Uma avaliação da abordagem TDD (Test Driven Development) em uma empresa desenvolvedora de software madura

Samira Ribeiro, Adriano Albuquerque, Leodércio Filho, Lobo Júnior, Sandra Régia, Niedja Cavalcante

Universidade de Fortaleza (UNIFOR) – Fortaleza – CE - Brasil

samira_ribeiro@atlantico.com.br, adrianoba@unifor.br

Instituto Atlântico – Fortaleza – CE – Brasil

leodercio_lima@atlantico.com.br, lobo@atlantico.com.br ,
carvalho_sandra@atlantico.com.br, niedja_cavalcante@atlantico.com.br

Abstract. Software development projects are increasingly seeking to identify the maximum possible errors before the final delivery of the product to the customer in order to reduce their maintenance costs of the software. This work uses the results obtained in both projects (a project that used TDD and another project that did not use TDD) to indicate that the use of TDD can result in projects with fewer failures. To support his statement in favor of TDD, the work explores two indicators, defect density in systemic test coverage and unit testing.

Resumo. Projetos de desenvolvimento de software buscam cada vez mais identificar o máximo possível de erros antes da entrega final do produto para o cliente, visando diminuir os seus custos com a manutenção do software. Este trabalho utiliza os resultados obtidos em dois projetos (um projeto que utilizou TDD e outro projeto que não utilizou TDD) para indicar que o uso de TDD pode resultar em projetos com um número menor de falhas. Para sustentar sua indicação em favor do TDD, o trabalho explora dois indicadores, densidade de defeitos em teste sistêmicos e cobertura de testes unitários.

1. Introdução

Com a grande competição existente no mercado de software, atualmente muito se investe na qualidade destes produtos para melhor atender os seus usuários. Para manter esta qualidade, as empresas estão investindo cada vez mais em testes.

Este relato tem por objetivo apresentar a experiência obtida na execução da técnica *Test Driven Development* (TDD) em um projeto real de uma organização com alto nível de maturidade. Segundo Beck (2003), o TDD é uma prática que faz parte da metodologia ágil Extreme Programming – XP. Começou a ser utilizado em 2012 por alguns projetos da organização e devido ao bom resultado apresentado por esta abordagem, está cada vez mais sendo utilizada pelos demais projetos da instituição.

O trabalho está assim organizado: a seção 2 apresenta os conceitos relacionados a TDD; a seção 3 mostra os resultados da análise comparativa entre as duas abordagens e a seção 4 apresenta as considerações finais.

2. Test-Driven-Development - TDD

Segundo Neto (2010), teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado. O seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final.

De acordo com Sakurai (2012), TDD ou desenvolvimento guiado por testes, é um conjunto de técnicas focadas em testes unitários, em que são testadas pequenas partes da aplicação. Ainda segundo Sakurai (2012), no ciclo de desenvolvimento TDD, o desenvolvedor deve criar um teste unitário que irá falhar, depois deverá implementar o mínimo necessário para que este teste unitário funcione e, finalmente, deve ser feito uma refatoração na implementação para remover qualquer duplicação no código e melhorar o código de forma que o mesmo possa executar as funcionalidades que devem ser executadas por ele. A implementação do teste unitário que irá falhar permitirá ao desenvolvedor ter um objetivo rápido: Fazer o teste funcionar implementando o mínimo necessário. A refatoração irá melhorar a qualidade do código, pois algumas vezes a implementação mínima para fazer o teste funcionar possui muitos códigos duplicados ou não utilizou da melhor forma possível os conceitos da orientação a objetos. O desenvolvedor vai aprendendo a programar, de forma evolutiva, na primeira vez que cria uma funcionalidade, pode ter criado de forma correta e funcional, mas na segunda vez que for criar ou modificar esta mesma funcionalidade é possível encontrar uma forma outra de realizar a mesma implementação, às vezes até melhorando a versão inicial. O objetivo final deste ciclo é que a aplicação possa ser implementada com um código limpo, com boas utilizações do conceito de orientação a objetos e permitindo que uma fácil manutenção e expansão da aplicação.

