

# Testes de Unidade

Paulo Cheque

**AgilCoop**

10/02/2009 Verão 2009



# Tipo de Teste

- Unidade: Classe/Módulo ou Método/Função
- Teste do código fonte com foco na funcionalidade
  - Erro comum em TDD
- Teste básico e muito importante
- Teste sólido e robusto
- Precisos: Descarta o trabalho de depuração

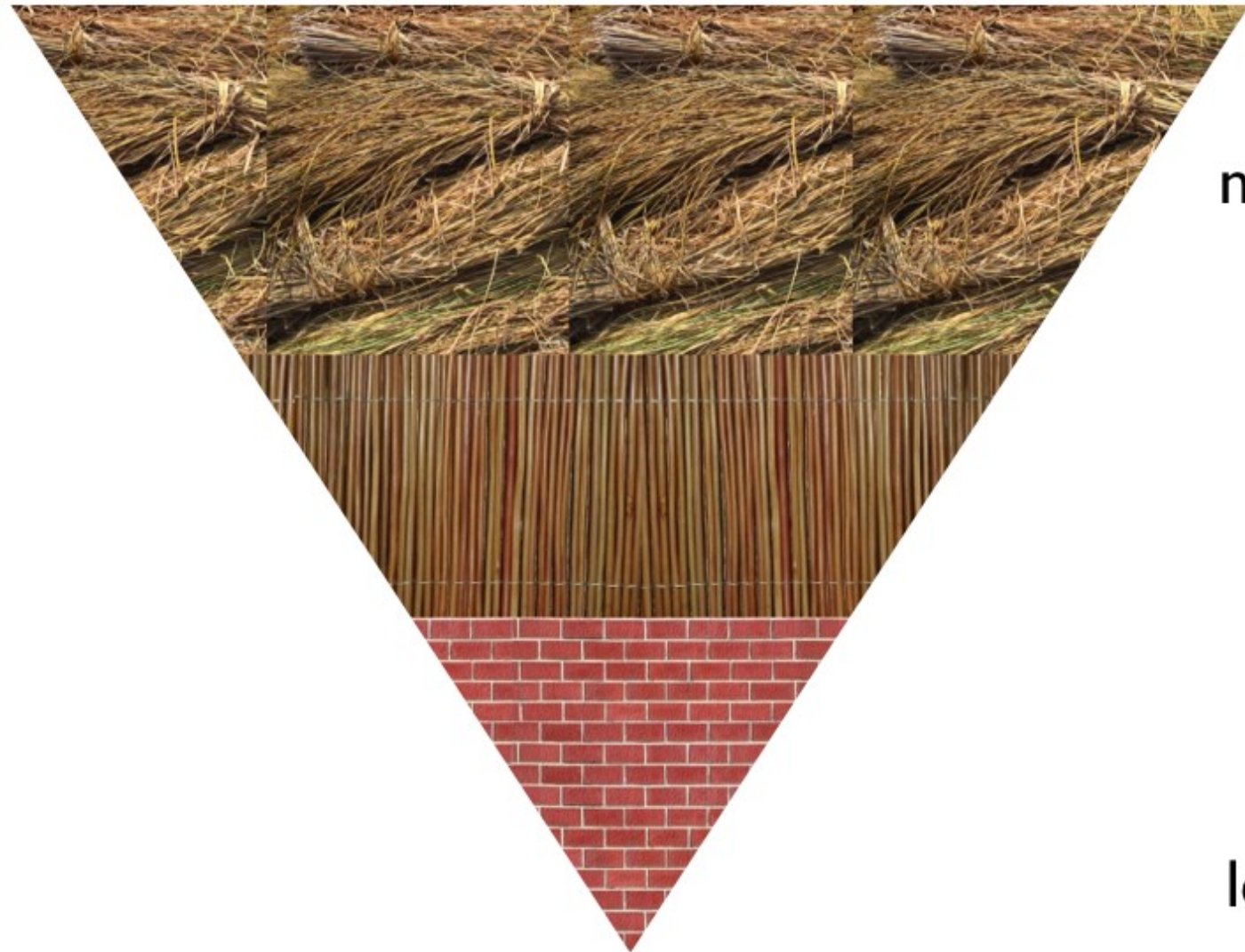
# Ótima Estratégia

GUI

end-2-end,  
integration, story,  
example, acceptance

unit/micro/isolation

# Péssima Estratégia



**bad:**  
most investment

**bad:**  
least investment

# Prática

Poker Texas Hold'Em (em Scala)

EXEMPLO

# Exemplo 1

```
34 @Test
35 public void algoritmoLentoTesteLento() {
36     long x = System.currentTimeMillis();
37     assertEquals(2, MathHelper.mdcAlgoritmoSuperLento(8364823434l, 836482343498123888l));
38     long y = System.currentTimeMillis();
39     System.out.println((y - x) / 1000); // ~ 13 minutos
40 }
```

```
556 @Test
557 public void iniciarTimerPausadoRestartaNormalmente() throws Exception {
558     DadoUmTimerPausado();
559     long elapsed = timer.elapsed();
560     timer.start();
561     Thread.yield();
562     Thread.sleep(40); // Corrida das threads
563     assertTrue(timer.running());
564     assertFalse(timer.stopped());
565     assertTrue(timer.elapsed() > elapsed);
566 }
```

