.

3.2 PROCESSO DE DESENVOLVIMENTO

Devido ao proprietário ter formação acadêmica na área de Ciências Humanas e não na área de Computação ou TI, não houve uma preocupação inicial em formalizar um modelo oficial de desenvolvimento de software. Por este motivo, desde o início de suas atividades, em 2006, até o início de 2011, a empresa não seguia nenhum modelo de desenvolvimento de software em seus projetos.

Com o passar do tempo e o aumento da carteira de clientes percebeu-se a necessidade de se adotar um modelo de desenvolvimento que facilitasse a padronização dos projetos e possibilite um resultado final de qualidade entregue em tempo hábil. Entre os principais problemas percebidos estavam a repetição de tarefas, falta de padronização dos módulos desenvolvidos, ausência de integração entre o código desenvolvido, atraso na entrega dos projetos e alta taxa de retrabalho.

A falta de padronização de código e ausência de uma ferramenta de integração contínua faziam com que comumente uma biblioteca ou módulo desenvolvido por um desenvolvedor ficasse sempre vinculada àquele desenvolvedor, visto que este era o único que entendia como aquele trecho de código funcionava e como se integrava com o restante do sistema. Sempre que era necessária um alteração naquela parte do sistema recorria-se à pessoa que a havia desenvolvido. Isto ocasionava sobrecarga de trabalho para este desenvolvedor, que já estava trabalhando em um outro projeto e precisava parar o que estava fazendo para trabalhar no código anterior.

Outro agravante da falta de padronização era a impossibilidade de reutilização das bibliotecas de código para serem aproveitadas em novos projetos. Era comum um desenvolvedor escrever um módulo que já havia sido desenvolvido em outro projeto. Muitas vezes isso acontecia por falta de comunicação entre os desenvolvedores e outras vezes acontecia porque o código não era suficientemente compreensível para o novo desenvolvedor.

De uma maneira geral, era necessário a melhoria da comunicação como um todo. Comunicação interna, entre os membros da equipe, comunicação através do código desenvolvido, para que todos entendessem o que estava sendo feito, e também a comunicação com o cliente. Com essa melhoria na comunicação, esperava-se uma colaboração maior entre

os desenvolvedores, diminuição do tempo de desenvolvimento e também diminuir a taxa de retrabalho.

3.3 IMPLANTAÇÃO DO XP

Uma vez identificados os problemas decorridos pela falta de um modelo de desenvolvimento de software na empresa e também o reconhecimento por parte da mesma de que havia chegado o momento de estabelecer um, o XP foi sugerido pelo autor desta monografia como uma alternativa. Devido à empresa já estar com outro projeto em desenvolvimento naquele momento, foi decidido que iria-se utilizar as práticas do XP no próximo projeto. Caso os resultados obtidos com o XP fossem satisfatórios, o modelo seria adotado de maneira definitiva na empresa.

Após algumas semanas a empresa foi procurada por uma empresa de lazer e entretenimento que tinha planos de lançar um novo sistema. Este sistema consistia em uma rede social para um público bem específico. Entre as funcionalidades a serem implementadas estavam a criação de um rede de amigos, álbum de fotos, postagem de vídeos, envios de convites, enquetes, ferramenta de envio de mensagens privadas entre os usuários, fóruns de discussão, bate-papo, além de toda a identidade visual do sistema. Para fins de identificação e para preservar a identidade do cliente, o projeto será chamado de Rede ABC.

Este projeto foi a oportunidade que estava sendo aguardada para iniciar a utilização do modelo XP. O cliente foi informado do modelo de desenvolvimento que seria adotado e concordou em ceder um de seus colaboradores para ser parte da equipe de desenvolvimento. Este colaborador ficaria disponível, a princípio, diariamente por duas horas durante as manhãs, salvo exceções que seriam comunicadas. De fato, o cliente se mostrou muito receptivo ao modelo XP e chegou a elogiá-lo mesmo antes do início do projeto, tendo enfatizado que jamais havia visto uma empresa trabalhar desta forma.

