Estratégias de Escrita de Testes Automatizados

Paulo Cheque

12/02/2009 Verão 2009

Sobre a Palestra

- Refatoração
- TAD
- TFD/POUT
- TDD
- BDD
- Padrões e Anti-padrões

(Refatoração)

- Uma modificação **feita em pequenos passos** no sistema que não altera o comportamento funcional, mas melhora sua estrutura interna

(Refatoração - Catálogos)

- Rename
- Extract Method
- Pull Up Method
- Encapsulate Field
- Replace Conditional with Polymorphism
- ...

EXEMPLO

(Refatoração - Motivação)

- Código ilegível
- Replicação de código
- Código extenso/mal organizado
- Muitos parâmetros
- Comentários dispensáveis
- Intimidade inapropriada
- ...

(Refatoração - Consequências)

- Melhora o design depois que o código foi escrito
- Agiliza o desenvolvimento
- Minimiza as chances de introduzir erros
- ...

(Execução dos testes)

- A execução deve ser ágil
- Sempre que uma nova porção de código é criada
- À noite, em uma máquina dedicada EXEMPLO
- Assim que alterações são detectadas no repositório
- É melhor escrever e executar testes incompletos do que não executar testes completos

Test-After Development (TAD)

- Conhecer a arquitetura
- Implementar
- Testar **automaticamente**

- Em comparação a testes manuais:
  - Evita erros de regressão
  - Documentação útil e atualizada do que foi testado

TAD - Desvantagens

- SUT com pouca testabilidade
- Alterar o SUT para testar, faz sentido?
- Difícil de testar
  - Anti-padrões
- Faz sentido? São deixados de lado!

Quando utilizar TAD

- Equipes que estão aprendendo testes automatizados
- Sistemas legados
- Antes de refatorações
- Correção de erros: Simule o erro com um teste

Test-First Development (TFD)

- Plain Old Unit Test (POUT)

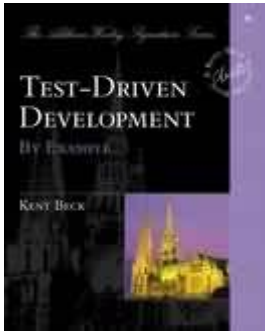
- Conhecer a arquitetura
- Testar
- Implementar

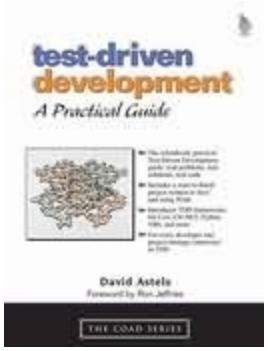
TFD - Características

- Vantagens:
  - Mais Testes
  - Menos Pressão
  - Mais Testabilidade
- Desvantagens:
  - Estudo adicional para design sem colocá-la em prática
  - Geração de código e testes não utilizados

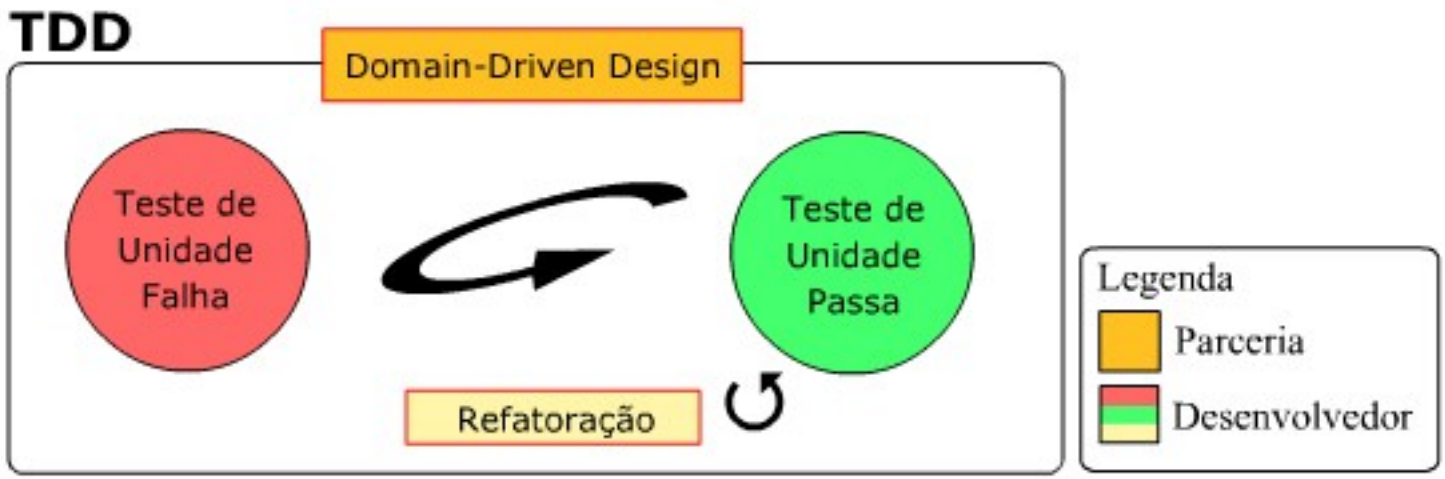
Test-Driven Development (TDD)

