TDD (Test Driven Development)

Izaquiel Queiroz [1], Henrique Silva[2]

Graduando SI na UFRPE-UAST. E-mail:izaquielqueiroz@live.com

Graduando SI na UFRPE-UAST. E-mail: henrique.silva.eversom@gmail.com

Resumo: Este artigo descreve Test Driven Development (TDD) um estilo de desenvolvimento de software orientado a testes, bem como sua definição e algumas pesquisas relacionadas que já foram realizadas e estão sendo realizadas atualmente.

Abstract: This article describes Test Driven Development (TDD) a test driven software development style as well as its definition and some related research that have been conducted and are currently being held.

1-**Introdução**

Este artigo descreve Test Driven Development (TDD) um estilo de desenvolvimento de software orientado a testes, bem como sua definição e algumas pesquisas relacionadas que já foram realizadas e estão sendo realizadas atualmente. A prática de TDD envolve a implementação de um sistema começando pelos casos de teste de um objeto. Escrevendo casos de teste e implementando estes objetos e métodos, surge a necessidade de outros métodos e objetos. No TDD, desenvolvedores usam testes para guiar o projeto do sistema durante o desenvolvimento. Eles automatizam estes testes para que sejam executados repetidamente. Através dos resultados (falhas ou sucessos) julgam o progresso do desenvolvimento. Os programadores fazem continuamente pequenas decisões aumentando as funcionalidades do software a uma taxa relativamente constante. Todos estes casos de teste devem ser realizados com sucesso sucessivamente antes de o novo código ser considerado totalmente implementado.

2-Revisão sistemática

Segundo Beck [Beck 2002], o TDD é um estilo de desenvolvimento de software ágil derivado do método Extreme Programming (XP) e pode ser visto como um conjunto de interações realizadas para completar uma tarefa.

Segundo NIST [NIST 2002], falhas de software são comuns e tão danosas que chegam a custar aproximadamente 60 milhões de dólares por ano. O estudo também mostra que embora nem todos os erros possam ser removidos, mais de um terço deles poderiam ser eliminados por uma infraestrutura de testes aperfeiçoada, que permitiria uma prévia e mais efetiva identificação e remoção dos erros nos softwares. De acordo com Arden Bement, diretor do National Institute of Standard and Technologies – NIST (Instituto Nacional de Padrões e Tecnologias), o impacto de erros de softwares é enorme pois todo os negócios nos Estados Unidos agora dependem de software para o desenvolvimento, produção, distribuição e suporte pós-vendas de produtos e serviços. Desenvolver softwares sem testes pode definir o sucesso ou o fracasso de um software, a perda ou ganho de um cliente importante para empresa e gerar grandes problemas como: alto acoplamento e erros após manutenção. Esses são pontos que preocupam ou pelo menos deveriam preocupar a maioria dos desenvolvedores de software ao construir um produto, seja ele para o cenário corporativo ou não.

Segundo Myers [Myers 2004] uma das principais causas da fraca abrangência de testes é o fato de muitos programadores começarem a testar com a falsa definição do termo “teste”, onde os mesmos encaram que a fase de testes é uma fase onde não se agrega valor ao software e sim uma fase onde será atestado que o software faz aquilo que deveria fazer.

O artigo de Robert Martin [Martin 2005] propõe cinco princípios de boas práticas de design de software, portanto, podemos utilizá-los no processo de refatoração do código. São eles:

- The Single Responsibility Principle (Princípio de responsabilidade única): Uma classe deve ter um, e somente um, motivo para ser alterada.
- The Open Closed Principle (Princípio aberto- fechado): Você deve ser capaz de estender o comportamento uma classe, mas não modificá-lo.
- The Liskov Substitution Principle (Princípio da substituição de Liskov): Uma classe derivada deve ser substituível por suas classes base.
- The Dependency Inversion Principle (Princípio de Injeção de Dependência): Dependa de abstrações, não de concretização.
- The Interface Segregation Principle (Princípio de segregação de interface): Codifique interfaces com fina granularidade, específicas para quem vai utilizá-las.

Segundo Koskela [Koskela 2007] Test Driven Development – TDD (Desenvolvimento orientado a testes) é uma abordagem que vem sendo largamente explorada no desenvolvimento de software. O TDD propõe uma abordagem diferente do modelo tradicional em relação aos testes. Decorrente do manifesto ágil, mais precisamente do XP (Extreme Programming) o TDD propõe a criação de testes unitários antes da codificação e não só isso, propõe que o design do software deve ser incremental e evolutivo. Com essa dinâmica, o TDD propõe que antes de qualquer código final seja adicionado, um teste unitário deve ser criado. Esse pequeno ciclo difere da forma comum utilizada ao desenvolver software no sentido de que o design da aplicação é definido no final, após vários testes e após várias refatorações de código.

Para André Cardoso (2013) desenvolvedor de PHP, Uma abordagem que vem sendo largamente explorada no desenvolvimento de software é o desenvolvimento orientado a testes (Test Driven Development - TDD). O TDD propõe uma abordagem diferente do modelo tradicional em relação aos testes, propondo a criação de testes unitários antes da codificação e não só isso, propõe que o design do software seja incremental e evolutivo.

3-Conclusão

É esperado que a equipe de desenvolvimento valorize o conceito de um bom código, que visem a qualidade interna do código e que o conceito adquirido possa auxiliar em futuras manutenções corretivas ou evolutivas. Para chegar ao cenário ideal, é esperado que a equipe utilize o TDD e que entre eles se crie uma atmosfera de compartilhamento de conhecimento. Com o uso contínuo do TDD é esperado que os desenvolvedores não só façam sistemas melhores, mas que o código desenvolvido seja fácil e de rápido entendimento por outros desenvolvedores.

4 - Referencias Bibliograficas

[Beck 2002] Beck, K. “Test-Driven Development By Example”.

[NIST 2002] National Institute of Standards and Technology- “Software Errors Cost U.S. Economy$59.5 Billion Annually”.

[Myers 2004] - Glenford J. Myers. “The Art of Software Testing. Wiley Second edition”.

[Martin 2005] R. Martin. “Principles of Oriented Object Design”.

[Koskella 2007] Lasse Koskella. “Test Driven: TDD and Acceptance TDD for Java Developers”. Manning Publications.

[André Cardoso 2013] Cardoso, André. “TDD, por que usar”?