Algumas reuniões iniciais com o cliente foram necessárias antes do início do desenvolvimento do projeto. Houve uma dificuldade inicial na comunicação com o cliente quando tentava-se explicar o conceito de *releases* curtos, pois o cliente imaginava que grandes partes do sistema seriam entregues de cada vez, mas após alguns encontros foi

possível se fazê-lo entender a ideia de entregas curtas e planejar o primeiro *release* juntamente com a criação das primeiras histórias. Cada *release* foi planejado para ter duração de duas semanas. No dia anterior ao término do *release* era realizada uma reunião com o cliente para realizar o teste de aceitação e para planejar as funcionalidades que seriam acrescentadas no próximo *release*.

Para tornar possível a adoção das práticas XP foi necessário a utilização de algumas ferramentas de desenvolvimento novas e a utilização de algumas delas exigiu um pequeno treinamento inicial. Os detalhes das ferramentas utilizadas juntamente com as práticas XP aplicadas no projeto são analisadas a seguir.

3.3.1 Cliente Presente

Ter o cliente presente foi uma das novidades que o XP trouxe para dentro da empresa. Nos projetos anteriores a comunicação com o cliente era escassa. Havia um levantamento de requisitos inicial, onde o cliente informava as características desejadas e suas expectativas em relação ao projeto. Em seguida estes requisitos eram transmitidos à equipe de desenvolvimento. O contato com o cliente era feito de maneira irregular, muitas vezes somente por telefone ou e-mail. Enviava-se uma amostra online para que o cliente pudesse avaliar e dar *feedback*. Esta abordagem dava certo algumas vezes, mas na maioria das vezes resultava em alguma discrepância entre o que o cliente realmente desejava e o que havia sido feito. Também era comum o cliente encontrar falhas que não haviam sido devidamente identificadas durante o desenvolvimento, resultando em retrabalho e atrasos.

Durante o projeto da Rede ABC tivemos a presença diária de um representante do cliente, que viria a ser um dos administradores do sistema após seu lançamento. Desta forma, este representante possuía pleno conhecimento dos objetivos do projeto e poderia auxiliar a equipe. A presença do cliente, entretanto, não foi em tempo integral. Devido à outros compromissos o cliente poderia passar somente duas horas diárias junto à equipe de desenvolvimento, comunicando com antecedência os dias em que não poderia comparecer, o que ocorreu em um número significativo de vezes.

O cliente juntava-se à empresa no início do expediente, pela manhã, para também participar das *stand up mettings.* Com isto, já havia um retorno para o cliente sobre o que havia sido feito no período em que ele não estava presente, além de poder acompanhar de perto a evolução do projeto. Após a reunião, o cliente ficava à disposição da equipe para dar *feedback* imediato em qualquer dúvida do projeto.

A ideia de ter o cliente presente durante o desenvolvimento foi inicialmente mal recebida por alguns membros da equipe, resultado da cultura de considerar os clientes em geral 'chatos' e que iriam solicitar coisas que não foram pedidas. Porém, com o decorrer do projeto, esta prática se mostrou bastante eficiente na medida em que foi percebida a maneira constante em que o projeto foi evoluindo. Toda a subjetividade que era utilizada nos projetos anteriores, a qual todos já estavam acostumados, deu lugar à objetividade. Sabia-se de maneira muito rápida exatamente o que o cliente desejava e o *feedback* era imediato, o que resultou em muito pouco retrabalho.

3.3.2 Jogo do planejamento

O planejamento para este projeto foi feito em duas fases. Na primeira, desenhou-se um panorama geral do sistema, englobando todas as funcionalidades desejadas pelo cliente. Em seguida, essas funcionalidades foram transformadas em histórias, sendo que cada história representava uma ou mais funcionalidades.