- TDD != TFD
- *"Código limpo que funciona"* - Ron Jeffries
- Desenvolvimento em ciclos pequenos:
  - Escreva um teste que falha
  - Implemente o necessário para o teste passar
  - Refatore
- SUT faz o que os testes pedem



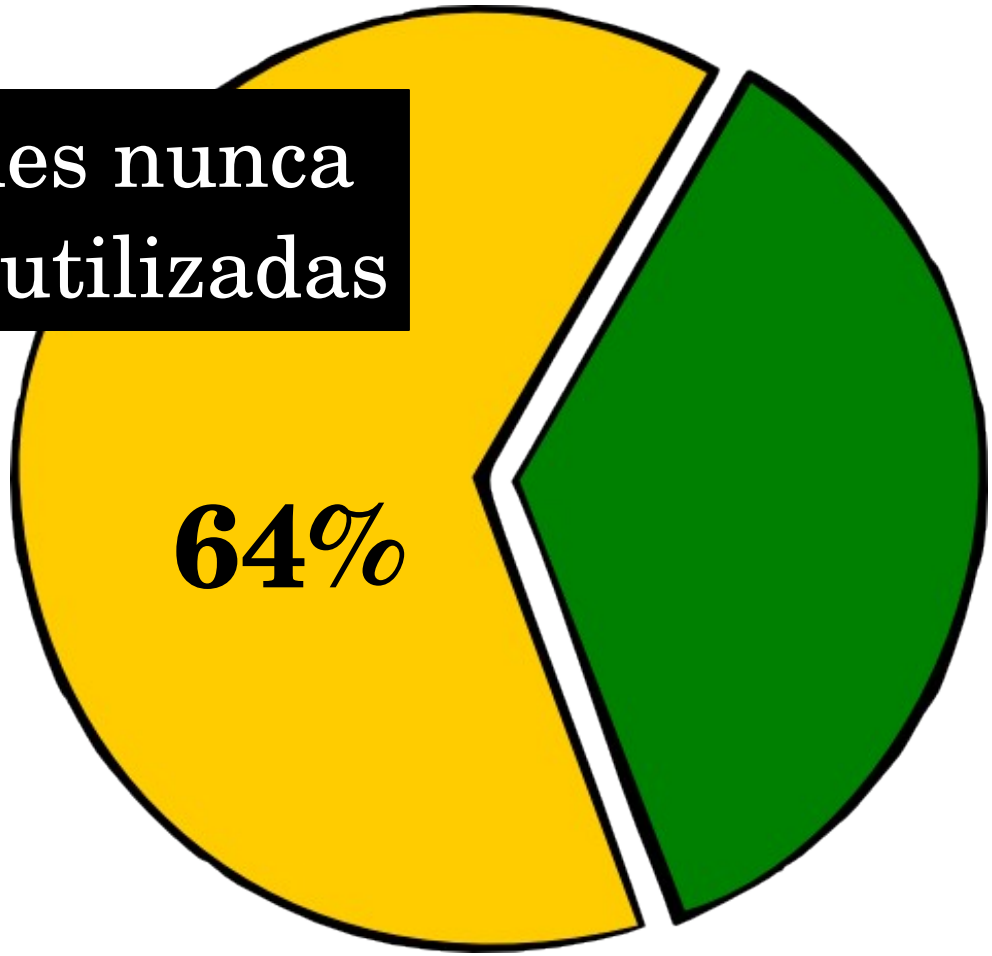


TDD


TDD
Domain-Driven Design
Teste de Unidade Falha
Teste de Unidade Passa
Refatoração
Legenda
Parceria
Desenvolvedor