# Razões

- Algoritmos pesados:
  - Matemática
  - Bioinformática
  - Computação gráfica
  - Otimização
- Alta complexidade computacional
- Algoritmos concorrentes
- Testes integrados (mini-integração) não isolados

# Rápido

- Centenas em poucos segundos
- Devem ser executados dezenas de vezes por dia, durante o desenvolvimento
  - Obter feedback rápido para não atrasar o desenvolvimento
- Solução, criar diferentes baterias de testes:
  - Bateria padrão
  - Bateria de testes pesados



# Exemplo 2

```
10 @Test
11 public void testaDownload() throws Exception {
12     WebUtils webUtils = new WebUtils(new URL("http://www.agilcoop.org.br"));
13     String slide = "portal/slides/cursos-de-verao-2009/AgilCoop-Verao2009-MetodosAgeisIntro.pdf";
14     File arquivo = webUtils.download(slide);
15     assertEquals(270107, arquivo.length());
16 }
17 }
```

# Isolado

- Não depender de outros casos de testes
- Não depender de fatores externos:
  - Servidores Web, FTP, etc
  - Sistema Operacional
  - Dia/Horário da execução do teste
  - ...

# Exemplo 3

```
8 public static String getDataFormatadaParaInscricao() {  
9     Date data = new Date();  
10    return new SimpleDateFormat("dd/MM/yyyy").format(data);  
11 }  
12  
13 }
```

```
8 @Test  
9 public void dadosFixos() throws Exception {  
10     assertEquals("10/02/2009", DateHelper.getDataFormatadaParaInscricao());  
11 }
```

# Repetitível

- Execução deve ser idêntica
- Mal Cheiro: Teste intermitente
- Cuidados com:
  - aleatoriedade
  - estatísticas
  - concorrência (race-condition)
  - etc
- Dicas p/ Refatoração: Isolar a lógica (Humble Objects)

# Exemplo 4

```
58 @Test
59 public void falsoPositivo1() {
60     try {
61         new Timer(0, 0, -1);
62     } catch (IllegalArgumentException e) {
63         assertTrue(true);
64     }
65 }
66
67 @Test
68 public void falsoPositivo2() {
69     try {
70         new Timer(0, 0, -1);
71         fail("Era pra lancar um erro");
72     } catch (Exception e) {
73     }
74 }
```

# Auto-Verifica

- Deve verificar/comparar algo: Passa ou Falha
- Evitar falsos positivos / falsos negativos
- Razão para existir: Evitar testes ambíguos

# Escrito nos momentos certos

- Concorrentemente com o código de produção
  - Antes
- Mais cedo possível
- Antes de uma refatoração
- Antes de mudanças em sistemas legados
  - Testes abrangentes e pouco específicos

# Teste de Qualidade

Fast

Isolated

Repeatable

Self-verifying

Timely

-- Brett e Tim (Object Mentor)

+ Útil



# Exemplo 6

```
20 @Test
21 public void testeInutil() {
22     Usuario u = new Usuario();
23     u.setNome("Fulano");
24     assertEquals("Fulano", u.getNome());
25 }
26 }
```

# Útil

- 1 teste => 1 mudança de código
- Deve ter uma razão para existir
- Evitar testes ambíguos

Obs: Muitos desenvolvedores ignoram testes de valores limites ou casos que acreditam que sejam ambíguos. Mesmo sendo, vale a pena verificar já que o custo da criação é muito pequeno!!!!

# Orientação a Objetos

- Classes abstratas
- Métodos de classe
- Protegidos => mesmo pacote
- Singletons (design perigoso)



# Teste a Funcionalidade

- Classes
  - Anônimas
  - Privadas
- Métodos
  - Privados
- Querer testar informações privadas significa falha no design
  - Não altere a visibilidade para ser testável!
  - Refatore para ser testável!

# Dicas

- Verificar valores limites
  - loops, comparações...

```
12  @Test
13  def menorFullHouseGanhaDeMaiorFlush() {
14      assertTrue(menorFullHouse() > maiorFlush())
15      assertTrue(maiorFlush() < menorFullHouse())
16  }
```

- Identificar conjuntos
- Casos de sucesso / fracasso
- Exceções
- Funções bijetoras: Testar ida e volta

# + Dicas

- Listas: Cheia, vazia, nula
- String: Vazia, nula, comprida, caracteres brancos (\t, \n, espaço ...)
- Números: 0, negativos, positivos, muitas casas decimais, valores mínimos/máximos (INT\_MAX)
- Expressão regular:
  - sequências/caracteres repetidas, caracteres acentuados e especiais, pontuação (pontos, vírgulas ...), colchetes, parenteses ...

# Algumas Ferramentas

- CxxTest (C++): <http://cxxtest.sourceforge.net>
- JUnit (Java): <http://www.junit.org>
- DUnit (Delphi): <http://dunit.sourceforge.net>
- VBUnit (Visual Basic): <http://www.vbunit.com>
- TestNG (Java): <http://testng.org>
- RSpec (Ruby): <http://rspec.info/>
  
- [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)

# +Ferramentas

- Só utilize em casos realmente pertinentes:
  - Testando métodos static
    - <https://jmockit.dev.java.net/>
  - Testando chamadas privadas
    - <http://sourceforge.net/projects/privaccessor>



# Contato

<http://www.agilcoop.org.br>

[agilcoop@agilcoop.org.br](mailto:agilcoop@agilcoop.org.br)

[paulocheque@agilcoop.org.br](mailto:paulocheque@agilcoop.org.br)