Passada esta primeira etapa, foi o momento de priorizar as funcionalidades, ou seja, definir junto ao cliente o que era mais importante e deveria ser feito primeiro. Isto foi difícil no início, pois o cliente priorizava muitas coisas simultaneamente, sendo impossível implementar tudo em apenas um *release.* Houve a necessidade de explicar ao cliente o conceito de *releases* curtos e a maneira incremental com que o software seria desenvolvido. Uma vez que este conceito foi absorvido e aceito pelo cliente, a fase de planejamento passou a ser feita de forma mais fluente.

O planejamento, no entanto, não foi realizado somente no início do projeto. Ao final de cada *release*, exatamente no dia anterior à entrega, era realizada uma reunião com o cliente. Esta reunião tinha como primeiro objetivo realizar o teste de aceitação, ou seja, fazer

o cliente testar tudo que havia sido feito naquele *release*, obtendo sua aprovação. Neste momento era comum o cliente solicitar algumas alterações ou ajustes. Os ajustes de pequeno porte eram providenciados imediatamente pela equipe de desenvolvimento enquanto os ajustes mais complexos acabavam virando uma outra história para ser implementada no próximo *release*. O segundo objetivo desta reunião era planejar o que deveria ser implementado a seguir, portanto, era necessário que o cliente novamente priorizasse quais as funcionalidades trariam maior valor ao seu negócio e que, portanto, deveriam ser realizadas primeiro.

Seguindo as orientações de Beck (2004), a equipe de desenvolvimento sempre implementava primeiro as histórias marcadas como prioridade pelo cliente.

3.3.3 Stand up Meetings

As *stand up meetings* eram realizadas diariamente as 9h da manhã, horário em que normalmente todos já haviam chegado na empresa, inclusive o representante do cliente. Foi estipulado que a reunião teria duração de 15 minutos, sendo prorrogada para 20 minutos em casos extraordinários. Nas primeiras reuniões, devido à falta de prática de todos, houve falta de tempo para que todos pudessem falar, entretanto a partir da terceira reunião, todos estavam cientes de que o tempo era curto e que deveriam ser breves e objetivos. O fato de as reuniões serem realizadas em pé também ajudou neste sentido.

Durante as reuniões cada membro da equipe apresentava brevemente o que havia feito no dia anterior, relatava se enfrentou alguma dificuldade e concluía dizendo o que estava programado para ser feito naquele dia. Com isto cada participante podia se atualizar sobre o que estava acontecendo no projeto como um todo, e se manter informado sobre o que seus colegas estavam fazendo. Em muitas oportunidades, ao relatar uma dificuldade durante as reuniões, o participante pôde resolver seu problema com a opinião e ajuda dos demais.

3.3.4 Programação em par

A primeira etapa para a implantação da programação em pares foi o reposicionamento das mesas e computadores. Anteriormente os desenvolvedores permaneciam em mesas posicionadas uma em frente à outra, o que impossibilitava que um desenvolvedor conseguisse ver o monitor do outro. Este posicionamento também tornava difícil a colaboração entre a equipe, uma vez que era necessário levantar da cadeira e dar a volta na mesa para poder ver o monitor do colega.

Para viabilizar a programação em pares as mesas foram posicionadas de maneira que todos pudessem ver o monitor de todos. Não houve problemas em relação ao tamanho das mesas e cadeiras, visto que as mesas comportavam bem duas pessoas sentadas olhando para a mesma tela.

A apesar de programação em pares ser uma novidade para todos na equipe, a adaptação inicial foi rápida devido ao fato de todos já se conhecerem e já terem trabalhado juntos em outros projetos.

Os pares se mantinham os mesmos durante cada *release*, alternando-se diariamente entre piloto (quem está com o teclado) e o co-piloto. No início de cada novo *release* os pares eram trocados. Esta prática também contribuiu para o nivelamento do conhecimento entre a equipe, visto que haviam desenvolvedores com mais experiência que outros.

