A técnica de Mutação de Teste combinado com o Desenvolvimento Guiado a Testes melhora a precisão dos casos de teste em capturar defeitos?

Arthur H. S. Braga[1]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais - Unid. Praça da Liberdade (PUC Minas)
Av. Brasil, 2023 – 30.140-002 – Belo Horizonte – MG – Brazil

arthur.braga.1098873@sga.pucminas.br

Abstract. *The growth of the software quality field in recent years has provoked an increasing demand in relation to code coverage and the quality of the tests developed. This study aims to explore the combination of two significant practices, Test-Driven Development (TDD) and Mutation Testing, in order to verify if the combination of the two practices would result in lower defect and failure rates, as well as if the union of both, would result in benefits for software development. Also, the intention is to metrify the software tests' general quality and capacity to find defects and flaws in the software development projects in test environments such as pre-production and development environments. For this, teams of students from the Modular Programming Laboratory and Software Testing of the Software Engineering course at the Pontifical Catholic University of Minas Gerais (PUC Minas) were used as a basis.*

Resumo. *O crescimento da área de qualidade de software nos últimos anos tem gerado demandas cada vez mais exigentes em relação a cobertura de código e qualidade dos testes desenvolvidos. Este estudo visa explorar a união de duas práticas, Test-Driven Development (TDD) ou em português Desenvolvimento Guiado a Testes e Mutação de Teste, para verificar se a combinação delas resultaria em menores taxas de defeitos e falhas, bem como a união de ambas resultaria em benefícios para o desenvolvimento de software. Também a intenção é de metrificar a qualidade geral dos testes e a capacidade dos mesmos em encontrar defeitos e falhas nos projetos de desenvolvimento software em ambientes de testes como pre-produção e ambiente de desenvolvimento. Para isso, foi utilizado como base, equipes de alunos da disciplina de Laboratório de Programação Modular e Teste de Software do curso de Engenharia de Software da Pontifícia Universidade Católica de Minas Gerais (PUC Minas).*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientadora de conteúdo (TCC I): Simone de Assis - simone@pucminas.br
Orientador acadêmico (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientador do TCC II: (A ser definido no próximo semestre)

Belo Horizonte, 13 de Novembro de 2022.

1. Introdução

O desenvolvimento orientado à testes, mais conhecido em seu nome em inglês *Test-Driven Development*, ou apenas TDD, é uma prática ágil comum e disseminada na comunidade de Engenharia de *Software*. Essa prática foi introduzida por Beck (2002) para desenvolvimento de *software* com o objetivo de melhorar o entendimento das regras de negócio antes mesmo de iniciar a codificação, auxiliar na criação de confiança dos desenvolvedores em relação ao código desenvolvido, aumentar o número de testes de unidade e cobertura de código. De acordo com uma pesquisa da *State of Agile Report* (2021) , estima-se que 33% das equipes de desenvolvimento de *software* adotam essa técnica em seu trabalho diário. Por outro lado, existe o teste de mutação, que é uma técnica onde são configurados "mutantes" com a intenção de alterar a aplicação a ser testada para provocar falhas nos testes e garantir que eles realmente estão sendo efetivos. Inclusive, é conhecido que esta é uma técnica entre as técnicas de teste mais eficazes [Petrović et al. 2021], que pode ser medida através da capacidade de detectar falhas e defeitos no código.

