On the Effectiveness of Unit Tests in Test-driven Development

Ayse Tosun, Burak Turhan, Muzamil Ahmed e Natalia Juristo

Augusto Calado Bueno

São Paulo
2021

Sumário

Descrição do Artigo

O artigo *On the Effectiveness of Unit Tests in Test-driven Development* de Ayse Tosun, Burak Turhan, Muzamil Ahmed e Natalia Juristo foi escolhido para o trabalho final da disciplina Qualidade de Software - TSW-006/2021-3. Este artigo aborda o tema da utilização da abordagem de desenvolvimento de software *Test-Driven Development* (TDD) e sua real efetividade em termos de criação de cenários de teste que são capazes de encontrar defeitos.

Introdução

A abordagem de *test-driven development* (TDD), foi introduzida nos anos de 1960 pela NASA como uma prática de desenvolvimento de software. Essa abordagem incentiva pensar na funcionalidade antes de se começar a escrever o código, ter refatorações frequentes e execução de teste de regressão de forma recorrente, e por esse motivo é vista como uma prática adequada para o desenvolvimento de software que utiliza métodos ágeis.

A abordagem TDD ainda não está totalmente adotada como prática de desenvolvimento padrão da indústria [3], pois, de acordo com Causevic et al [4]. existem alguns fatores, como falta de experiência/conhecimento sobre TDD, aumento do tempo de desenvolvimento e falta de habilidades em escrever cenários de teste o que dificultam a adoção.

Há diversos estudos tentaram investigar os efeitos dessa abordagem na qualidade interna do código, na qualidade externa e produtividade de desenvolvimento. Alguns estudos indicam que essa abordagem possui efeitos inconsistentes na produtividade [6][7], mas outros indicaram que há ganhos na qualidade externa [8].

Há ainda estudos que buscam entender a relação do efeito do TDD em outros atributos do software, como qualidade dos testes produzidos. Entretanto, estes estudos são menos frequentes que os voltados à qualidade de código e produtividade. Tais estudos, buscam verificar a capacidade dos testes de unidade de detectar defeito, uma vez que a codificação é conduzida, principalmente, por casos de teste. Portanto, espera-se que o impacto imediato do TDD em comparação com uma abordagem de teste final tradicional (*test last development*) seja em casos de teste de unidade.

A Fim de analisar o impacto do TDD o artigo *On the Effectiveness of Unit Tests in Test-driven Development*, realiza uma análise empírica para verificar seus efeitos na eficácia dos cenários de teste de unidade em um experimento realizado com participantes provenientes da indústria.

Objetivo

O Objetivo do artigo consiste em investigar o real impacto da abordagem *test-driven development* (TDD) na efetividade dos cenários e casos de teste de unidade em comparação com a abordagem de *test last development* (TLD) em um contexto industrial.

Resumo

Métricas

Os autores do artigo discorrem sobre a evolução do que é considerado a métrica mais efetiva para medir qualidade nos testes ao longo do tempo. Inicialmente, as técnicas de teste de software focaram na definição e obtenção de um nível mais alto de cobertura de código (*code coverage*).

Cobertura de código (*code coverage*), indica qual parte do código (por exemplo, linha, instruções e ramificações) é executada durante o teste. Há diversas formas de se medir a cobertura de código, como por exemplo *Branch coverage e Method coverage*. Ainda com relação a cobertura de código, as referências utilizadas pelos autores [9] afirmam que essa métrica não indica, necessariamente, a eficácia de um conjunto de testes; em vez disso, mede a penetração/profundidade (thoroughness) de um conjunto de testes.

Depois, o foco da voltou-se para técnicas de teste de software baseadas em defeito, como o *mutation score* que é calculado através da aplicação da prática de teste *mutation testing que* é uma forma de medir a capacidade de um caso de teste de detectar defeitos no código. Para verificar o quão bom é um conjunto de testes, várias versões do código do programa, nas quais diferentes tipos de defeitos são injetados, podem ser criadas.

O *mutation score* é calculado como a porcentagem de defeitos (mutantes) detectadas pelo conjunto de testes e pode ser usada para medir a eficácia de um conjunto de testes em termos de sua capacidade de detectar defeitos.