Programar em pares facilitou a construção de um código mais legível para todos e também na detecção prematura de possíveis *bugs* (defeitos). Ao programar em pares o piloto, ao mesmo tempo que digita o código, explica em voz alta o que está fazendo, de maneira que o co-piloto possa entender e também dar sua opinião. Se alguma coisa soa estranha ou não faz sentido, o co-piloto imediatamente intervem dando sua opinião. Percebeu-se com esta prática que o fluxo de desenvolvimento foi mais constante do que nos projetos anteriores, uma vez que havia a comunicação explícita do que estava sendo feito.

3.3.5 Desenvolvimento Guiado por Testes

O Desenvolvimento Orientado a Teste, ou TDD (*Test Driven Development*), não era praticado nos projetos habituais da empresa, o que acabou sendo uma das grandes dificuldades da equipe de desenvolvimento no início do projeto. Apenas um membro da equipe, o autor desta, tinha experiência com este conceito e dominava as ferramentas necessárias para se utilizar TDD juntamente com a linguagem de programação PHP.

Por este motivo, antes do início da produção de código para a construção do software foi realizado um treinamento específico com o objetivo de introduzir o conceito de TDD. Neste treinamento, que teve duração de um dia, foi abordado o uso da ferramenta PHPUnit[5], desde sua instalação até a criação de testes automatizados, de maneira que todos puderam criar um projeto fictício usando TDD, servindo então como base para utilização no projeto real.

Durante os primeiras semanas do projeto a equipe ainda tinha dúvidas sobre como criar testes para determinadas situações, duvidas estas que eram imediatamente sanadas com o auxílio dos demais colegas e com a consulta da documentação oficial.

Nos primeiros dias de desenvolvimento percebeu-se que a produção estava um pouco aquém do esperado, devido à falta de experiência na programação com testes. Este problema foi resolvido gradualmente, na medida em que os desenvolvedores adquiriam experiência.

Este atraso de maneira alguma significou atraso na entrega dos *releases*, visto que o código que foi produzido orientado a testes foi de grande qualidade e dificilmente apresentava falhas. À medida que cada funcionalidade era concluída com seus próprios conjuntos de testes unitários, tinha-se certeza de que estava funcionando e que continuaria a funcionar enquanto os testes tivessem passado.

---

[5] http://www.phpunit.de

3.3.6 Refatoração

Com a implantação do TDD, a prática de refatoração foi muito facilitada. Em projetos anteriores tinha-se medo de fazer qualquer tipo de refatoração, pois a possibilidade de quebrar algo era grande. Corria-se também o risco de ajustar o código para resolver um problema e ao mesmo tempo estragar outra coisa em outro lugar. O TDD possibilitou que esse medo fosse superado, uma vez que ao fazer uma refatoração, bastava rodar os testes automatizados para ver se tudo continuava funcionando. Dessa maneira, todos na equipe sentiam-se à vontade para fazer qualquer refatoração que fosse necessária, uma vez que, se algo desse errado o problema poderia ser automaticamente percebido e resolvido.

A prática da refatoração foi uma constante durante todo o projeto, visto que as funcionalidades eram entregues de maneira incremental, ou seja, criava-se a funcionalidade de maneira simples para depois adicionar os itens mais complexos. Cada vez que isto era feito era necessário refatorar o código.

3.3.7 Código Coletivo

A prática do código coletivo foi outro paradigma quebrado. O costume era de o código ter um 'pai': aquele desenvolvedor que criou inicialmente o código. Usualmente somente ele entendia e somente ele poderia alterar ou dar manutenção ao software. Com a implantação do XP, todos passaram a ser 'pais' do código e não precisavam pedir para que outra pessoa fizesse alterações.

O resultado disto foi agilidade no desenvolvimento. Ninguém precisava esperar por ninguém para fazer qualquer ajuste necessário no código e os testes automatizados garantiam que tudo estivesse funcionando. Com isto a equipe também manteve um conhecimento amplo do projeto, pois cada dupla de programadores sabia o que o outro par havia desenvolvido.