Na literatura, o tema de TDD e mutação de teste é bastante pesquisado, como a origem e evolução da mutação de teste [Woodward 1993] até a capacidade de reduzir custos [Wong and Mathur 1995] e a efetividade do TDD [Turhan et al. 2010]. Porém, foram encontrados poucos estudos que tem o objetivo de unir a mutação de teste e a prática do TDD, sendo um deles [Roman and Mnich 2021, Derezińska and Trzpil 2015], que encontraram resultados em equipes que usam TDD mais mutação de teste escrevem melhores códigos. A ideia principal deste trabalho é descobrir se a combinação dessas duas práticas resultariam em uma melhor capacidade dos casos de testes em capturar de defeitos e falhas, já que o TDD e a mutação sozinhos não influenciam de forma direta neste resultado. **Visto isso, neste artigo, é explorado algumas questões de pesquisa como:** RQ1) Equipes que utilizam de TDD e mutação de teste têm melhor precisão em capturar falhas e defeitos do que equipes que utilizam o TDD isoladamente? RQ2) Equipes que utilizam TDD e mutação de teste criam casos de testes mais eficientes em capturar *bugs*? RQ3) Equipes que utilizam TDD e mutação de teste encontram menos *bugs* na fase de produção? RQ4) Equipes que utilizam TDD e mutação de teste escrevem códigos mais resilientes a falhas?

Muitos estudos realizados levaram em consideração os benefícios e os bons impactos da prática do TDD em contextos controlados e de mundo real [Yenduri and Perkins 2006]. Algumas dessas vantagens o código limpo e melhores escritos, modularidade e flexibilidade na estrutura do *software*, aumento no número de testes de unidade e o aumento na cobertura de teste. Já na mutação de teste foram identificados alguns benefícios [Ahmed et al. 2016], como a identificação de fragmentos de código fracamente testados e apontamento de testes fracos. A comunidade de Engenharia de *Software* já entende que a prática de ambas abordagens são de grande importância para o sucesso de um projeto. Entretanto, poucos estudos foram realizados com a intenção de combinar essas duas técnicas, que isoladamente, já trazem benefícios à Qualidade de *Software*, mas que combinadas, potencialmente melhorariam a qualidade do projeto de forma geral.

O objetivo geral desta pesquisa é **verificar se a combinação das duas práticas, TDD e Mutação de Teste, resultaria em menores taxas de defeitos e falhas, bem como a união de ambas resultaria em melhores benefícios para o desenvolvimento**

de *software*. Já os objetivos específicos são: (I) metrificar a qualidade geral dos testes, (II) a capacidade dos mesmos em encontrar defeitos e falhas nos projetos de desenvolvimento *software* em ambientes de testes como pre-produção e ambiente de desenvolvimento e (III) medir a resiliência dos casos de teste em relação a mudanças no código original.

Os resultados esperados para esta pesquisa e o as hipóteses levantadas são que a combinação de TDD e mutação de teste criam testes mais resilientes a mudanças no código. Projetos que unem ambas as técnicas são melhores sucedidos em encontrar defeitos e falhas em estágios mais primários de desenvolvimento de *software*. Indicando um outro caminho do que acontece na utilização dessas abordagens isoladamente de forma benéfica porém menor.

Este trabalho está dividido e organizado em quatro seções. Na Seção dois, apresenta-se a fundamentação teórica. A Seção três contempla os trabalhos relacionados. Já a Seção 4 descreve a metologia proposta para a realização do trabalho.

2. Fundamentação Teórica

Para melhor compreensão dos conceitos que compõe a solução abordada por este estudo, esta seção apresenta uma fundamentação teórica sobre: Teste de *Software*, Qualidade de *Software*, *Test-driven development* e Mutação de teste.

2.1. Teste de *Software*

O Teste de *Software* é dos pilares da Engenharia de *Software* e tem como sua responsabilidade investigar e fornecer informações sobre a Qualidade de *Software* [Graham et al. 2021]. O Teste de *Software* é executado pelo Testador, também conhecido como *Quality Assurance*. A sua função é investigar se o *Software* em questão apresenta algum defeito. O teste de *Software* tem como objetivo minimizar os defeitos mirando em diminuir os riscos o máximo possível. Mas, mesmo assim, o processo de Teste não consegue garantir que todo *software* funcione corretamente, sem a presença de erros.

