Padrões de Testes Automatizados

Paulo Cheque

10/02/2009 Verão2009

Introdução

- Testes codificados
- Exigem boa programação
- Mesmos problemas de um software
- Devem receber o mesmo tratamento
- Exigem manutenção
- Exigem um design (simples)
- Permite erros

Motivação

- Padrões => Soluções comuns para situações recorrentes
- Facilitar e otimizar a escrita dos testes
- Refatoração de código de teste

Legenda

```
// Comentário
@Test
public void metodo() {
    String a = “some string”;
}
```

Definições

- Padrões:
  - Descrição de uma solução para problemas recorrentes no desenvolvimento de software
- Anti-Padrões:
  - Um padrão não recomendado, que traz problemas para o desenvolvimento
- Cheiros:
  - Sintomas do código fonte que indicam que existe alguma coisa errada

Cheiros

- Código
- Comportamento
- Projeto

Cheiros do Código

- Obscure Test
- Conditional Test Logic
- Hard-to-Test Code
- Test Code Duplication
- Test Logic Production

Cheiros de Comportamento

- Assertion Roulette
- Erratic Test
- Fragile Test
- Frequent Debugging
- Manual Intervation
- Slow Tests

Cheiros do Projeto

- Buggy Tests
- Developert not Writing Tests
- High Test Maintenance Cost
- Production Bugs

Padrões de

- Organização
- Verificação de resultados
- Valor
- Suporte para SetUp
- Suporte para TearDown

- Estratégias
- Arquitetura testável

Padrões de Organização

- Métodos:
  - Test Utility Method
  - Test Helper
  - Parametrized Test
  - Testcase Superclass
- Classes:
  - Testcase Class per Class/Feature/Fixture
- Baterias:
  - Named Test Suite

Premissas

- Nunca adicionar código de testes ao SUT
  - Não utilizar lógica dos testes no sistema
- Não incluir código de testes em bibliotecas
  - Criar uma biblioteca separada
- Padronizar nomes de classes para facilitar a identificação e utilização de *scripts*
  - AlgumaClasse**Test**

Métodos

- Test Utility Method / Parametrized Test
  - Refatoração: Extract Method em uma classe de teste
- Test Helper
  - Extract Method + Extract Class
  - Métodos úteis para mais de uma classe de teste
  - AlgumaNome**TestHelper**
- Testcase Superclass
  - Extract Method + Extract Superclass

Test Utility Method / Parametrized Test

```
64  @Test
65  def cartasInvalidasLancamExcecao() {
66    testaCartaInvalida(1, PAUS)
67    testaCartaInvalida(AS+1, PAUS)
68    testaCartaInvalida(AS, PAUS-1)
69    testaCartaInvalida(AS, OUROS+1)
70  }
71
72  private def testaCartaInvalida(valor:Int, naipe:Int) {
73    try {
74      new Carta(valor, naipe)
75      fail("ops")
76    } catch {
77      case e:RuntimeException => assertTrue(true)
78      case _ => fail("ops")
79    }
80  }
```

Test Helper

```
 7 object PokerHelperTest {
 8
 9   def c(tuplas:Tuple2[Int, Int]*): Combinacao = {
10     var list = List[Carta]()
11     tuplas.foreach { tupla:Tuple2[Int, Int] =>
12       list = list + new Carta(tupla._1, tupla._2)
13     }
14     var comb = CombinacaoFactory.get(list)
15 //     var comb = new Combinacao(list)
16     comb
17   }
18
19   def menorCartaMaisAlta(): Combinacao = {
20     c((2, PAUS), (3, PAUS), (4, PAUS), (5, PAUS), (7, COPAS))
21   }
22
23   def maiorCartaMaisAlta(): Combinacao = {
24     c((9, PAUS), (J, PAUS), (Q, PAUS), (K, PAUS), (AS, COPAS))
25   }
26
27   def menorPar(): Combinacao = {
28     c((2, PAUS), (2, COPAS), (3, PAUS), (4, PAUS), (5, PAUS))
29   }
30
31   def maiorPar(): Combinacao = {
32     c((AS, PAUS), (AS, COPAS), (K, PAUS), (Q, PAUS), (J, PAUS))
```

Testcase Class per ...