3.3.8 Código Padronizado

A padronização do código criado no projeto foi implantada através da ferramenta PHPCS[6] (*PHP Code Sniffer*) que detecta erros de formatação nos códigos de acordo com um padrão preestabelecido. Antes de enviar o código para o repositório de controle de versão, rodava-se a ferramenta no código recém-criado e o PHPCS identificava os pontos que precisavam ser formatados. O código só era aceito se toda a formatação estivesse correta e todas as classes e métodos criados estivessem documentados. Desta maneira garantiu-se que todo o código obedeceria o mesmo padrão e que, como consequência, seria entendido por todos na equipe, além da facilitar a refatoração.

3.3.9 Design simples

O design simples foi adotado desde o início do projeto, uma vez que eram construídas somente as funcionalidades suficientes para atender à história que estava sendo implementada. A equipe procurou adicionar as funcionalidades de maneira incremental e sem considerar o que poderia ser solicitado no futuro. Cada história foi implementada da maneira mais simples que poderia ser feita para funcionar.

Adotar esta abordagem foi importante para o projeto visto que alguns dos requisitos mudaram e com isto não foi necessário investir muito tempo para realizar grandes alterações ou mudar grandes estruturas de código. O design simples proporcionou agilidade para a equipe responder rapidamente às mudanças de requisitos que ocorreram no projeto.

---

[6] http://pear.php.net/package/PHP_CodeSniffer/

3.3.10 Integração Contínua

A integração foi feita através da ferramenta *Phing*[7] e da ferramenta de controle de versão GIT, que foi fornecida através do serviço online *BitBucket*[8].

Antes de se iniciar os trabalhos do dia, todos atualizavam suas máquinas com a última versão do código-fonte do projeto. Este processo é chamado de *checkout*. Após isto todos iniciavam seus trabalhos de criação e refatoração de código. A cada tarefa que era concluída, a dupla de programadores enviava o código recém-criado para o repositório de controle de versão rodando a ferramenta *Phing*.

O *Phing* facilitou o uso das diversas ferramentas que estavam sendo utilizadas para garantir a qualidade do código criado. Uma vez que a ferramenta era disparada, automaticamente rodavam-se os testes unitários e o PHPCS para manter o padrão de código. Caso algum teste unitário falhasse ou o código não estivesse no padrão aceito, o código não era enviado ao repositório, cabendo então aos desenvolvedores resolver o problema antes de tentar enviar novamente o código para o repositório. Caso nenhum teste falhasse e o código estivesse no padrão aceito, então o código era imediatamente enviado ao repositório, processo esse conhecido como *commit*.

Este processo permitiu que todo o código produzido fosse integrado facilmente entre os desenvolvedores e evitou conflitos que eram comuns em projetos anteriores, quando em cada computador havia uma versão diferente do código. Todos puderam trabalhar com a mesma versão de código e rastrear, através do repositório, cada alteração feita no código, proporcionando também um histórico evolutivo do código produzido.

3.3.11 Metáfora

O objetivo da utilização da metáfora é fazer com que cliente e desenvolvedores tenham uma melhor comunicação através da utilização de uma linguagem que seja entendida

[7] http://www.phing.info

[8] https://bitbucket.org/

por ambos. Devido à natureza do projeto, uma rede social, ser de uso bastante popular e os termos comuns muitos conhecidos, a metáfora foi pouco utilizada. Mesmo assim foi necessária a criação de um glossário de termos, no qual alguns termos técnicos foram explicados através de metáforas para facilitar o entendimento por parte do cliente. Isto facilitou a participação do cliente nas conversas com os desenvolvedores.