2.2. Test-driven Development

Test-driven Development (TDD), ou em português Desenvolvimento guiado por testes, é uma técnica de desenvolvimento de *software* que tem como objetivo principal criar casos de teste antes da implementação do *software* propriamente [Lodi 2021, Beck 2002]. O processo passa por 6 etapas básicas. 1) Adicione um teste, 2) Execute o teste e veja-o falhar, 3) Escreva o código de implementação do sistema que faça o teste passar, 4) Execute os testes novamente, 5) Refatore os testes. Esse ciclo é conhecido em inglês como *Red, Green, Refactor* como pode ser visto na Figura 1. A Figura 1 apresenta três círculos com as cores Vermelho, Verde e Azul, respectivamente representando o ciclo do TDD. Vermelho (*Red*) como a criação e falha inicial dos testes, Verde (*Green*) como o teste passando pela primeira vez após a primeira implementação do código, e por final, Azul (*Refactor*) como a refatoração do código inicial.

2.3. Mutação de Testes

Mutação de teste também é uma técnica e prática da área de Teste de *software* que tem como seu objetivo testar a qualidade dos testes [Petrović et al. 2021, Woodward 1993]. Enquanto os casos de testes do sistema estão testando o sistema em si, a mutação de teste

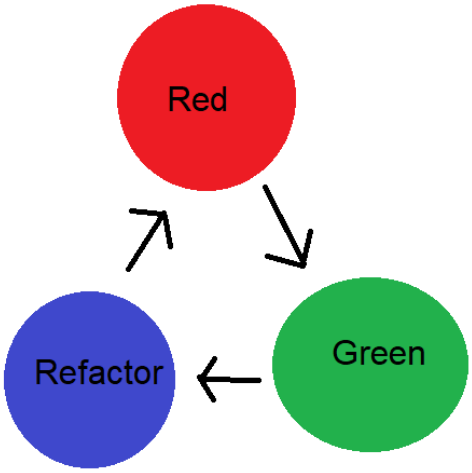

Figura 1. Processo do TDD: Red, Green, Refactor

cria “mutantes” ou “mutações” no sistema original para que um falso erro seja introduzido. Assim que o mutante é introduzido, a suíte de testes original é executada, e se os testes falharem, significa que a mutação foi efetiva e que os testes estão com potencial de capturar erros reais.

3. Trabalhos Relacionados

Os trabalhos relacionados discutidos nesta seção envolvem a análise sobre a aplicação do Test-driven Development (TDD) e Mutação de teste, no contexto da Engenharia de *Software*. Desse modo, os trabalhos estão ordenados de acordo com o seu nível de aproximação com o que se propõe realizar neste trabalho.

Roman e Mnich et al (2021) realizam uma pesquisa que consiste em um estudo de caso para avaliar um cenário do mundo real com equipes que fazem o uso de TDD e mutação de teste com equipes que somente utilizam o TDD de forma isolada. Para isto eles separam duas equipes, uma que aplica o TDD de forma isoladamente e outra que aplica o TDD com a mutação de teste. O estudo avalia algumas perguntas como “Os testes escritos com a abordagem TDD+Mutação oferecem melhor cobertura de código do que os escritos em uma abordagem TDD pura sem processo de mutação envolvido?” e chegam ao resultado de que os testes TDD+Mutação alcançaram melhor cobertura no código de grupos TDD do que os testes TDD em seu próprio código (53,3% vs. 33,5% de cobertura de declaração e 64,9% vs. 37,5% de cobertura de mutação). Este trabalho tem uma grande relação com o estudo proposto neste artigo, a combinação do TDD com mutação de teste para analise de possíveis benefícios para um projeto de *Software*.

Derezínska e Trzpil (2015), realizaram um estudo para avaliar um caso do mundo real com equipes que fazem o uso de TDD e mutação de teste com equipes que somente utilizam o TDD de forma isolada, porém em um ambiente de .NET. A intenção desse estudo é avaliar grupos diferentes de desenvolvedores que utilizam TDD e mutação de teste e grupos que utilizam TDD de forma isolada. O ambiente é fixado em .NET e um contexto específico de equipes da mesma organização. Os pesquisadores notaram que times que utilizam TDD mais mutação de teste possuem melhores resultados e escrevem

melhores casos de testes, mais capazes de capturar erros e defeitos. Esse trabalho tem relação com o estudo proposto neste artigo, que é a combinação do TDD com mutação de teste, porém, em um contexto específico da linguagem .NET, com a intenção de analisar de possíveis benefícios para um projeto de *Software*.