Pesquisas de Referência

A pesquisa publicada no artigo *On the Effectiveness of Unit Tests in Test-driven Development* estende três estudos realizados anteriormente que buscaram verificar o impacto da abordagem de desenvolvimento de software na qualidade dos cenários de teste de unidade em detectar defeitos. Esses estudos são:

- A. Causevic, D. Sundmark, and S. Punnekkat. 2012. Test case quality in test driven development: A study design and a pilot experiment. In 16th International Conference on Evaluation & Assessment in Software Engineering (EASE).

- A. Causevic, D. Sundmark, and S. Punnekkat. 2012. Quality of testing in test driven development. In the Eighth International Conference on the Quality of Information and Communications Technology (QUATIC).

- L. Madeyski. 2010. The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment. Information and Software Technology 52, 2 (2010).

O estudo realizado por Madeyski em *The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment*, foi realizado no âmbito acadêmico e contou com 22 estudantes de mestrado como participantes da pesquisa. As métricas adotadas no experimento foram *branch coverage*, visando medir a abrangência dos testes produzidos, e *mutation score* para medir a eficácia dos testes de unidade. A variável independente do estudo foi a abordagem de desenvolvimento de software com dois tratamentos, *test-first* (TF) e *test-last* (TL). Os resultados do estudo não revelaram diferenças significativas ao comparar o *mutation score* e o *branch coverage* dos dois tratamentos.

O estudo de Madeyski foi estendido por Causevic et al. em outras duas publicações. Os participantes da pesquisa também foram alunos de mestrado (14 participantes), as métricas dos experimentos foram *statement coverage*, *mutation score* para medição de qualidade interna do teste. Para medição da qualidade externa do teste foram utilizados as métricas *defect detection ability* e *failing assertion.* A variável independente da pesquisa se manteve a mesma, TF e TL.

Formato do Experimento

O experimento contou com a participação de 24 profissionais desenvolvedores de software vinculados a uma corporação multinacional que provê serviços de segurança para a vida digital de pessoas e negócios. Esses profissionais não tinham experiência com TDD.

A metodologia adotada pelos autores do artigo para a realização da pesquisa foi a experimentação. A variável independente do estudo foi a abordagem de desenvolvimento de software com dois tratamentos. O primeiro é o *test driven development* (TDD) e o segundo é o *incremental test last development* (ITLD). Os autores comentam que o ITLD foi adotado, pois de acordo com [13], essa abordagem força um controle que envolve a realização de testes, fazendo com que a comparação seja menos tendenciosa e mais justa.

Como variável dependente, os autores adotaram a efetividade dos cenários de teste que tem como métrica o *mutation score* que, de acordo com as referências utilizadas pelos autores, é uma métrica frequentemente associada à medição da

efetividade de cenários de teste [1, 2, 4, 9]. Outras duas métricas adotadas para medir a abrangência dos testes foram *branch coverage* e *method coverage. Mutation score* e *branch coverage* foram escolhidos para confirmar ou refutar os resultados obtidos pelas pesquisas referentes (Madeysky e Causevic et al.); já o *method coverage* foi escolhido para verificar uma hipótese levantada pelos autores de que o TDD leva a construção de menos métodos mas com mais ramificações.

Para ambos os tratamentos (TDD e ITLD), foram aplicados os seguintes passos - divisão/decomposição da especificação em pequenas tarefas de programação, codificação, teste e refatoração). Como objeto da pesquisa, dois programas foram escolhidos para os participantes implementarem. O primeiro é o *MarsRover API* que deveria ser desenvolvido usando ITLD, e o segundo é o *Bowling Score* que deveria ser desenvolvido utilizando o TDD.

Ambos os programas utilizam Java como linguagem de programação, JUnit como framework para implementação de teste unitário, o framework Judy como ferramenta de mutação para código fonte java e EclEmma para realizar as medições de cobertura de código.

Os participantes do experimento foram expostos às duas tarefas de implementação, pois, de acordo com os autores, essa abordagem de experimento foi escolhida, devido desejo dos voluntários de participar em todos os eventos (exercícios e treinamentos), portanto, não houve o agrupamento de participantes em dois grupos como nos outros estudos referências de Madeysky e Causevic et al.