3.3.12 Ritmo Sustentável

O horário de trabalho normal se manteve durante todo o projeto, que era das 9h às 18h. Portanto o princípio de ritmo sustentável sugerido pelo XP não foi desrespeitado, com exceção do dia em que o projeto foi ao ar. Neste dia houve a necessidade de dois desenvolvedores ficarem até mais tarde no escritório para garantir que tudo ocorresse bem e para dar suporte caso houvesse algum ajuste de última hora a ser feito. Nesta ocasião o cliente realizaria um grande evento de inauguração da Rede ABC, portanto houve o cuidado de manter estes desenvolvedores à postos para qualquer eventualidade até as 22h daquele dia.

3.3.13 Releases Curtos

Houve uma dificuldade inicial, ainda durante a fase de planejamento, para que o conceito de *releases* curtos pudesse ser entendido pelo cliente, que imaginava que seriam entregues numerosas funcionalidades após um grande espaço de tempo. Foi necessário explicar que o que aconteceria era justamente o contrário. Pequenas funcionalidades seriam entregues rapidamente, em curto espaço de tempo, e qualquer complexidade adicional seria adicionada em iterações posteriores.

Após esta fase inicial de planejamento os *releases* puderam ser planejados de maneira mais dinâmica e o curto espaço entre as entregas proporcionou ao cliente a oportunidade de ver os resultados mais rapidamente. Cada vez que uma funcionalidade era

entregue o cliente já podia imediatamente dar sua opinião, testando, aprovando ou solicitando ajustes. Com esta prática, tanto a equipe quanto o cliente tiveram confiança de que o projeto estava no caminho certo.

3.4 AVALIAÇÃO

A percepção geral da equipe envolvida neste projeto foi a de que o produto entregue foi de qualidade muito superior aos projetos anteriores. O nível de retrabalho foi mínimo e o controle sobre todo o processo de desenvolvimento foi maior.

Com o objetivo de avaliar este estudo de caso, foi aplicado um questionário composto de sete questões abertas para que os membros da equipe pudessem dar sua opinião em relação à experiência de ter participado pela primeira vez em um projeto XP.

O questionário de avaliação, apresentado no Apêndice A, é comentado a seguir.

1 - Quais são os benefícios que você enxerga foram atingidos com a utilização do XP?

Dos quatro participantes da equipe, três destacaram a qualidade do produto final. Um desenvolvedor destacou a melhoria na comunicação interna da equipe e também a comunicação entre equipe e cliente.

2 - Com a utilização do XP, quais são as mudanças que a seu ver são mais significativas?

A principal mudança destacada foi exatamente a de seguir um padrão de desenvolvimento de software, visto que até então a empresa nunca havia adotado um.

3 - Quais são as desvantagens observadas com a utilização desse processo?

Todos os quatro desenvolvedores apontaram como principal dificuldade utilizar o desenvolvimento orientado à testes, visto que era uma prática desconhecida pela maioria da equipe.

4 – Das práticas do XP que foram aplicadas tem alguma com a qual você não concorda ou que você mudaria?

Nenhum desenvolvedor citou alguma prática que gostaria mudar.

5 - O que ainda deveria ser melhorado?

Um dos desenvolvedores apontou que a possível melhoria seria explorar de maneira mais completa os recursos das novas ferramentas utilizadas. Como as ferramentas foram adotadas no início do projeto e todos tiveram que aprender durante o processo de desenvolvimento o uso das novas ferramentas foi feito de maneira básica. A empresa poderia oferecer mais treinamentos ou disponibilizar tempo dentro do horário de trabalho para que a equipe pudesse explorar melhor as ferramentas.

6 - Você considera que a qualidade do produto (sistema elaborado) ficou superior àquela esperada utilizando o processo anterior?

Todos responderam que sim.

7 - Você voltaria a usar esse modelo no próximo projeto de desenvolvimento?

Todos responderam que sim.

