Pontifícia Universidade Católica de Minas Gerais

Curso de Engenharia de Software

Gabriel Henrique Miranda Rodrigues

Proposta de Experimento

Comparação entre Test-Driven Development e Desenvolvimento Tradicional na Qualidade e Produtividade de Software

Belo Horizonte
2025

Sumário

1 Identificação Básica

1.1 Título do Experimento

Comparação entre Test-Driven Development (TDD) e Desenvolvimento Tradicional: Efeitos na Qualidade do Código e Produtividade de Desenvolvedores em Ambiente Corporativo.

1.2 Código de Identificação

EXP-TDD-2025-001

1.3 Versão do Documento

Versão 1.0 — Documento inicial contendo o planejamento completo do experimento.

1.4 Datas

Data de criação: [21/11/2025]
Última atualização: [21/11/2025]

1.5 Autor

Gabriel Henrique Miranda Rodrigues, graduando em Engenharia de Software pela Pontifícia Universidade Católica de Minas Gerais.
Contato: [Inserir e-mail institucional]

1.6 Responsável Principal

Gabriel Henrique Miranda Rodrigues, sob orientação do(a) Prof(a). Danilo de Quadros Maia Filho.

1.7 Projeto Relacionado

Este experimento faz parte do curso de Engenharia de Software da PUC Minas. O trabalho está vinculado à área de qualidade de software e práticas de desenvolvimento, buscando contribuir com evidências sobre a efetividade de técnicas utilizadas no dia a dia de equipes de desenvolvimento.

2 Contexto e Problema

2.1 Descrição do Problema

Um dos desafios mais comuns no desenvolvimento de software é equilibrar produtividade e qualidade. Equipes frequentemente enfrentam situações em que defeitos são descobertos apenas em fases tardias do projeto, gerando retrabalho, atrasos e custos adicionais. Esse cenário levanta uma questão importante: existem práticas de desenvolvimento que ajudam a reduzir defeitos sem comprometer a velocidade de entrega?

O Test-Driven Development, ou TDD, é uma técnica que propõe inverter a ordem tradicional do desenvolvimento. Em vez de escrever o código primeiro e os testes depois, o desenvolvedor começa escrevendo um teste que define o comportamento esperado, implementa o código mínimo para fazer esse teste passar, e então refatora o código para melhorar sua estrutura. A ideia por trás dessa abordagem é que pensar nos testes antes força o desenvolvedor a entender melhor os requisitos e a considerar casos de erro desde o início.

Apesar dos benefícios relatados por alguns praticantes e estudos, existe uma percepção comum de que o TDD torna o desenvolvimento mais lento, já que exige escrever testes antes mesmo de ter código funcionando. Essa percepção gera resistência à adoção da técnica em muitas empresas.

Este experimento busca investigar, de forma controlada, se o TDD realmente produz código com menos defeitos quando comparado ao desenvolvimento tradicional, e qual é o impacto no tempo necessário para completar uma tarefa.

2.2 Contexto Organizacional e Técnico

O experimento será conduzido em uma organização de desenvolvimento de software que utiliza a plataforma .NET Core com a linguagem C#. Os desenvolvedores da empresa trabalham em projetos comerciais e utilizam ferramentas modernas como Visual Studio, Git para controle de versão, e frameworks de teste como xUnit ou NUnit.

O ambiente corporativo foi escolhido porque representa um contexto real de trabalho, diferente de experimentos realizados apenas com estudantes em ambiente acadêmico. Isso aumenta a relevância prática dos resultados, já que as condições do experimento se aproximam do dia a dia de equipes de desenvolvimento.

Os participantes possuem diferentes níveis de experiência, desde desenvolvedores juniores até profissionais mais experientes. Essa diversidade será considerada na análise dos resultados.

2.3 Trabalhos e Evidências Prévias

O TDD tem sido objeto de estudo na área de Engenharia de Software há mais de duas décadas. Diversos pesquisadores investigaram seus efeitos em diferentes contextos, com resultados variados.

Erdogmus, Morisio e Torchiano (2005) conduziram um experimento com estudantes e observaram que participantes usando TDD produziram mais testes e, consequentemente, código que passava em mais casos de teste. Por outro lado, Rafique e Misic (2013), em uma revisão sistemática de literatura, concluíram que as evidências sobre produtividade são inconclusivas, embora haja indicações de melhoria na qualidade externa do código.

Estudos mais recentes, como o de Fucci et al. (2017), questionam se os benefícios observados vêm especificamente do TDD ou simplesmente do ato de escrever testes, independentemente da ordem. Essa discussão reforça a importância de novos estudos que controlem adequadamente as variáveis envolvidas.

A variação nos resultados encontrados na literatura justifica a realização deste experimento em um contexto específico — uma empresa brasileira usando tecnologias .NET — para contribuir com mais evidências sobre o tema.

2.4 Referencial Teórico

O Test-Driven Development foi popularizado por Kent Beck no início dos anos 2000, como parte das práticas do Extreme Programming (XP). A técnica se baseia em um ciclo curto e iterativo, frequentemente chamado de "Red-Green-Refactor".

Na primeira etapa (Red), o desenvolvedor escreve um teste automatizado que descreve um comportamento desejado do sistema. Como o código ainda não existe, esse teste necessariamente falha. Na segunda etapa (Green), o desenvolvedor escreve o código mais simples possível para fazer o teste passar. Não há preocupação com elegância ou otimização nesse momento, apenas com a corretude. Na terceira etapa (Refactor), o desenvolvedor melhora a estrutura do código, removendo duplicações e aplicando boas práticas, sempre garantindo que os testes continuem passando.

Esse ciclo se repete para cada pequena funcionalidade, resultando em um desenvolvimento incremental guiado pelos testes. Os defensores do TDD argumentam que essa abordagem leva a código mais modular, mais fácil de manter e com menos defeitos.

O desenvolvimento tradicional, por sua vez, segue um fluxo diferente. O desenvolvedor primeiro implementa a funcionalidade completa (ou uma parte significativa dela) e depois escreve testes para verificar se o código funciona como esperado. Nessa abordagem, os testes servem principalmente para validação posterior, não para guiar o design do código.

Ambas as abordagens têm seus defensores, e a escolha entre elas muitas vezes depende da cultura da equipe, do tipo de projeto e das preferências individuais. Este experimento busca fornecer dados objetivos que possam informar essa decisão.