Com relação ao ambiente de aplicação das atividades propostas pela pesquisa, os autores forneceram aos participantes máquinas virtuais previamente configuradas com todos os programas necessários para a execução do experimento.

Os procedimentos de análise utilizados foram média, mediana, desvio padrão, valores máximo e mínimo das métricas mutation score e code coverage.

Para avaliar as hipóteses da pesquisa, os autores utilizaram o teste da soma dos postos de Wilcoxon. Como justificativa para a aplicação desse teste e não o teste t de Student foi que os valores de mutation score extraídos do código fonte dos participantes da pesquisa não seguem uma distribuição normal.

Resultados

Nem todos os participantes conseguiram realizar todas as tarefas do experimento devido a erros de compilação, códigos de teste de unidade não executando com sucesso e falta de código de teste de unidade nas implementações de alguns participantes;

Estatística Descritiva

Mutation Score						
	Número Participantes	**Min.**	**Mediana**	**Média**	**Max.**	**Desvio Padrão**
ITLD	13	2	**62**	**54.7**	83.0	27.9
TDD	18	6	**84**	**70.6**	93.0	28.4

Tabela 1 - Mutation Score

Analisando a tabela acima, os autores observaram que os participantes escreveram testes de unidade mais eficazes na detecção de defeitos (injetadas no código fonte pela ferramenta Judy) em TDD (mediana do mutation score obteve valor de 84% e 71% média), em comparação com ITLD (62% na mediana e 55% na média).

No intuito de realizar uma comparação por pares entre os valores de *mutation score* de ambos os tratamentos (TDD e ITLD), os autores geraram uma outra tabela contendo apenas os participantes que conseguiram realizar todas as tarefas para as duas abordagens.

Mutation Scores (Comparação por pares)						
	Número Participantes	**Min.**	**Mediana**	**Média**	**Max.**	**Desvio Padrão**
ITLD	**10**	17	**67**	**57.5**	83.0	26.17
TDD	**10**	56	**87.5**	**83.5**	93.0	10.69

Tabela 2- Mutation Score (Comparação por pares)

A análise dos autores com relação a tabela 2, indica que o TDD pontua melhor em termos de *mutation score*, ou seja, valores medianos e médios mais altos e desvio padrão mais baixo. Isso indica que os casos de teste de unidade escritos em TDD podem ser capazes de detectar mais defeitos do que os casos de teste de unidade escritos em ITLD.