- Class (Primeira opção):
  - AlgumaClasse => AlgumaClasseTest
- Feature (pode ser um Método):
  - Organizando classes de teste com muitos testes
  - Número de métodos de uma classe de teste cresce mais rapidamente que de uma classe do SUT
- Fixture:
  - Organizando de acordo com o Set Up

Named Test Suite

- Grupos de testes
- Exemplos:
  - Suite de testes fumaça
  - Suite de testes de procedures
- TestNG (Java) tem um bom suporte:

```
@Test(groups = {“SmokeTests”, “Interface”})
public void verificaLinksQuebrados() { ... }
```

Verificações de Resultados

- State Verification
- Behavior Verification
- Custom Assertion
- Delta Assertion
- Guard Assertion
- Unfinished Test Assertion

Verificação

- State Verification
  - assertEquals(estadoEsperado, sut.getEstado())
- Behavior Verification
  - Sem estado (logs), usar Spy ou Mock

- Cuidado com Anti-Padrão: Verificar a implementação e não a funcionalidade

```
expect(mock).doIt()).times(53)
```

Custom Assertion

```
public class IsNotANumber extends TypeSafeMatcher<Double> {
  @Override public boolean matchesSafely(Double number) {
    return number.isNaN();
  }
  public void describeTo(Description description) {
    description.appendText("not a number");
  }
  @Factory public static <T> Matcher<Double> notANumber() {
    return new IsNotANumber();
  }
}
assertThat(Math.sqrt(-1), is(notANumber()));
```

Asserção

- Delta Assertion
  - assertEquals(lista.size()+1, lista.size())
- Guard Assertion
  - Verificação por precaução
- Unfinished Test Assertion
  - @Ignore ou skip
  - fail(“Ainda não implementado”)

Padrões de Valor

- Literal Value
  - Informação *hard-coded*: BigDecimal("13.42")
  - Cuidado com testes não repetitíveis
- Derived Value
  - Implementar um algoritmo que gere o valor esperado
  - Cuidado para não reproduzir o algoritmo do SUT nos testes

Padrões de Valor

- Generated Value
  - Gerar valores distintos a cada teste
  - Útil para testes de integração
  - Cuidado com a reprodutibilidade dos testes

```
id = IDGenerator.uniqueID();
```

- Dummy Object
  - Apenas para não atrapalhar os testes

```
expect(obj.toString()).andReturn(anything());
```

Padrões de Suporte de Setup

- Fresh Fixture Setup
- Shared Fixture Construction

Setup: Zerados

- In-line setup
  - Cada teste tem o seu
- Delegated setup
  - SetUp distintos para cada teste
  - Extract Method
- Implicit setup
  - TestCase per Fixture
  - Padrão dos arcabouços

Setup: Compartilhados

- Prebuilt fixture
- Lazy setup
- Suite fixture setup
- Setup decorator
- Chained tests
  - Add => Select => Update => Delete

Padrões de Suporte de Teardown

- Teardown Strategy
  - Garbage-Collected Teardown
  - Automated Teardown
- Code Organization
  - In-line Teardown
  - Implicit Teardown: Padrão

Teardown: Estratégias

- Garbage-Collected Teardown
  - Delete
  - frame.cleanUp()
  - System.gc()
- Automated Teardown
  - Registro do que foi inserido para ser limpado no teardown

Teardown: Organização

- In-line Teardown
  - Cada teste tem o seu
  - Após as verificações
- Implicit Teardown
  - Padrão dos arcabouços

Estratégias

- Gravação
- Scripts
- Testes com Arcabouços (Frameworks)

- Dirigido por dados (Data-Driven)
- Espionagem (Back Door)
- Camadas

Data-Driven Tests

```
22  @DataProvider(name = "situacoesParaDescontos")
23  public Object[][] situacoesParaDescontos() {
24     return new Object[][] {
25           {new Date(), 1000, 100.0f, 10.0f},
26           {new Date(), 2000, 100.0f, 15.0f},
27           {new Date(), 3000, 100.0f, 20.0f},
28           {new Date(), 1000,  50.0f, 5.0f}
29           // ...
30     };
31  }
32
33  @Test(dataProvider = "situacoesParaDescontos")
34  public void testaRegraDeDescontos(
35        Date dataDeCadastro, Integer quantidadeDePontos, Float valorDaCompra, Float valorEsperado) {
36     Desconto d = new Desconto(dataDeCadastro, quantidadeDePontos, valorDaCompra);
37     assertEquals(valorEsperado, d.resultado());
38  }
```

Padrões de Arquitetura Testável

- Injeção de Dependência
- Busca de Dependências (lookup)
- Humble Object
- Test Hook (Anti-Padrão)

Injeção de Dependência

- Implementação
  - Construtor
  - Setter / variáveis públicas
  - Lookup
- Ajuda a orientação a objetos:
  - Não pergunte, diga!