O cliente também se manifestou positivamente em relação à adoção do XP em seu projeto. Um dos pontos destacados pelo cliente foi a facilidade de comunicação e rapidez no *feedback*. A utilização do XP também proporcionou ao cliente um controle muito grande sobre o projeto, pois participou diariamente do desenvolvimento e de todas as fases de planejamento. Sabia-se, portanto, que desde o início o projeto estava indo para o caminho correto.

4 CONCLUSÕES E TRABALHOS FUTUROS

A adoção do XP na empresa foi um divisor de águas. Antes do seu uso os projetos eram executados sem a utilização de um padrão. Nenhum modelo de desenvolvimento de software era adotado. Em decorrência disso não havia muito controle do que era produzido, a taxa de retrabalho era alta e a equipe via-se utilizando a maioria do tempo em manutenções corretivas de projetos anteriores, tendo cada vez menos tempo para trabalhar em novos projetos. O XP foi a alternativa sugerida para mudar esta realidade.

O projeto da Rede ABC foi aquele no qual a empresa teve a oportunidade de aplicar o XP do início ao fim. Tinha-se a partir daí um rotina a ser seguida, reuniões diárias, novas ferramentas a serem utilizadas, novos conceitos a serem aprendidos e, principalmente, novos paradigmas a serem quebrados. Os desenvolvedores tiveram que abrir mão de seu individualismo em nome da equipe. Ninguém mais era dono de um código, mas sim um colaborador de um código feito por uma equipe. Todos passaram a saber o que todos estavam fazendo e o cliente participou de todo o processo.

O principal benefício da adoção do XP foi a conclusão do projeto dentro do prazo e do orçamento planejados, com uma qualidade superior aos projetos anteriores. O XP, portanto, passou a ser adotado em todos os novos projetos realizados pela empresa.

Apesar de a empresa não ter utilizado nenhum processo de desenvolvimento de software até desenvolver o projeto relatado neste estudo, há o registro de diversas informações como data de início, previsão e conclusão de cada projeto, orçamento, número de chamados técnicos para correção de *bugs* entre outros. Como trabalhos futuros estas informações poderiam ser utilizadas em um estudo comparativo no qual se confrontariam um ou mais projetos desenvolvidos sem metodologia, com projetos concluídos utilizando-se a metodologia XP. Este estudo daria uma visão quantitativa da efetividade da utilização do XP.

REFERÊNCIAS

ALEXANDRE, Mário Jesiel de Oliveira. **A construção do trabalho científico**: um guia para projetos, pesquisas e relatórios científicos. São Paulo: Forense Universitária: 2003.

ALMEIDA JÚNIOR, João Baptista de. O estudo como forma de pesquisa. In: CARVALHO, Maria Cecília M. de. **Construindo o Saber**: metodologia científica, fundamentos e técnicas. 13 ed. Campinas: Papirus, 2007.

ASTELS, David; MILLER, Granville; NOVÁK, Miroslav. **Extreme programming:** guia prático . Rio de Janeiro: Campus, 2002.

BARROS, A. J. S. e LEHFELD, N. A. S. **Fundamentos de Metodologia**: Um Guia para a Iniciação Científica. 2 Ed. São Paulo: Makron Books, 2000.

BECK, Kent. **TDD**: Desenvolvimento Guiado por Testes. Porto Alegre: Bookman, 2010.

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

CANFORA, Gerardo; CIMITILE, Aniello. **Software Maintenance.** University of Sannio, Faculty of Engineering at Benevento, Roma, 2000.

COFFIN, Rod. A Tale of Two Projects. **The Conference on AGILE 2006**. Disponível em <http://dx.doi.org/10.1109/AGILE.2006.3>. Acesso em 12 nov 2012.

CRESWELL, John W. **Projeto de pesquisa:** métodos qualitativo, quantitativo e misto . 3. ed. Porto Alegre: Artmed, 2010.

ENGHOLM JÚNIOR, Hélio. **Engenharia de Software na prática.** São Paulo: Novatec Editora, 2010.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 5 ed. São Paulo: Atlas, 2010.