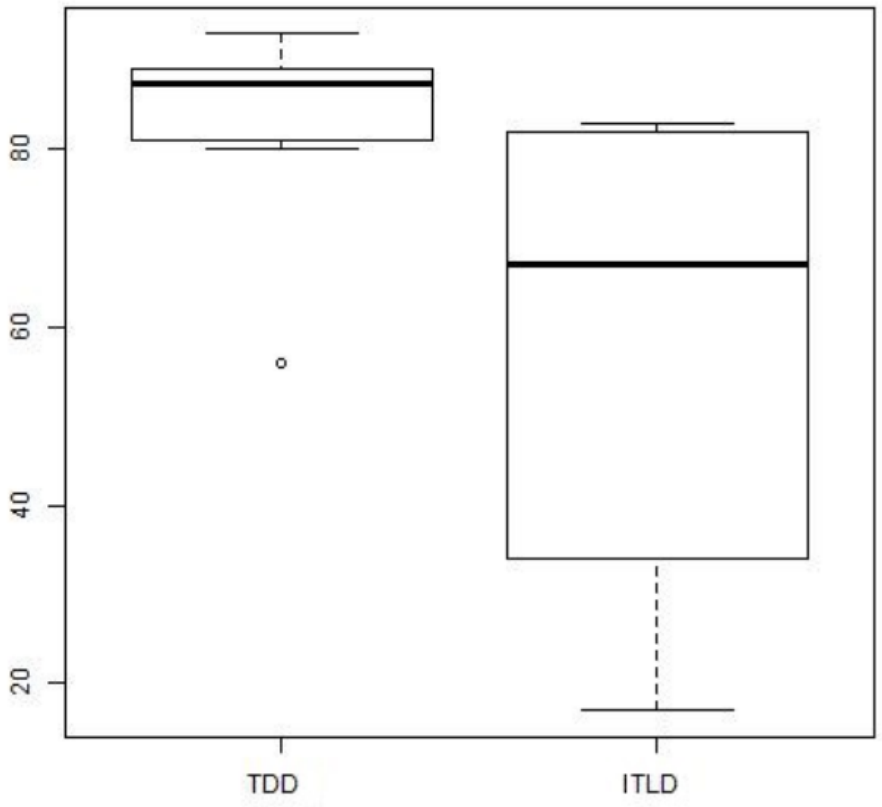

Imagem 1 - Boxplot mutation score

Cobertura de Código - *Branch Coverage*					
	Min.	**Mediana**	**Média**	**Max.**	**Desvio Padrão**
ITLD	0	**32.5**	**37.5**	96.4	33.8
TDD	0	**89.2**	**59,8**	100	45.6

Tabela 3 - Cobertura de Código Branch Coverage

A partir dos resultados obtidos para a cobertura de código que utiliza o *branch coverage*, é possível verificar que aproximadamente 37% dos ramos são cobertos pelo ITLD, enquanto que 60% dos ramos são cobertos utilizando o TDD.

Cobertura de Código - *Method Coverage*					
	Min.	**Mediana**	**Média**	**Max.**	**Desvio Padrão**
ITLD	11.1	**87.9**	**79.1**	100	23.8
TDD	0	**70.7**	**61.4**	100	30.8

Tabela 4 - Cobertura de Código Method Coverage

Já os resultados para a cobertura de código utilizando o *method coverage*, nota-se que mais métodos são cobertos utilizando a abordagem ITLD.

Os autores do artigo comentam que os resultados obtidos através dos experimentos divergem com os resultados dos estudos realizados por Causevic et al. e Madeyski que não identificaram grandes diferenças dos valores obtidos para a o *branch coverage* variando a abordagem de desenvolvimento (*test-driven development* ou *test-last development*).

Validações Estatísticas - Teste das Hipóteses

Hipóteses - Mutation Score
• $H0\mu(MS)_{TDD} = \mu(MS)_{ITLD}$: Não há diferença no mutation score (MS) dos testes de unidade implementados pelos sujeitos durante TDD e ITLD • $H1\mu(MS)_{TDD} != \mu(MS)_{ITLD}$: Há uma diferença significativa no mutation score dos testes de unidade implementados pelos sujeitos durante TDD e ITLD • Nível de Significância = 0.05

• Mutation Score
  ○ p-valor = 0.04
  ○ **rejeitada**

Os resultados do teste de soma de classificação de Wilcoxon entre as abordagens ITLD e TDD no *mutation score* mostram que há uma diferença significativa entre a eficácia dos testes de unidade medidos nas duas abordagens (p - valor = 0,04). Um *mutation score* mais alto mostra uma capacidade de detecção de defeitos mais alta. No experimento, os casos de teste de unidade escritos em TDD conseguiram ter uma detecção de defeitos maior que os casos de teste escritos em ITLD.

Hipóteses
• H0μ(BRCov)TDD = μ(BRCov)ITLD: Não há diferença entre TDD e ITLD para o branch coverage (BRCov). • H0μ(MTCov)TDD = μ(MTCov)ITLD: Não há diferença entre TDD e ITLD para a cobertura de métodos (MTCov). • Nível de Significância = 0.05

• Mediana BRCov
  ○ p-value = 0.033
  ○ **rejeitada**
• Mediana MTCov
  ○ p-value = 0.044
  ○ **rejeitada**

Os Autores do artigo, através dos resultados da aplicação da soma de classificação de wilcoxon, validaram a hipótese de que TDD tem um efeito positivo no *branch coverage* em relação ao ITLD. Porém, para o *method coverage*, o ITLD se mostrou melhor, pois cobre uma quantidade significativamente maior de métodos.

