Pontifícia Universidade Católica de Minas Gerais

Curso de Engenharia de Software

Gabriel Henrique Miranda Rodrigues

Plano de Experimento

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

Belo Horizonte
2025

Sumário

1 Identificação Básica **3**
1.1 Título do Experimento . . . 3
1.2 Código de Identificação . . . 3
1.3 Versão do Documento . . . 3
1.4 Datas . . . 3
1.5 Autor . . . 3
1.6 Responsável Principal . . . 3
1.7 Projeto Relacionado . . . 3

2 Contexto e Problema **4**
2.1 Descrição do Problema . . . 4
2.2 Contexto Organizacional e Técnico . . . 4
2.3 Trabalhos Prévios . . . 4
2.4 Referencial Teórico . . . 4

3 Objetivos e Questões **5**
3.1 Objetivo Geral . . . 5
3.2 Objetivos Específicos . . . 5
3.3 Questões de Pesquisa . . . 5
3.4 Métricas . . . 5

4 Escopo e Contexto do Experimento **5**
4.1 Escopo Funcional . . . 5
4.2 Contexto do Estudo . . . 6
4.3 Premissas . . . 6
4.4 Restrições . . . 6
4.5 Limitações Previstas . . . 6

5 Stakeholders e Impacto Esperado **6**
5.1 Stakeholders Principais . . . 6
5.2 Interesses e Expectativas . . . 6
5.3 Impactos Potenciais . . . 7

6 Riscos e Critérios de Sucesso **7**
6.1 Riscos de Alto Nível . . . 7
6.2 Critérios de Sucesso . . . 7
6.3 Critérios de Parada . . . 7

7 Modelo Conceitual e Hipóteses **7**
7.1 Modelo Conceitual 7
7.2 Hipóteses Formais 8
7.3 Nível de Significância 8

8 Variáveis, Fatores e Tratamentos **8**
8.1 Objetos de Estudo 8
8.2 Participantes 8
8.3 Variável Independente 8
8.4 Tratamentos 9
8.5 Variáveis Dependentes 9
8.6 Variáveis de Controle 9
8.7 Variáveis de Confusão 9

9 Desenho Experimental **9**
9.1 Tipo de Desenho 9
9.2 Randomização 9
9.3 Contrabalanceamento 10
9.4 Grupos e Sessões 10

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	Data	Descrição
1.0	21/11/2025	Versão inicial do plano

Tabela 1: Histórico de versões

1.4 Datas

Data de criação: 21/11/2025
Última atualização: 28/11/2025

1.5 Autor

Gabriel Henrique Miranda Rodrigues
Área: Engenharia de Software
E-mail: gabriel.rodrigues.1447565@sga.pucminas.br

1.6 Responsável Principal

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

1.7 Projeto Relacionado

Este experimento faz parte do Trabalho de Conclusão de Curso (TCC) do curso de Engenharia de Software da PUC Minas, com foco em práticas de desenvolvimento e qualidade de software.

2 Contexto e Problema

2.1 Descrição do Problema

No desenvolvimento de software, equipes frequentemente enfrentam o desafio de equilibrar velocidade e qualidade. Defeitos descobertos tardiamente geram retrabalho e atrasos. O Test-Driven Development (TDD) propõe escrever testes antes do código, prometendo melhorar a qualidade. Porém, existe a percepção de que isso torna o desenvolvimento mais lento.

Este experimento busca comparar TDD e desenvolvimento tradicional em termos de defeitos e tempo de desenvolvimento.

2.2 Contexto Organizacional e Técnico

O experimento será realizado em uma empresa de desenvolvimento que utiliza C# e .NET Core. A empresa possui desenvolvedores de diferentes níveis de experiência. O ambiente inclui Visual Studio, Git e frameworks de teste como xUnit ou NUnit.

2.3 Trabalhos Prévios

Erdogmus et al. (2005) observaram que TDD produziu código com menos defeitos em experimento com estudantes. Rafique e Misic (2013) fizeram revisão sistemática indicando melhoria na qualidade, mas resultados inconclusivos sobre produtividade. Fucci et al. (2017) questionaram se os benefícios vêm do TDD ou simplesmente de escrever testes.