Desperdício

Funcionalidades nunca ou raramente utilizadas



TDD

- Design evolui junto com o conhecimento
  - *"Code for tomorrow, design for today"* - K. Beck
- Desenvolvimento com passos pequenos
- Expressa a intenção do programador com testes
- Servem como documentação

Design

- Single Responsibility Principle (SRP)
- Open Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Injeção de Dependência

- Evitar comandos 'new' de colaboradores dentro dos objetos
  - Construtor(Colaborador)
  - setColaborador(Colaborador)
  - metodo(Colaborador)
- OO: Não pergunte, diga!

Dependency Lookup

- Permitir que o acoplamento normal entre objetos seja quebrado durante a automação de testes
- Implementação: Procurar dependência através de um serviço de registros de dependências
- Exemplo:
- getCurrentTimeAsHTML()
  - return “” + new TimeProvider().getTime() + “”

Humble Object

- Dependências muito acopladas à frameworks
- Humble Object: Adaptador conectado ao framework que redireciona as chamadas para um componente facilmente testável

Test Hook

- Evitar ao máximo
- Proibido!
- Lógica de testes em produção
- if(TESTING) { faça algo }
- Else { faça o que se deve fazer }

TDD

- O código:
  - Nome dos testes definem o comportamento esperado
  - Fatorados (sem replicação)
  - Alta cobertura
  - Evita código inútil
  - Alta qualidade
    - Refatorações seguras

TDD - Obstáculos

- Difícil costume
  - Muito diferente do modo tradicional de programar
- Difícil aplicar em sistemas complexos
- Difícil aplicar em sistemas legados

Padrões de Barra Vermelha

- Um passo: Comece pelo que você sabe fazer
- Teste Iniciador: Comece testando situações simples
- Teste explicativo: Conversar em termos de testes
- Teste de aprendizado: Criar testes para conhecer aplicações externas
- Teste de regressão: Escreva teste que simule um erro antes da sua correção

Padrões de Barra Verde

- Falsificação: Até implementar de verdade
- Triangularização: Testar dois casos para buscar uma abstração
- Implementação óbvia: Quando possível
- Um para Muitos: Antes de testar coleções, testar um elemento

Anti-Padrões

- Testes focados na implementação e não na funcionalidade
- Testes frágeis
- Falta de testes para casos importantes

Classicist TDD vs Mockist TDD

- TDD Clássico vs TDD Mockado
- http://martinfowler.com/articles/mocksArentStubs.html
- Clássico:
  - Usa objetos reais sempre que o custo for baixo
  - Mini-integration tests
- Mockado:
  - Isola todas as dependências, mesmo as que são facilmente utilizadas

Classicist TDD

- 1. Visão + Controlador Stubs
- 2. Modelo
- ou
- 1. Modelo (DDD)
- 2. Visão e Controlador sem stubs ou dublês

- Bugs: Afetam testes da funcionalidade incorreta e de camadas superiores

Mockist TDD

- Estruturado:

  Visão + resto mockado

  Controlador + resto mockado

  Modelo + resto mockado (DDD)