Veloso e Hora (2022) realizam um estudo empírico que tem como objetivo caracterizar testes de alta qualidade através de técnicas de mutação. Uma métrica válida para mensurar a qualidade de testes é a cobertura de código, que consiste em calcular a quantidade de caminhos lógicos e encontrar o percentual de cobertura desses caminhos via linhas de código de teste. O problema com essa prática é que ela é limitada, pois não apresenta com precisão a qualidade dos testes, e nesse momento, o estudo utiliza a mutação de teste. É mencionado também que os testes são uma importante parte da Engenharia de *Software* e eles servem de suporte a qualidade. Para ter um *software* de boa qualidade, é necessário ter testes automatizados de alto desempenho com altas taxas de cobertura e efetividade. Através dessa ótica, é possível notar que é preciso realizar testes nos próprios testes, com o intuito de garantir que os mesmos desempenham o papel que é proposto. Para isto, é utilizado a mutação. A pesquisa indica a utilização de testes de mutação no nível de métodos que se provaram útil para redução de suites e avaliar a qualidade dos mesmos. O estudo avalia as técnicas e comparam testes de alto e baixa desempenho para descobrir a efetividade de cada um deles. A pesquisa mostra por evidências empíricas que existe uma diferença entre os testes, porém, pequena, identificada nos parâmetros de tamanho, número de *assertions* e modificações. Este estudo tem objetivo de caracterizar testes de alta performance e a ideia proposta neste estudo, também avalia a qualidade de testes.

Petrović et al. (2021) tem como objetivo responder uma pergunta: "Testes de mutação melhoram as práticas de testes". Dois dos quatro pesquisadores trabalham na multinacional Google, os mesmos tentam responder a pergunta através de suas experiências profissionais na empresa. É sabido que teste é uma parte essencial no desenvolvimento de *software* e que desenvolvedores de *software* necessitam de uma guia para saber o quanto devem testar e aonde adicionar novos testes. A pesquisa tenta responder a pergunta procurando entender quais são as principais estratégias e técnicas de testes praticados dentro da organização e quais são os problemas relacionados a eles. Os resultados mostram que desenvolvedores que trabalham em projetos com testes de mutação escrevem mais testes em média por períodos mais longos de tempo, em comparação com projetos que consideram apenas a cobertura de código. Testes mutantes são resultados de teste eficazes e desenvolvedores expostos a testes mutantes ajudam a escrever mais testes eficazes. O que também é avaliado neste estudo.

Lasynskyi e Sosnowski (2021) estudaram a utilização de algumas práticas de teste, entre elas TDD e mutação de teste, para avaliar sistemas que precisão de um alto monitoramento, geralmente apoiados pelo *site reabillity*. Foi analisado um amplo escopo de dados coletados durante o teste de um projeto industrial complexo. Esse estudo provou a utilidade dos repositórios e técnicas de *software* na avaliação melhoria processos de teste de *software* para suportar esses sistemas. A conclusão deste estudo mostrou a utilidade dos repositórios de *software* na avaliação e melhoria processos de teste de *software*. A interpretação do relatório de teste gerada é aprimorada, correlacionando entradas registradas com *logs* de aplicativos, adaptação necessária da análise de *log* para sua espe-

cificidade, incluindo características estruturais e semânticas de palavras e frases usadas. Ele também fornece uma visão sobre a consistência do conjunto de testes com a aplicação e aponta os ajustes necessários durante o ciclo de vida do sistema. Esse estudo também se utiliza de TDD para obter melhores benefícios para o *software*.

4. Materiais e métodos

A partir da relação desta pesquisa com os trabalhos relacionados, os materiais e os métodos utilizados por este estudo se baseiam fortemente nas suas técnicas e práticas empregadas, como descritas na seção anterior. Desse modo, alguns métodos são reutilizados por sua eficiência e aprimorados com o intuito de obter resultados mais enriquecedores, ou substituídos por outros métodos mais eficientes para o contexto desta pesquisa.