Ameaças para a Validade dos Resultados

- Wilcoxon utilizado para não forçar nenhuma suposição de distribuição para os dois experimentos.
- Utilização de mutation score e cobertura de código para garantir a confiabilidade das medidas
- Para evitar o risco de haver diferentes implementações entre participantes, os autores promoveram um ambiente padronizado para os participantes realizarem as implementações
- Não compartilhamento das hipóteses e dados obtidos obtidos entre as fases de teste aplicando a abordagem TDD e ITLD para não influenciarem o comportamento dos participantes.

Conclusão

Os resultados obtidos através do experimentos são diferentes de estudos anteriores realizados em ambientes acadêmicos. Os participantes conseguiram realizar testes de unidade mais eficazes com o TDD. Além disso, a escrita de casos de teste tiveram um *branch coverage* maior quando utilizado TDD. No entanto, os casos de teste escritos utilizando ITLD cobrem mais métodos do que aqueles escritos com TDD.

Por fim, os autores comentam que para pesquisas futuras, um contexto mais amplo e mais participantes poderiam estar envolvidos, e análises mais detalhadas quanto ao tipo de teste de unidade que costuma ser mais desenvolvido quando a abordagem de desenvolvimento varia.

Referências

[1] ANDREWS, J.H.; BRIAND, L.C. e LABICHE, Y.. 2005. Is mutation an appropriate tool
for testing experiments?. Em 27 - International Conference on Software Engineering (ICSE).

[2] DELAMARO, M. E. e OFFUTT, J.. 2014. Assessing the Influence of Multiple Test Case
Selection on Mutation Experiments. Em IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops. 171–175.

[3] GOPINATH, R.; JENSEN, C.; e GROCE, A.. 2014. Code coverage for suite evaluation
by developers. Em 36 - International Conference on Software Engineering (ICSE)

[4] CAUSEVIC, A.; SUNDMARK, D.; e PUNNEKKAT, S.. 2011. Factors Limiting Industrial
Adoption of Test Driven Development: A Systematic Review. Em 4 - IEEE International Conference on Software Testing, Verification and Validation.

[5] JIA, Y. e HARMAN, M.. 2011. An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering,
649–678. https://doi.org/10.1109/TSE.2010.62

[6] MUNIR, HUSSAN; MOAYYED, MISAGH; e PETERSEN, KAI.. 2014. Considering Rigor and
Relevance when Evaluating Test Driven Development: A Systematic Review. Inf. Softw. Technol. 56, 4 (Abril 2014), 375–394. https://doi.org/10.1016/j.infsof.2014.01.002

[7] TURHAN, BURAK; LAYMAN, LUCAS; DIEP, MADELINE; SHULL, FOREST; e ERDOGMUS, HAKAN..
2010. How Effective is Test Driven Development, Chapter Making Software: What Really Works, and Why We Believe It.

[8] RAFIQUE, YAHYA; e MISIC, VOJISLAV B.. 2013. The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. IEEE Transactions
on Software Engineering, 835–856. https://doi.org/10.1109/TSE.2012.28

[9] MADEYSKI, L.. 2010. The impact of Test-First programming on branch coverage and
mutation score indicator of unit tests: An experiment. Information and Software Technology.

[10] CAUSEVIC, A.; SUNDMARK, D.; e PUNNEKKAT, S.. 2012. Test case quality in test driven development: A study design and a pilot experiment. Em 16 - International Conference on Evaluation & Assessment in Software Engineering (EASE).

[11] CAUSEVIC, A.; SUNDMARK, D.; e PUNNEKKAT, S.. 2012. Quality of testing in test driven development. Em 8 - International Conference on the Quality of Information and Communications Technology (QUATIC).

[12] TOSUN, AYSE; TURHAN, BURAK; AHMED, MUZAMIL; e JURISTO, NATALIA. 2018. On the Effectiveness of Unit Tests in Test-Driven Development;

[13] TOSUN, AYSE; DIESTE, OSCAR; FUCCI, DAVIDE; VEGAS, SIRA; TURHAN, BURAK; ERDOGMUS, HAKAN; SANTOS, ADRIAN; OIVO, MARKKU; TORO, KIMMO; JARVINEN, JANNE; e JURISTO, NATALIA.. 2016. An industry experiment on the effects of test-driven development on external quality and productivity. Empirical Software Engineering (2016).