3. Análise Comparativa

Com o intuito de identificar a real eficiência da prática de TDD no processo de desenvolvimento de software, foi realizado um comparativo de indicadores de qualidade em dois projetos com características similares e desenvolvidos pelo Instituo Atlântico, que é uma instituição de pesquisa e desenvolvimento sediada em Fortaleza e com filial em São Paulo. A sede conta com aproximadamente 120 colaboradores. Desde a sua fundação, o Instituto Atlântico iniciou um amplo programa de qualidade. Atualmente os processos da organização estão aderentes à norma ISO 9001 e ao nível 5 do modelo CMMI-DEV 1.2.

O primeiro projeto, denominado "Projeto1", utilizou TDD e tinha dois objetivos: (i) estender um serviço web que permite aos usuários publicar e distribuir conteúdo para negócios ou uso pessoal, como por exemplo: uma publicação impressa com qualidade profissional ou uma publicação digital para visualização móvel ou online; e (ii) disponibilizar uma API (Application Programming Interface) pública que visa facilitar o acesso de empresas parceiras à plataforma de impressão sob demanda ao serviço web

descrito anteriormente. Além dos profissionais de gestão (como gerente técnico e coordenadores), este projeto possuía três analistas de sistemas, um analista de requisitos, um analista de teste e um especialista de interface. A modalidade do projeto é "Fábrica de Solução", que se caracteriza pelo fato do projeto trabalhar com a equipe do cliente desde a etapa inicial de avaliação de necessidades e concepção de requisitos. Um analista de sistema da equipe é muito experiente em TDD e o outro dois pouco experiente. O projeto utilizou como ciclo de vida, o processo ágil de desenvolvimento.

Já o segundo projeto, denominado "Projeto 2", não utilizou TDD e tinha como objetivo viabilizar um projeto de P&D para desenvolvimento de um software para automação e gestão de clínicas radiológicas e automatização da manipulação das imagens utilizadas em seus diagnósticos. Além dos profissionais de gestão (como gerente técnico e coordenadores), este projeto possuía três analistas de sistemas, um analista de requisitos e um analista de teste. A modalidade do projeto também era "Fábrica de Solução" e utilizou em seu o ciclo de vida o processo ágil de desenvolvimento.Vale ressaltar que o período planejado para cada uma das *sprints* dos dois projetos era de um mês.

Ressalta-se que todos os processos da organização são aplicáveis aos projetos. A adaptação do processo organizacional para geração do processo definido no projeto ocorre na fase de planejamento e deve considerar o ciclo de vida adotado e o porte do projeto. Nos gráficos apresentados na próxima seção, teremos os limites de baseline (vermelhos) calculados a partir da amostra coletada pela organização (Base Histórica) e limites de especificação (verdes) conforme as metas estabelecidas para o projeto. A base histórica relacionada ao processo de testes sistêmicos é estratificada, levando em consideração as características do projeto e sendo classificados como: Densidade de Defeitos (projetos não ágeis), DDTS Modalidade Fábrica de Software, DDTS Modalidade Fábrica de Sistemas e DDTS Modalidade Fábrica de Solução. Cada baseline de desempenho da organização possui uma versão que geralmente é atualizada a cada 3 meses, quando uma nova baseline de processos é disponibilizada e a base histórica é atualizada, levando em consideração ao últimos dados revisados e registrados na base histórica organizacional. Os dois projetos fazem parte da mesma baseline de testes sistêmicos sendo classificados como “DDTS Modalidade Fábrica de Solução”, atualmente na versão 1.2. O acompanhamento estatístico dos projetos de desenvolvimento é realizado a partir da Ferramenta de Acompanhamento de Projetos. As versões das baselines a serem utilizadas pelo projeto são selecionadas automaticamente pela ferramenta ao ser informada a versão da baseline de processos que o projeto irá utilizar. Cada versão de processo liberada possui versões de modelos e baselines associadas. As coletas acontecem após a conclusão da sprint do projeto, sendo responsabilidade do gerente de projeto coletar e analisar, através do “Relatório de Controle do Projeto” (RDP). Este relatório é validado pelo analista da qualidade (QA), a fim de atestar a consistência das coletas e análise e apresentado pelo coordenador em uma reunião com o gerente e participação do QA. No RDP são acompanhados diversos indicadores, dentre eles um que monitora a maturidade da equipe. Este indicador leva em consideração os seguintes fatores: Familiar com os processos da organização, Experiência com a aplicação em desenvolvimento, Liderança Técnica/Coordenação, Motivação e Integrantes em tempo parcial. Cada fator possui um peso que indica a importância do fator para a maturidade da equipe. A maturidade da equipe varia entre 0