4.1. Pesquisa e contextos

Esta pesquisa conduz de um estudo controlado, baseada em casos de estudos participativos com objetivo de **coletar dados quantitativos** acerca da utilização das duas práticas combinadas de TDD e mutação de teste e TDD isoladamente. A justificativa da utilização desse método advém dos resultados encontrados nos trabalhos relacionados de [Roman and Mnich 2021, Derezińska and Trzpil 2015], porém, adequado para o contexto deste trabalho.

Desse modo, a partir dos resultados encontrados pela pesquisa de [Roman and Mnich 2021, Derezińska and Trzpil 2015] este trabalho é aplicado em 8 equipes de 3 até 5 alunos da disciplina de Laboratório de Programação Modular do curso de Engenharia de *Software* da Pontifícia Universidade Católica de Minas Gerais (PUC Minas), nas turmas noturnas de 2º período. Selecionados dois grupos onde o primeiro grupo usa o TDD de forma isoladamente e o outro usa o TDD com a prática de mutação de teste. Assim proporcionando uma diversidade de experiência e perfil de alunos, visto que é uma disciplina que possui um enfoque em TDD.

Porém, dado o fomento da disciplina na auto organização e liberdade na escolha dos integrantes e da proposta de trabalho, cada equipe possui um número diferente de participantes e nível de complexidade da solução. Desse modo, este estudo utilizou como critério de seleção das equipes que se voluntariaram a escolher TDD ou TDD com mutação de teste: 1) baixa proximidade social do pesquisador com os integrantes até o 1° contato; 2) proposta de trabalho que possui dúvidas ou desafios com relação o escopo da solução, de modo não ser um tema trivial com amplo conhecimento sobre o assunto; e 3) disponibilidade de agenda dos integrantes.

4.2. Ferramentas

Durante a execução da pesquisa, é utilizado um *software* como base que os alunos desenvolvem 8 métodos com os mesmos requisitos para os dois grupos, com a criação de testes unitários. O grupo 1 utiliza TDD e o grupo 2 utiliza TDD com mutação de teste e os dois utilizam de também métodos criados previamente para integrar e criar novos. Para o projeto, é utilizada a linguagem de programação *Java JDK 18* (*Java devoloper kit*), o *framework* de teste *JUnit* e o *Pitest* para os testes de mutação.

Aos participantes é enviado um link de acesso a um ambiente privado na plataforma *GitHub Classroom*, onde constam um sistema previamente desenvolvido com

alguns métodos que é reaproveitado pelos alunos posteriormente. Os alunos desenvolvem os próprios métodos e utilizando os já existentes. O execução é acompanhada pelos pesquisadores e o professor da disciplina durante o horário de aula. A plataforma escolhida garante melhor controle da pesquisa e gerenciamento dos dados gerados pelos alunos.

4.3. Requisitos do *Software*

O *Software* a ser desenvolvido é um sistema de controle de uma biblioteca para aluguel de livros. É entregue aos alunos uma parte do sistema já desenvolvido, com alguns *bugs* previamente inseridos nos métodos criados, que os alunos não tem conhecimento sobre. Os mesmos continuam o desenvolvimento.

- **Requisito 1:** cadastro , atualização, leitura e deleção de um livro
- **Requisito 2:** cadastro, atualização, leitura e deleção de um usuário
- **Requisito 3:** empréstimo de livro para um usuário
- **Requisito 4:** devolução do livro e data de devolução
- **Requisito 5:** inclusão do usuário a uma fila de espera quando um livro não estiver disponível
- **Requisito 6:** nível de usuário (administrador do sistema e usuário locatário)

