8
Conclusão

Este trabalho comprova a viabilidade de práticas ágeis em um ambiente empresarial real. Ao mesmo tempo, ressalta os cuidados que devem ser tomados para garantir o sucesso das mudanças e da aplicação das práticas. Pode-se dizer, então, que representa um caso de sucesso da aplicação de métodos ágeis no desenvolvimento de software entre equipes pequenas.

Com uma boa preparação e algum treinamento, a utilização de *Test Driven Development* é uma solução efetiva para a melhoria do código produzido. Os conceitos de barra vermelha indicando progresso (existe um problema a ser resolvido); barra verde como uma medida evidente de sucesso; e *refactoring* para melhorar o design do código, ajudam os programadores a obter um design mais coeso e menos acoplado. Com a execução dos testes, o feedback com relação às decisões tomadas é obtido instantaneamente.

As melhorias são percebidas logo que o ambiente é preparado:

- O processo de desenvolvimento fica mais organizado. As modificações são feitas com mais segurança e são disponibilizadas para o cliente com mais freqüência e mais rapidez.
- Os programadores sabem mais claramente o que devem e o que não devem fazer. Se algum requisito está mal especificado, os programadores percebem rapidamente que existe um problema.
- Os projetos são divididos em componentes realmente reaproveitáveis. Antes da experiência, a empresa possuía apenas um componente verdadeiramente reutilizável. Depois da experiência, cinco novos componentes reaproveitáveis foram feitos.
- O controle de versões é feito de verdade. No passado, apesar de usar ferramentas como o CVS, não existia um controle real de versões e era praticamente impossível determinar as melhorias presentes em cada uma delas. Hoje, é possível obter qualquer uma das versões liberadas; listar as

melhorias e correções associadas com cada uma delas; e, se for preciso, voltar para uma versão anterior.

- As tarefas mais complicadas e estressantes – como o *build* do sistema, a execução e verificação dos testes e a integração – são feitas rapidamente e de maneira automatizada. Isso significa mais tempo para se preocupar com o que efetivamente importa – a escrita do código.

Tudo isso mesmo com o pouco tempo disponível – pouco mais de seis meses. Isso mostra que os resultados do uso deste ambiente se manifestam rapidamente.

Este trabalho pode ser proveitoso para pessoas que estejam interessadas em melhorar a forma como desenvolvem software, mas desconhecem as práticas de XP – em especial o TDD. As informações oferecidas possibilitam determinar, por exemplo, se uma empresa está pronta para dar início às mudanças. Além disso, servem de base para o planejamento das melhorias, ajudando a tornar o processo de introdução das novas práticas e ferramentas o menos incômodo possível. Isso é vital em um ambiente real de trabalho, onde não se pode perder tempo.

Pela análise de apenas algumas práticas de *eXtreme Programming*, pode-se perceber que esta metodologia é simples quando comparada com outros processos de desenvolvimento de software. São poucas práticas e a maioria delas é fácil de lembrar. Apesar disso, é preciso certo esforço para aderir às práticas e muita disciplina de todos os membros da equipe para seguir as regras inerentes a cada uma delas. Por isso, XP também pode ser considerada uma metodologia rigorosa.

8.1.Trabalhos Futuros

O uso de métricas é fundamental para determinar se o processo de desenvolvimento de software de uma empresa está trazendo benefícios ou não. A coleta dos dados para obtenção de uma métrica pode exigir disciplina das pessoas em prover informações. Um trabalho futuro é estudar formas de automatizar ao máximo a coleta de dados e pesquisar uma maneira eficiente que contribua para que as pessoas sintam-se confortáveis em fornecer os dados que não podem ser obtidos automaticamente.

Segundo pesquisas (Barry, 1999), uma maneira de evitar o desperdício de tempo é por meio do reuso. O reuso pode reduzir em até 47% o custo de um

projeto. Em XP, o *Test Driven Development* ajuda a criar soluções mais coesas e menos acopladas. A rotatividade dos pares na prática do *Pair Programming* e o ambiente comunicativo ajudam a garantir que todos saibam o que os outros programadores estão fazendo. Isso aumenta, mas não garante a chance de ocorrer reuso. Um trabalho futuro é estudar que outros mecanismos para divulgação e controle dos componentes reutilizáveis podem ajudar a assegurar o reuso de software dentro de uma corporação.

Um dos principais equívocos relacionados com TDD é achar que esta é uma técnica para testes. Apesar do nome, o TDD é uma técnica de análise e design. Para tentar esclarecer os conceitos relacionados com o TDD, surgiu o *Behaviour Driven Development* (BDD). Segundo a descrição encontrada no principal site[46] de BDD, este é um refinamento do *Test Driven Development* que, entre outras coisas, ajuda a acelerar o aprendizado das técnicas e dos benefícios oferecidos por essa forma de desenvolvimento. Por meio de um vocabulário específico, os programadores podem descrever mais claramente o comportamento de um método ou uma classe. Apesar de atualmente já existirem *frameworks* como o JBehave[47] para utilizar esta técnica em Java, estes ainda são muito prematuros e, por isso, ainda não é recomendado o seu uso em projetos reais.

O *eXtreme Programming* é uma metodologia focada na melhoria da qualidade em desenvolvimento de software. Desenvolver software, no entanto, requer outras preocupações como: gerenciamento, controle e administração. Estas atividades não estão diretamente ligadas à atividade de programação, mas são de extrema importância para o processo como um todo. O *Scrum* (Rising & Janoff, 2000) é um exemplo de metodologia ágil que trata de alguns desses outros aspectos.

[46] http://behaviour-driven.org/BehaviourDrivenProgramming

[47] http://jbehave.org/