e 22,5. Maturidade “0” significa que a equipe tem baixa maturidade e 22,5 uma equipe representa uma equipe com alta maturidade. Após definidos os fatores, estes só poderão ser mudados caso não haja um atendimento aos limites organizacionais do projeto. As duas equipes que foram utilizadas apresentam maturidade próximas, para a equipe que não utilizou o TDD apresentou uma maturidade de 19,00 enquanto que a equipe que utilizou TDD apresentou uma maturidade 18,50.

3.1 Análise Quantitativa – Indicadores de Qualidade

Para facilitar a análise e a compreensão das vantagens do TDD, seguem abaixo os indicadores de qualidade de desenvolvimento de software dos dois projetos, que estão representados graficamente nas Figuras 1, 2, 3 e 4.

I. Densidade de defeitos em teste sistêmicos (DDTS)

- **Projeto 1 (Com TDD):**

Data da Coleta	Baseline de release *	Tamanho	Tamanho Regressão	Quantidade total de defeitos	DD Baseline	DD Blocker	DD Critical
25/01/2013	papi_s4_release_1	23,00	144,00	2,00	0,01	0,00	0,00
22/10/2012	papi_s1_release_1	27	171	8,00	0,06	0,00	0,00
05/12/2012	papi_s2_release_1	26	197	5,00	0,04	0,00	0,00
10/01/2013	papi_s3_release_1	25	222	5,00	0,04	0,00	0,00
25/01/2013	papi_s4_release_1	23	144	2,00	0,01	0,00	0,00

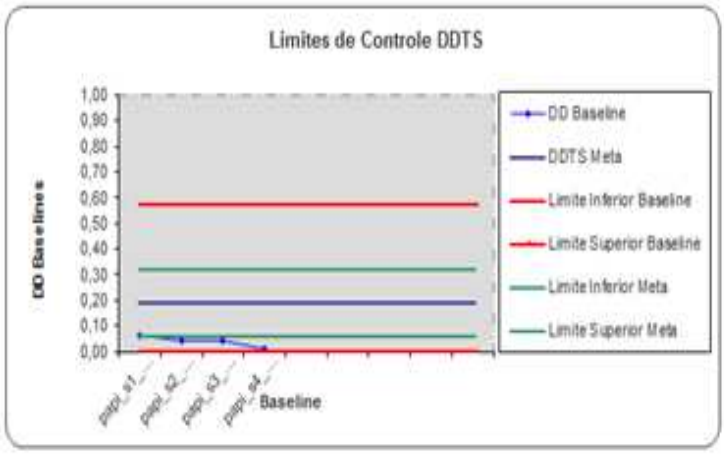

Figura 1. Indicador DDTS do Projeto 1

Análise do Indicador: Dentre os bugs de testes sistêmicos identificados, nenhum foi considerado crítico. Como a abordagem de testes utilizada pelo projeto, envolve todas as funcionalidades desenvolvidas (testes de regressão), garante-se uma maior cobertura nos testes e consequente satisfação do cliente. De acordo com a estratégia utilizada, é esperado que o número de bugs seja reduzido ao passar das *sprints*, portanto o indicador poderá ser influenciado, nesse caso com uma considerável redução. Com o reflexo da boa cobertura dos testes unitários e da integração contínua do projeto, o indicador de "Densidade de Defeitos em Testes Sistêmicos" tem sempre apresentado uma baixa quantidade de defeitos em relação ao tamanho que foi testado, ressaltando-se que o tamanho total do que foi testado na *sprint* é referente ao tamanho da *sprint* somado ao

mais o tamanho da regressão. O indicador sempre apresentou um bom resultado, ficando sempre abaixo do limite mínimo definido.