4.4. Métricas de avaliação

- **Número de *bugs* encontrados por cada grupo:** Respondendo a RQ 1 e RQ2, como foi mencionado anteriormente, é entregue um *software* previamente construído com alguns requisitos faltantes a serem implementados. Este mesmo *software* também é entregue com alguns *bugs* conhecidos.
- **Capacidade dos casos de teste em capturar *bugs*:** Respondendo as RQ 1 e RQ 2, os casos de teste são avaliados pelos pesquisadores após execução da pesquisa para avaliar quais *bugs* foram encontrados por cada grupo de alunos.
- **Número de *bugs* encontrados anteriormente a entrega final:** Respondendo a RQ 3, é avaliado durante a execução da pesquisa a agilidade em termos de minutos o quão rápido cada grupo encontra os *bugs* introduzidos previamente ou novos durante o desenvolvimento.
- **Capacidade dos casos testes em se manter resilientes a mudanças introduzidas nos requisitos:** Respondendo a RQ 4 Algumas mudanças são introduzidas durante a execução da pesquisa que os alunos devem implementar, provocando assim alteração no código que possa provocar quebra nos testes desenvolvidos dos alunos. Tentando entender assim, se os testes são resilientes a mudanças do código.

4.5. Atividades e cronograma

Para conduzir o estudo, este é o cronograma e atividades a serem realizadas.

1. Desenvolvimento da aplicação para a pesquisa
   (a) Desenvolvimento da criação, consulta, atualização e destruição de dados (CCAD) ou em inglês Create, Read, Update and Delete (CRUD) ou de usuário
   (b) Desenvolvimento do CCAD de livros
   (c) Desenvolvimento do empréstimo de livro

2. Elaboração da dinâmica da pequisa
3. Teste da aplicação da pesquisa
    (a) Coleta de dados do teste piloto
4. Aplicação da pesquisa com os alunos
    (a) Execução
    (b) Analise dos dados

A Tabela 1 mostra o cronograma das atividades de execução durante o semestre.

Tabela 1. Conograma de atividades

Atividades	Mês/Quinzena						
	Fev/1	Fev/2	Mar/1	Mar/2	Abr/1	Abr/2	Mai/1
1(a)	X						
1(b)		X					
1(c)			X				
2				X			
3					X		
4(a)					X		
4(b)					X		
5(a)						X	X
5(b)						X	X

Referências

Ahmed, I., Gopinath, R., Brindescu, C., Groce, A., and Jensen, C. (2016). Can testedness be effectively measured? In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 547–558. Association for Computing Machinery, New York, NY, USA.

Beck, K. (2002). *Test Driven Development. By Example (Addison-Wesley Signature)*. Addison-Wesley Longman, Amsterdam.

Derezińska, A. and Trzpil, P. (2015). Mutation testing process combined with test-driven development in .net environment. In Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., and Kacprzyk, J., editors, *Theory and Engineering of Complex Systems and Dependability*, pages 131–140, Cham. Springer International Publishing.

Graham, D., Black, R., and Van Veenendaal, E. (2021). *Foundations of software testing ISTQB Certification*. Cengage Learning.

Lasynskyi, M. and Sosnowski, J. (2021). Extending the space of software test monitoring: Practical experience. volume 9, pages 166166–166183.

Lodi, G. (2021). Why test-driven development? In *Test-Driven Development in Swift*, pages 1–12. Springer.

Petrović, G., Ivanković, M., Fraser, G., and Just, R. (2021). Does mutation testing improve testing practices? arXiv.

Roman, A. and Mnich, M. (2021). Test-driven development with mutation testing – an experimental study. volume 29, pages 1–38.

Turhan, B., Layman, L., Diep, M., Erdogmus, H., and Shull, F. (2010). How effective is test-driven development. pages 207–217.

Veloso, V. and Hora, A. (2022). Characterizing high-quality test methods: A first empirical study. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 265–269.

Wong, W. and Mathur, A. P. (1995). Reducing the cost of mutation testing: An empirical study. volume 31, pages 185–196.

Woodward, M. (1993). Mutation testing—its origin and evolution. volume 35, pages 163–169.

Yenduri, S. and Perkins, A. L. (2006). Impact of using test-driven development: A case study. volume 1, pages 126–129.