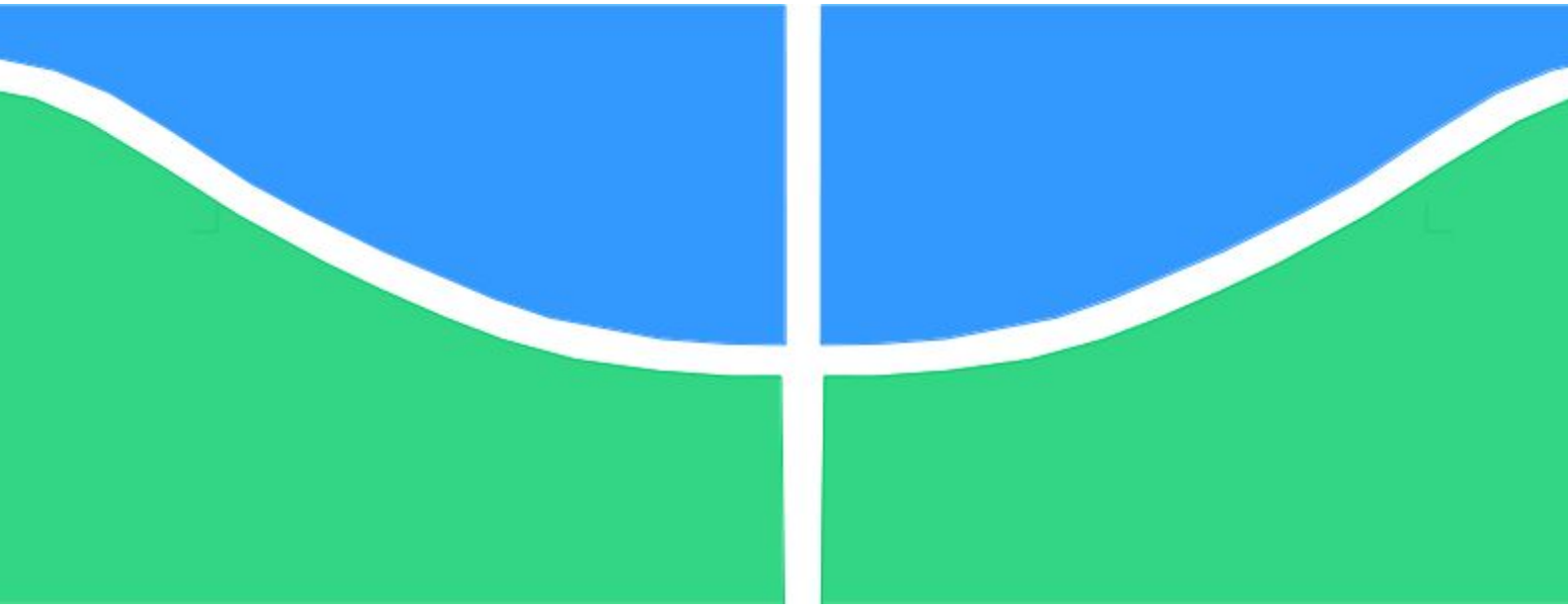

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Testes de Software

TBL 02 — Fase 03 — Test-driven Development

Orientadores:
Professor: Ricardo Ajax
Monitora: Amanda Bezerra

Brasília, DF
16 de novembro de 2019

1. GRUPO 07 - INTEGRANTES E CONTRIBUIÇÕES

Tabela 1 — Integrantes e Contribuições ao TBL 02, Fase 3

Integrante	Matrícula	Contribuição
Amanda Pires	15/0004796	100%
André Pinto	17/0068251	100%
Ivan Dobbin	17/0013278	100%
Leonardo Medeiros	17/0038891	100%
Lieverton Silva	17/0039251	100%
Renan Cristyan	17/0044386	100%
Welison Regis	17/0024121	100%
Wictor Girardi	17/0047326	100%

Fonte: dos autores, 2019.

2. INTRODUÇÃO

Test-Driven Development (TDD) é uma prática de desenvolvimento de software ágil. Esta foi criada com o objetivo de reduzir esforço de desenvolvimento, realizar rápidas iterações e garantir que os requisitos do cliente sejam preservados.