Esses resultados variados justificam novos estudos em contextos específicos.

2.4 Referencial Teórico

O TDD foi popularizado por Kent Beck como parte do Extreme Programming. Baseia-se no ciclo Red-Green-Refactor:

Red: Escrever um teste que falha.

Green: Escrever código mínimo para o teste passar.

Refactor: Melhorar o código mantendo os testes passando.

No desenvolvimento tradicional, o código é escrito primeiro e os testes depois, servindo principalmente como verificação.

3 Objetivos e Questões

3.1 Objetivo Geral

Analisar o Test-Driven Development com o propósito de comparar seus efeitos em relação ao desenvolvimento tradicional, com respeito à qualidade do código e produtividade, do ponto de vista de desenvolvedores de software, no contexto de implementação de funcionalidades em C# .NET Core em ambiente corporativo.

3.2 Objetivos Específicos

O1: Comparar a quantidade de defeitos no código produzido com TDD versus desenvolvimento tradicional.

O2: Comparar o tempo necessário para completar tarefas usando cada abordagem.

O3: Avaliar a cobertura de testes alcançada em cada abordagem.

O4: Identificar a percepção dos desenvolvedores sobre cada técnica.

3.3 Questões de Pesquisa

Q1: O código desenvolvido com TDD apresenta menos defeitos?

Q2: Existe diferença no tempo de desenvolvimento entre as abordagens?

Q3: A cobertura de testes é maior com TDD?

Q4: Qual a percepção dos desenvolvedores sobre cada técnica?

3.4 Métricas

Questão	**Métrica**	**Unidade**	**Fonte**
Q1	Defeitos detectados	Quantidade	Testes de aceitação
Q2	Tempo de conclusão	Minutos	Registro manual
Q3	Cobertura de código	Percentual	Ferramenta de cobertura
Q4	Percepção dos desenvolvedores	Escala Likert	Questionário

Tabela 2: Métricas associadas às questões

4 Escopo e Contexto do Experimento

4.1 Escopo Funcional

Incluído: Implementação de funções de validação e manipulação de dados em C# (validação de CPF, e-mail, cálculos), criação de testes unitários, refatoração básica.

Excluído: Integração com banco de dados, interfaces gráficas, APIs externas, deploy.

4.2 Contexto do Estudo

Empresa de pequeno a médio porte com desenvolvedores profissionais de nível júnior a sênior. As tarefas serão exercícios isolados sem impacto em projetos reais.

4.3 Premissas

Participantes possuem conhecimento básico de C# e .NET Core.
Ambiente de desenvolvimento estará configurado e funcional.
Participantes terão disponibilidade sem interrupções.
Todos receberão treinamento sobre TDD antes do experimento.

4.4 Restrições

Número de participantes limitado aos disponíveis na empresa.
Tempo total não pode exceder um turno de trabalho (4 horas).
Tarefas devem ser simples para conclusão no tempo disponível.
Uso apenas de ferramentas já disponíveis.

4.5 Limitações Previstas

Experimento em uma única empresa limita generalização dos resultados.
Amostra de conveniência pode não representar todos os desenvolvedores.
Tarefas simplificadas não capturam toda complexidade de projetos reais.

5 Stakeholders e Impacto Esperado

5.1 Stakeholders Principais

Desenvolvedores participantes.
Gestores e líderes técnicos da empresa.
Orientador acadêmico.
Comunidade acadêmica.
Banca do TCC.

5.2 Interesses e Expectativas

Desenvolvedores: aprender sobre TDD e entender seus benefícios.
Gestores: evidências para decisão sobre adoção do TDD.

Orientador: experimento com rigor metodológico.

Comunidade: contribuições sobre práticas de desenvolvimento.

5.3 Impactos Potenciais

Participantes dedicarão 3 a 4 horas, temporariamente reduzindo disponibilidade para outras atividades. Como benefício, receberão treinamento em TDD. Resultados poderão influenciar práticas de desenvolvimento na empresa.

6 Riscos e Critérios de Sucesso

6.1 Riscos de Alto Nível

Número insuficiente de participantes.

Desistências durante o experimento.

Problemas técnicos no ambiente.

Participantes não seguirem o protocolo corretamente.

Mudança de prioridades na empresa.