IG TECNOLOGIA. **Tablet**: que bicho é esse?. Disponível em <http://tecnologia.ig.com.br/noticia/2010/01/14/tablet+que+bicho+e+esse+9295069.html>. Acesso em 12 nov 2012.

KNIBERG, Henrik. **Scrum e XP direto das fronteiras**: Como nós fazemos Scrum. Disponível em < http://www.infoq.com/br/minibooks/scrum-xp-fromthe-trenches>. 2008. Acesso em 11 ago 2012.

KUHN, Giovane Roslindo; PAMPLONA, Vitor Fernando. **Apresentando XP**: Encante seus clientes com Extreme Programming. 2009. Disponível em <http://javafree.uol.com.br/artigo/871447/Apresentando-XP-Encante-seus-clientes-com-

Extreme-Programming.html>. Acesso em 19 nov 2012.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de metodologia científica.** 7 ed. São Paulo: Altlas, 2010.

OLIVEIRA, S. R. B.; ROCHA, T. A.; VASCONCELOS, A. M.L. Adequação de Processos para Fábricas de Software. **Anais do Simpósio Internacional de Melhoria de Processosde Software – SIMPROS 2004**. São Paulo, 2004.

PRESSMAN, Roger S. **Engenharia de Software.** 6. ed. São Paulo: McGraw-Hill, 2006.

RODRIGUES, Auro de Jesus. **Metodologia Científica**: completo e essencial para a vida universitária. São Paulo: Avercamp, 2006.

SEBRAE. **Fatores condicionantes e taxas de sobrevivência e mortalidade das micro e pequenas empresas no Brasil**: 2003-2005. 2007. Disponível em <http://www.biblioteca.sebrae.com.br/bds/bds.nsf/8F5BDE79736CB99483257447006CBAD3/$File/NT
00037936.pdf>. Acesso em 11 ago 2012.

SEBRAE. **Taxa de Sobrevivências das MPEs Brasileiras.** 2011. Disponível em <http://files.provisorio.ws/empredi/1281126849349546/13191254361404223Taxa>. Acesso em 11 ago
2012.

SHORE, James; WARDEN, Shane. **The art of agile development.** Beijing: O'Reilly, 2008.

SOMMERVILE, Ian. **Engenharia de Software.** 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

TELES, Vinícius Manhães. **Extreme Programming**: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2006.

TELES, Vinicius Manhães. **Um estudo de caso da adoção das práticas e valores do Extreme Programming.** (Dissertação de Mestrado). 2005. Disponível em: <http://www.improveit.com.br/xp/dissertacaoXP.pdf> Acesso em 03 ago 2012.

THE STANDISH GROUP INTERNATIONAL, Inc. Extreme chaos. The Standish Group International, Inc, 2001. Disponível em: <http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf>. Acesso em: 14 out 2012.

UNIVERSIDADE DO SUL DE SANTA CATARINA. Pró-Reitoria Acadêmica. Programa de Bibliotecas. **Trabalhos acadêmicos na Unisul**: apresentação gráfica para TCC, monografia, dissertação e tese. Cristiane Salvan Machado et. al. (Org). 2. ed. rev. e ampl. Tubarão : Ed. Unisul, 2008.

APÊNDICES

APÊNDICE A – Questionário de Avaliação

1 - Quais são os benefícios que você enxerga foram atingidos com a utilização do XP?

2 - Com a utilização do XP, quais são as mudanças que a seu ver são mais significativas?

3 - Quais são as desvantagens observadas com a utilização desse processo?

4 – Das práticas do XP que foram aplicadas tem alguma com a qual você não concorda ou que você mudaria?

5 - O que ainda deveria ser melhorado?

6 - Você considera que a qualidade do produto (sistema elaborado) ficou superior àquela esperada utilizando o processo anterior?

7 - Você voltaria a usar esse modelo no próximo projeto de desenvolvimento?