"Test Driven Development é um método extremo de desenvolvimento de programação no qual um sistema de software é desenvolvido em poucas iterações." (TORT; OLIVÉ; SANCHO, 2011, p. 1088, tradução nossa).

Neste esquema é criado antes testes para as funcionalidades do sistema. TDD possui o seguinte ciclo de desenvolvimento:

1. Escreva um Teste que inicialmente não passa (Red)
2. Adicione uma nova funcionalidade do sistema
3. Faça o Teste passar (Green)
4. Refatore o código da nova funcionalidade (Refactoring)
5. Escreva o próximo Teste

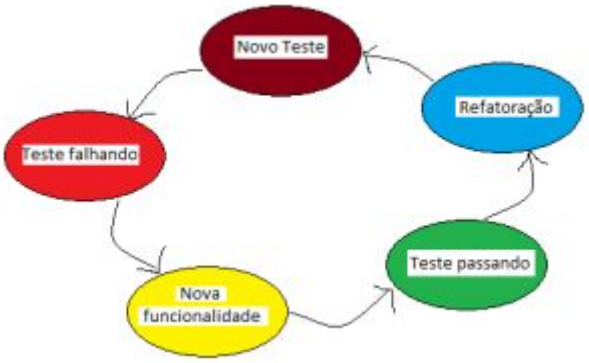

Figura 1: Ciclo de desenvolvimento. Fonte: DEVMEDIA, 2010.

Descrição das Fases:

1. Novo teste: existe uma nova funcionalidade, neste momento realiza-se o contrário do tradicional, assim testa-se e depois codifica-se.
2. Teste falhando: o teste foi escrito, mas não tem implementação, assim o teste consequentemente irá falhar.
3. Nova funcionalidade: nesta etapa se escreve o código de maneira mais simples possível, esquecendo padrões, boas práticas e etc. Assim codifica-se a nova funcionalidade de maneira mais rápida possível para o teste passar. Quando o teste passa, parte-se para a próxima etapa.
4. Refatoração: neste momento analisa-se o código que foi feito de maneira simples e aplica-se às boas práticas, retira-se duplicidade, renomeá-se variáveis, extraí-se métodos, classes, interfaces, utiliza-se algum padrão conhecido e etc, ou seja, o objetivo é deixar o código simples, claro e funcional.

3. FALSIFICAÇÃO

A técnica de falsificação é um dos primeiros passos no desenvolvimento orientado a testes por ser simples e poderosa. Sua implementação é basicamente escrever um teste que falhe e, para fazê-lo passar, utiliza-se constantes, valores previamente conhecidos em relação a uma determinada entrada, e posteriormente, gradualmente, ir substituindo as constantes por variáveis.

A falsificação causa efeitos positivos psicologicamente e também no controle de escopo:

- Psicológico: aumenta a confiança, uma vez que você terá vários testes para provar que a implementação continua funcionando mesmo depois de uma refatoração e, é possível alcançar a barra verde rapidamente.
- Controle de escopo: ajuda a trabalhar com um foco bem definido, logo que o programador pode tratar um problema isolando um pedaço dele e gradativamente adicionando complexidade à solução.

4. TRIANGULAÇÃO

A técnica da triangulação consiste em realizar o mesmo caso de teste com entradas diferentes, de forma que seja possível generalizar os resultados. Por exemplo, ao realizar um teste em uma função que faz algum cálculo, fornecer pelo menos 3 conjuntos de entradas diferentes. Dessa forma, se todos os casos passam, é possível abstrair que a função funciona como o esperado. Fornecer entradas inválidas também ajuda a descobrir problemas na implementação do código.

4.1. TRIANGULAÇÃO COM PARÂMETROS EM FRAMEWORKS DE TESTES UNITÁRIOS

Os testes parametrizados permitem que um desenvolvedor execute o mesmo teste repetidamente usando valores diferentes. Para implementar o teste parametrizado são necessárias cinco etapas.