6.2 Critérios de Sucesso

Pelo menos 80% dos participantes completarem todas as tarefas.

Dados suficientes para responder às questões de pesquisa.

Protocolo seguido sem desvios significativos.

6.3 Critérios de Parada

Menos de 10 participantes confirmados.

Problemas técnicos críticos não resolvidos.

Empresa retirar apoio ao estudo.

Piloto revelar problemas graves no protocolo.

7 Modelo Conceitual e Hipóteses

7.1 Modelo Conceitual

O modelo baseia-se na ideia de que a técnica de desenvolvimento influencia qualidade e tempo. O TDD teoricamente melhora a qualidade ao forçar o desenvolvedor a pensar nos requisitos antes de codificar, permitindo feedback rápido. Porém, pode aumentar o tempo inicial por exigir testes antes do código funcional.

O desenvolvimento tradicional pode ser mais rápido inicialmente, mas potencialmente resulta em mais defeitos porque os testes são escritos depois.

7.2 Hipóteses Formais

Para defeitos (Q1):

$H_{0,1}$: Não existe diferença na quantidade de defeitos entre TDD e tradicional.

$H_{1,1}$: TDD apresenta menos defeitos que tradicional.

Para tempo (Q2):

$H_{0,2}$: Não existe diferença no tempo de desenvolvimento.

$H_{1,2}$: Existe diferença no tempo de desenvolvimento.

Para cobertura (Q3):

$H_{0,3}$: Não existe diferença na cobertura de testes.

$H_{1,3}$: A cobertura é maior com TDD.

7.3 Nível de Significância

Será adotado $\alpha = 0,05$ para todos os testes. O poder estatístico desejado é 0,80. Com base em estudos anteriores, estima-se que 12 a 20 participantes sejam adequados para um desenho pareado.

8 Variáveis, Fatores e Tratamentos

8.1 Objetos de Estudo

Implementações de funcionalidades em C# .NET Core produzidas pelos participantes, como validadores de dados e funções de cálculo.

8.2 Participantes

Desenvolvedores da empresa, entre 12 e 20 profissionais, com diferentes níveis de experiência.

8.3 Variável Independente

Técnica de desenvolvimento com dois níveis:

Nível 1: Test-Driven Development (TDD)

Nível 2: Desenvolvimento Tradicional

8.4 Tratamentos

Tratamento A (TDD): Participante segue o ciclo Red-Green-Refactor. Escreve teste que falha, implementa código para passar, refatora.

Tratamento B (Tradicional): Participante implementa funcionalidade completa primeiro, depois escreve testes unitários.

8.5 Variáveis Dependentes

Número de defeitos: quantidade de falhas detectadas pelos testes de aceitação.

Tempo de conclusão: tempo total em minutos.

Cobertura de testes: percentual de código coberto pelos testes unitários.

8.6 Variáveis de Controle

Complexidade das tarefas (equivalente para todas).

Ambiente de desenvolvimento (mesma configuração).

Tempo disponível (igual para todos).

Especificação das tarefas (mesmas instruções).

8.7 Variáveis de Confusão

Experiência prévia com TDD.

Experiência geral em programação.

Familiaridade com o domínio das tarefas.

Essas serão coletadas via questionário e consideradas na análise.

9 Desenho Experimental

9.1 Tipo de Desenho

Desenho within-subjects (medidas repetidas) com contrabalanceamento. Cada participante executa tarefas em ambas as condições, servindo como seu próprio controle. Isso reduz variabilidade e aumenta poder estatístico.

9.2 Randomização

Participantes serão divididos aleatoriamente em dois grupos:

Grupo 1: Primeira tarefa com TDD, segunda com tradicional.

Grupo 2: Primeira tarefa com tradicional, segunda com TDD.

A randomização será por sorteio computadorizado.

9.3 Contrabalanceamento

O contrabalanceamento evita que efeitos de aprendizagem ou fadiga favoreçam uma técnica. Serão usadas duas tarefas diferentes de complexidade equivalente (Tarefa X e Tarefa Y).

9.4 Grupos e Sessões

Dois grupos de contrabalanceamento. Cada participante realiza duas sessões, uma com cada técnica. Experimento em um único dia com intervalo entre sessões.