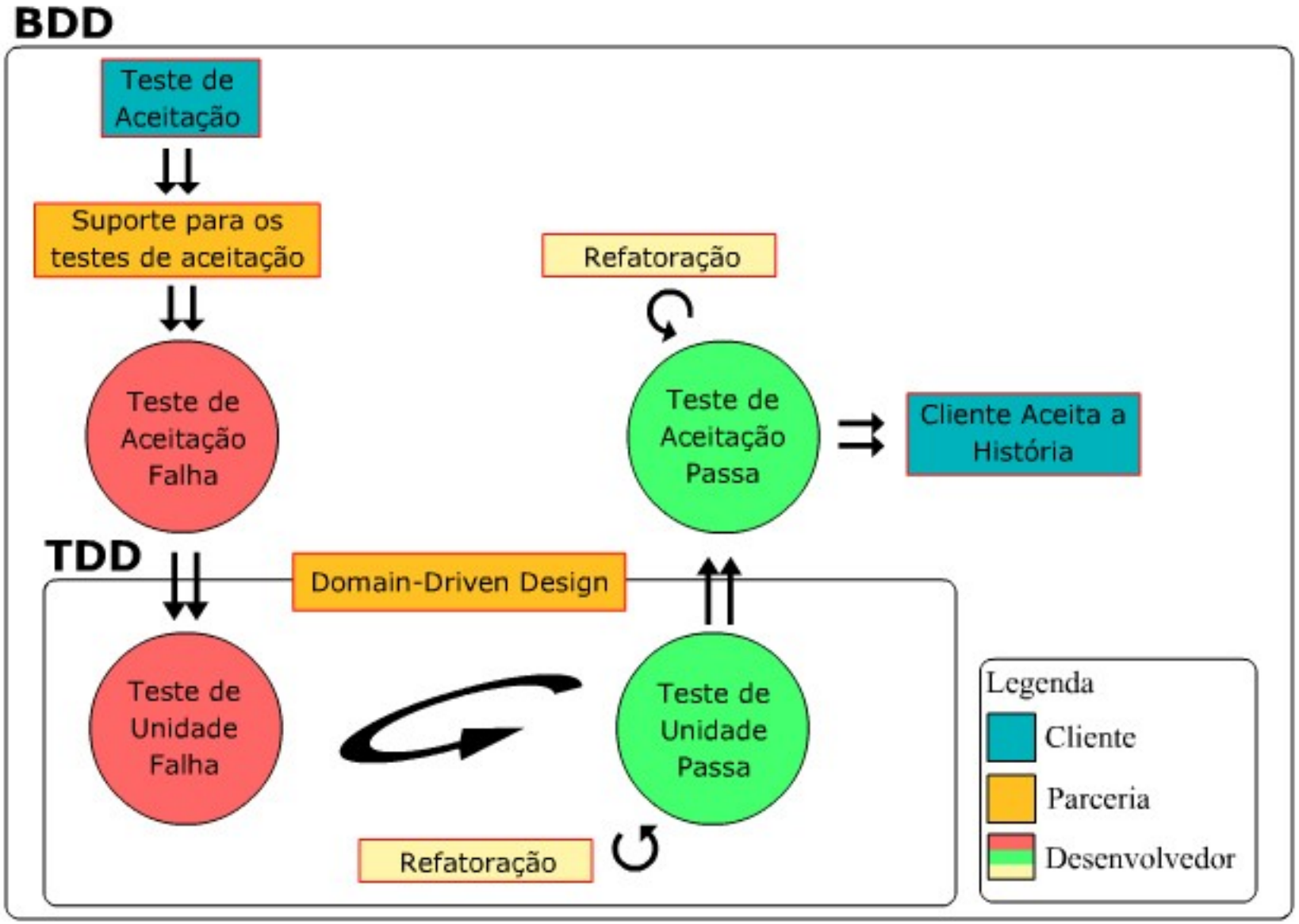
- Bugs: Afetam apenas os casos de testes mais relevantes

- Mais presos à implementação

Behaviour Driven Development (BDD)

- Ciclo adicional para testes de aceitação
- Testes ficam como pendentes
- Linguagem do cliente (comum)
- As a, I want to, So that
- Given, When, Then
- Imagem do ciclo
- Exemplo rSpec
- Documentação executável

BDD


BDD
Teste de Aceitação
Suporte para os testes de aceitação
Teste de Aceitação Falha
Refatoração
Teste de Aceitação Passa
Cliente Aceita a História
TDD
Domain-Driven Design
Teste de Unidade Falha
Teste de Unidade Passa
Refatoração
Legenda
Cliente
Parceria
Desenvolvedor

Program Exploration (Pex)

- Projeto de pesquisa da Microsoft
  http://research.microsoft.com/en-us/projects/Pex

- Gera entrada de dados nos testes através de testes exploratórios

- Tenta entender o código do algoritmo para gerar outros casos de testes pertinentes

Contato

http://www.agilcoop.org.br

agilcoop@agilcoop.org.br

paulocheque@agilcoop.org.br