- Sobrescrever a classe de teste com @RunWith (Parameterized.class);
- Criar um método estático público sobrescrito com @Parameters que retorne uma coleção de objetos como conjunto de dados de teste;
- Criar um construtor público que obtenha o equivalente a uma linha de dados de teste;
- Criar uma variável de instância para cada coluna dos dados de teste;
- Criar os casos de teste usando as variáveis de instância como a fonte dos dados de teste.

O caso de teste será chamado uma vez para cada linha de dados.

5. ANÁLISE DAS TÉCNICAS

5.1. Pontos Positivos

- **Código com mais qualidade:** O próprio programador deve criar casos de testes para suas novas implementações ou modificações no código-fonte.

Quando os casos de teste passarem ele deve refatorar para buscar um código mais claro e eficiente.

- **Aumenta a coragem do programador:** O programador não precisa se preocupar se as mudanças que ele fizer vão quebrar o sistema, pois os casos de testes irão apontar os defeitos. Ou seja, é possível fazer várias modificações e, se os casos não passarem, consertar os problemas para que as alterações sejam incluídas.
- **Refatoração de código:** Uma das bases do TDD é refatorar o código após seus casos de testes passarem. Dessa forma, um código que funciona e faz o que deveria pode ser melhorado para executar mais rápido, consumir menos memória e ser mais legível para facilitar a manutenção, por exemplo.

5.2. Pontos Negativos

- **Manutenção:** a suíte de testes tem que receber manutenção, os testes podem não ser completamente determinísticos (podem depender de fatores externos).
- **Dificuldade de escrita:** os testes podem ser difíceis de escrever, pode passar do level de um teste de unidade.
- **Desenvolvimento mais lento:** pode deixar o desenvolvimento inicialmente mais lento, para ambientes startup de rápida iteração a implementação do código não estará pronta por algum tempo, por causa do tempo gasto para escrever os testes numa fase inicial.
- **Qualidade dos testes pode ser baixa:** escrever testes unitários é uma forma de arte. Acontece muito do foco estar em cobertura de código, porém isso não diz precisamente sobre a qualidade dos teste.

5.3. Aplicabilidade Prática

Desenvolver em unidades menores, através de testes, reduz a complexidade e permite o desenvolvimento de uma aplicação de dentro para fora pois, o desenvolvedor trabalha diretamente no modelo, além de favorecer o baixo acoplamento e a alta coesão dos componentes criados. Porém é necessária a adaptação do desenvolvedor a essa forma de programação. Boa parte dos programadores vêm o desenvolvimento orientado a testes como uma barreira para chegar ao resultado final.

Outra dificuldade da aplicação do TDD se dá quando o desenvolvedor está aprendendo uma nova tecnologia ou um novo paradigma, tornando a aprendizagem trabalhosa de forma que o resultado não vale o esforço.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] BECK, Kent. **Test-Driven Development By Example**. Addison-Wesley Professional, 1ª edição, 2002.

[2] GAMA, Alexandre. **Test Driven Development: TDD simples e prático.** DEVMEDIA, Rio de Janeiro, 2010. Disponível em: https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533 Acesso em: 16/11/19.

[3] GHARAI, Amir. **Pros and cons of Test Driven Development. Testing excellence**, 3 dez. 2019. Disponível em: https://www.testingexcellence.com/pros-cons-test-driven-development/. Acesso em: 16/11/19.

[4] MOREIRA, Wellington. **TDD - Test-Driven Development - Guia Rápido.** Disponível em: http://blog.sciensa.com/tdd-test-driven-development-guia-rapido/. Acesso em: 15/11/19.

[5] TORT, Albert; OLIVÉ, Antoni; SANCHO, Maria-Ribera. **An approach to test-driven development of conceptual schemas.** Data & Knowledge Engineering Amsterdã, n. 70, p. 1088–1111, 2011. Disponível em: https://www.sciencedirect.com/science/article/pii/S0169023X11000978. Acesso em 16/11/19.

[6] NANTHAAMORNPHONG, Aziz; CARVER, Jeffrey. **Test-Driven Development in HPC Science: A Case Study. Computing in Science & Engineering**, v. 20, n. 5, p. 98-113, sep./oct. 2018. Disponível em: https://ieeexplore-ieee-org.ez54.periodicos.capes.gov.br/document/8452053/authors#authors . Acesso em: 16/11/19.