Injeção de Dependência

```
3  public class Compra {
4     private Cliente cliente;
5     private Float valor;
6
7     // Se adicionar uma nova regra de desconto, bá-bau!
8     public float valorDaCompraComDesconto() {
9        Desconto desconto = new Desconto(cliente.getDataDeCadastro(), cliente.getQuantidadeDePontos(), valor);
10       return valor - desconto.resultado();
11    }
12
13    // Injeçao de dependência via argumento
14    public float valorDaCompraComDesconto(Desconto desconto) {
15       return valor - desconto.resultado();
16    }
17 }
```

Humble Object

- Objeto possui regras importantes a serem testadas e tarefas complexas e não testáveis de frameworks, de concorrência, etc

- Refatoração:
  - Quebrar o objeto em 2:
    - Um contendo as regras
    - Outro que será um adaptador (Humble Object) que delegará as regras para o objeto anterior

Test Hook

- Modificar o sistema para funcionar diferentemente na execução do teste

```
if(TESTING) { ... } else { ... }
```

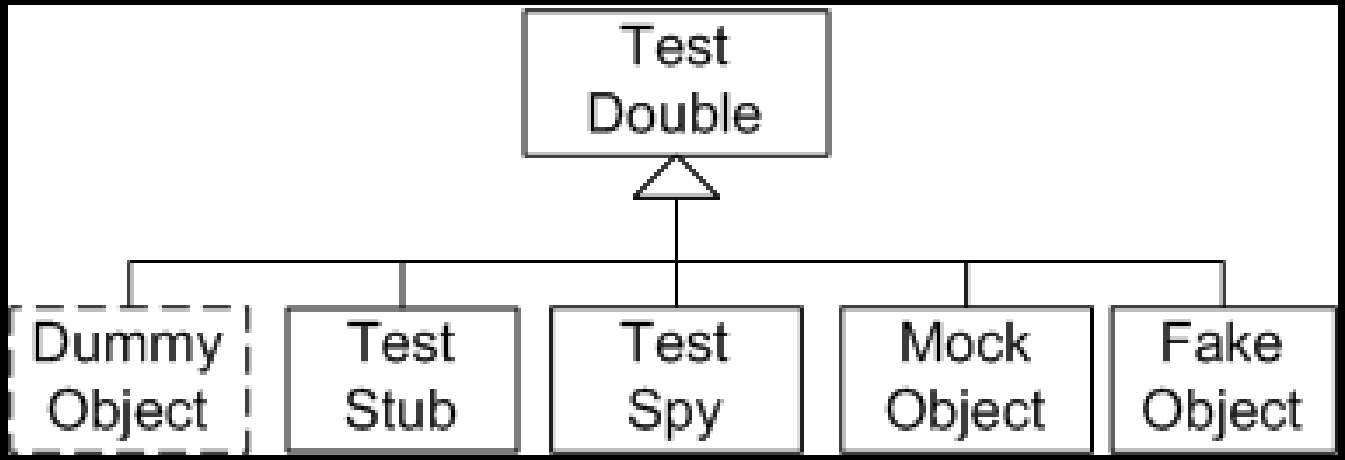
- Anti-Padrão: Adicionar código de teste ao sistema

Anti-Padrões

- Alterar o SUT para passar o teste
- Adequar o teste ao SUT, não o SUT ao teste
- Testar a implementação, não a funcionalidade

- Anti-Padrões de código fonte
- Cheiro => Anti-Padrão

Dublês


Test
Double
Dummy
Object
Test
Stub
Test
Spy
Mock
Object
Fake
Object

Contato

- http://www.agilcoop.org.br

- agilcoop@agilcoop.org.br
- paulocheque@agilcoop.org.br