- **Projeto 2 (Sem TDD):**

Data da Coleta	Baseline de release *	Tamanho	Tamanho Regressão	Quantidade total de defeitos	DD Baseline	DD Blocker	DD Critical
7/16/2012	DABI_06_release_01	50.30	0.00	24.00	0.48	0.00	0.06
2/24/2012	DABI_01_release_01	60.90		46.00	0.76	0.00	0.13
3/26/2012	DABI_02_release_02	59.92		34.00	0.57	0.00	0.03
5/2/2012	DABI_03_release_01	56.82		31.00	0.55	0.00	0.05
5/11/2012	DABI_04_release_01	199.1		42.00	0.21	0.01	0.03
6/6/2012	DABI_05_release_01	55.27		10.00	0.18	0.00	0.00
7/16/2012	DABI_06_release_01	50.30		24.00	0.48	0.00	0.06

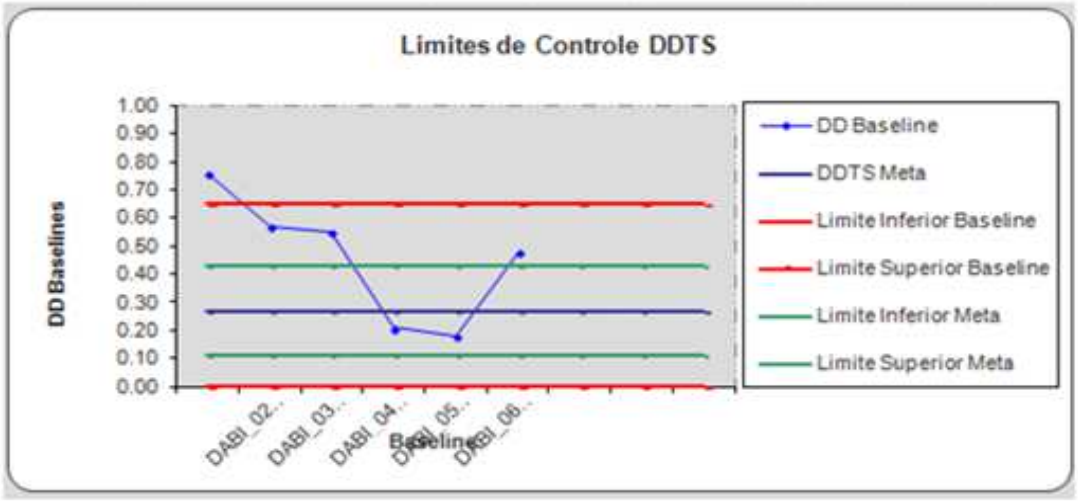

Figura 2. Indicador DDTS do Projeto 2

Análise do Indicador: Pode-se perceber que apenas durante duas *sprints* o projeto ficou dentro dos limites definidos, nas demais *sprints* o projeto não apresentou um bom resultado ficando acima do limite superior. Ressalta-se que não foram considerados defeitos referentes a testes de regressão e que o tamanho do que foi testado é referente apenas ao tamanho da *sprint*.

II. Cobertura de Teste Unitário – CTU

- **Projeto 1 (Com TDD)**

Data da Coleta	Mês/Release	CTU
25/01/2013	papi_s4_release_1	82,9%
22/10/2012	papi_s1_release_1	88,0%
05/12/2012	papi_s2_release_1	89,4%
10/01/2013	papi_s3_release_1	89,3%
25/01/2013	papi_s4_release_1	82,9%

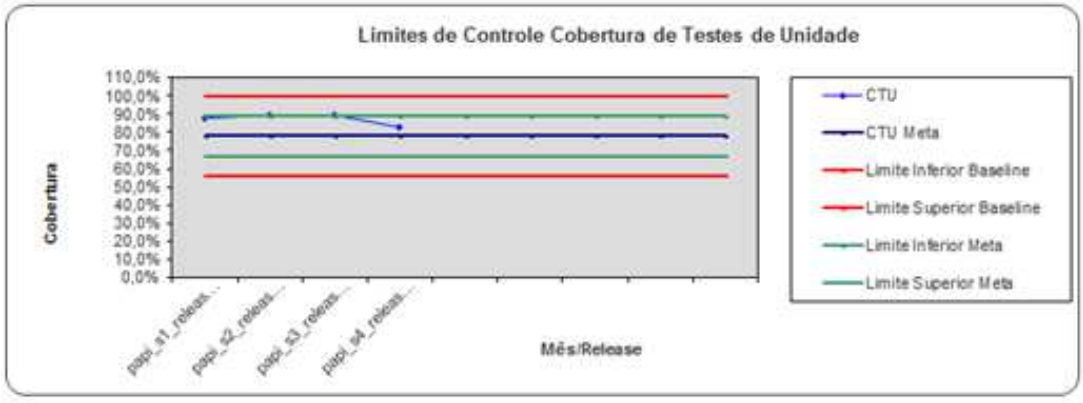

Figura 3. Indicador CTU do Projeto 1

Análise do Indicador: Como pode ser visto no gráfico, o indicador CTU se manteve dentro dos limites definidos para o projeto, refletindo um valor de 82,9% na cobertura do código. A cobertura de teste contemplou 72,4% das classes, 85,1% dos métodos e 82,0% das linhas de código. Ressalta-se que o projeto possuía 100% de cobertura de testes de aceitação automatizados. A média da cobertura de testes unitários ao longo do projeto foi de 85,8%, sendo um bom resultado para os testes realizados. Em nenhum momento o projeto apresentou um resultado inferior ao limite mínimo definido.

- **Projeto 2 (Sem TDD):**

Data da Coleta	Mês/Release	CTU
7/16/2012	DABI_06_release_01	85.0%
2/24/2012	DABI_01_release_01	81.0%
3/26/2012	DABI_02_release_02	79.0%
5/2/2012	DABI_03_release_01	74.5%
5/11/2012	DABI_04_release_01	74.5%
6/8/2012	DABI_05_release_01	81.0%
7/16/2012	DABI_06_release_01	85.0%

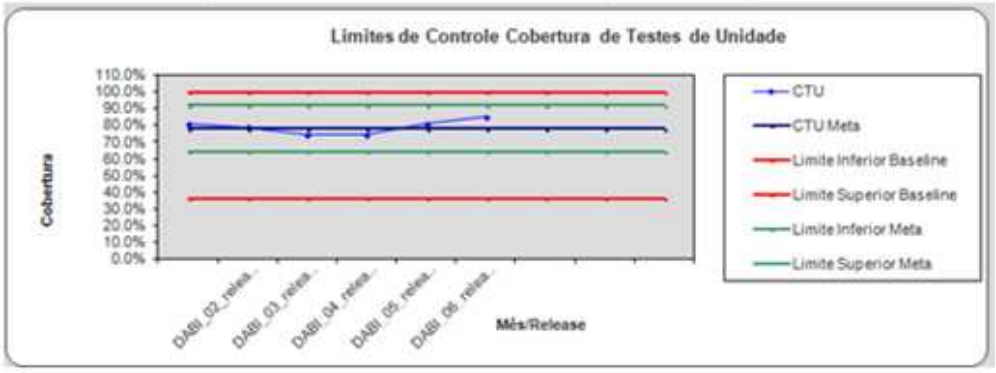

Figura 4. Indicador CTU do Projeto 2

Análise do indicador: O CTU ficou dentro dos limites estabelecidos para o projeto, apresentando uma cobertura de 85% na última coleta. A cobertura de teste contemplou 76,4% das classes, 88,1% dos métodos e 86,0% das linhas de código. A média da cobertura de testes unitários foi 79% ao final do projeto.

3.2 Análise Qualitativa

Em relação ao Projeto 2 (o que não utiliza as práticas TDD), os testes foram realizados da forma tradicional, ou seja, no final da *sprint*. Esta é uma situação que pode contribuir para um cenário de teste com pouca qualidade, baixa cobertura ou até mesmo o teste pode não ser realizado por falta de tempo, por ter sido deixado para o final do ciclo de desenvolvimento. Além disso, pode acontecer também da estimativa de teste não prever o tempo real necessário para execução destes testes. Com isso acontecia uma situação que parecia ser favorável, mas não era, ou seja, as *sprints* iam sendo executadas com uma velocidade maior, porém a qualidade do teste não era tão boa. Com isso, as próximas *sprints* do projeto eram mais lentas, por conta do retrabalho e da necessidade de alocar mais tempo para os testes. Ou seja, os *bugs* iam surgindo gradativamente nas ultimas *sprints*, quando o custo, a complexidade do software e o impacto das mudanças eram maiores. Percebeu-se também, que esta forma de executar teste também ajudou para que o cliente identificasse um maior numero de *bugs*, o que não é bom para a imagem do produto e nem para o custo do projeto, conforme Myers (1979). Neste caso é importante destacar também que diferentemente do TDD, um item de *backlog* para implementar um caso de uso é composto por: especificação + codificação + teste unitário + teste integração + teste sistêmico, ou seja, a codificação é estimada isoladamente dos testes e , com isso os testes são sempre deixados por ultimo.

Isto podia induzir a equipe a ter uma ideia equivocada sobre a evolução do projeto, interpretando que o conceito de "Done", é apenas o código implementado, podendo não priorizar os testes e produzindo assim um software com pouca qualidade.

Já no Projeto 1, aonde foi utilizado o TDD, a estimativa da implementação já contemplava o tempo de teste, ou seja, o conceito de "Done" para a codificação era "código feito mais código testado". Se uma das duas atividades não tivesse sido executada, então a implementação não estava concluída. Isto fazia com que a velocidade das *sprints* fosse menor, porém estável, pois quase sempre consumiam o mesmo tempo, portanto mais fácil de estimar e, além disso, contribuía para que o teste não fosse despriorizado e deixado para o final do ciclo de desenvolvimento. Com isso percebemos que com as práticas TDD a velocidade do time era menor, mas em compensação a qualidade do produto era maior, garantida pelo foco nos testes ainda na fase de desenvolvimento. Outro ponto positivo é que, como o teste unitário é executado em tempo de implementação, ao realizar qualquer mudança no código, dá para identificar naquele mesmo momento onde o erro ocorreu, pois o código irá indicar isso na ferramenta destinada ao teste unitário.

Considerações Finais

Através deste trabalho foi possível tirar algumas conclusões com base nos resultados aqui apresentados e na pesquisa realizada sobre o assunto. São elas: O projeto que utilizou TDD conseguiu reduzir a quantidade de problemas encontrados nos testes sistêmicos, tendo apresentado também uma boa cobertura nos testes unitários; Embora no começo da execução das atividades de implementação a produtividade do time não seja tão significativa devido à utilização da técnica, percebemos que ao longo do projeto essa produtividade tende a crescer, pois a equipe já tem obtido um bom conhecimento da técnica; Em relação ao custo e beneficio, com o TDD é possível realizar mudanças no código sem causar grandes impactos no sistema. Por outro lado, é consumido muito tempo para implementar, já que o teste sempre deve ser realizado.

Com o objetivo de enriquecer a pesquisa, para trabalhos futuros pretende-se utilizar mais indicadores de qualidade como Produtividade e Retrabalho e mais projetos para realizar esta comparação com o intuito de enriquecer a pesquisa. No geral foi possível concluir que TDD é uma prática bastante promissora, pois trata o sistema na "raiz", achando o maior numero de defeitos no inicio do desenvolvimento, isto agrega qualidade ao produto e sai bem mais em conta para projeto, já que defeitos são identificados mais cedo.

Referências

Beck, Kent. Test-Driven Development: A Pratical Guide, 2003.

MYERS, Glenford J., The Art of Software Testing, 1979

Neto, Arilo Claudio Dias. Introdução a Teste de Software, 2012.

Sakurai, Rafael. TDD - Desenvolvimento guiado por testes, 2012.