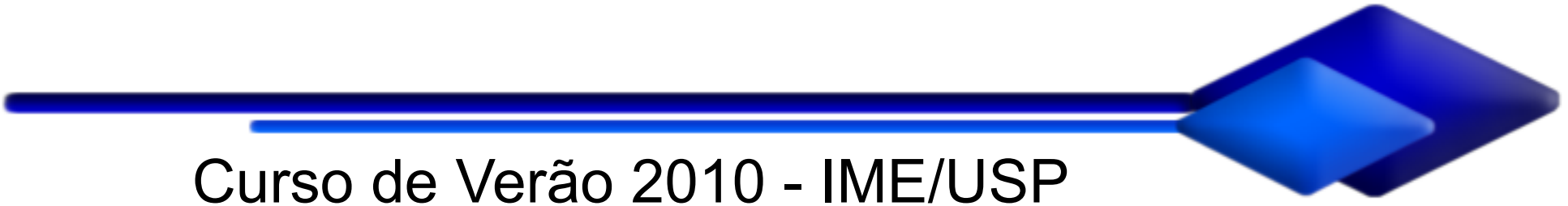
Testes de Unidade



Curso de Verão 2010 - IME/USP

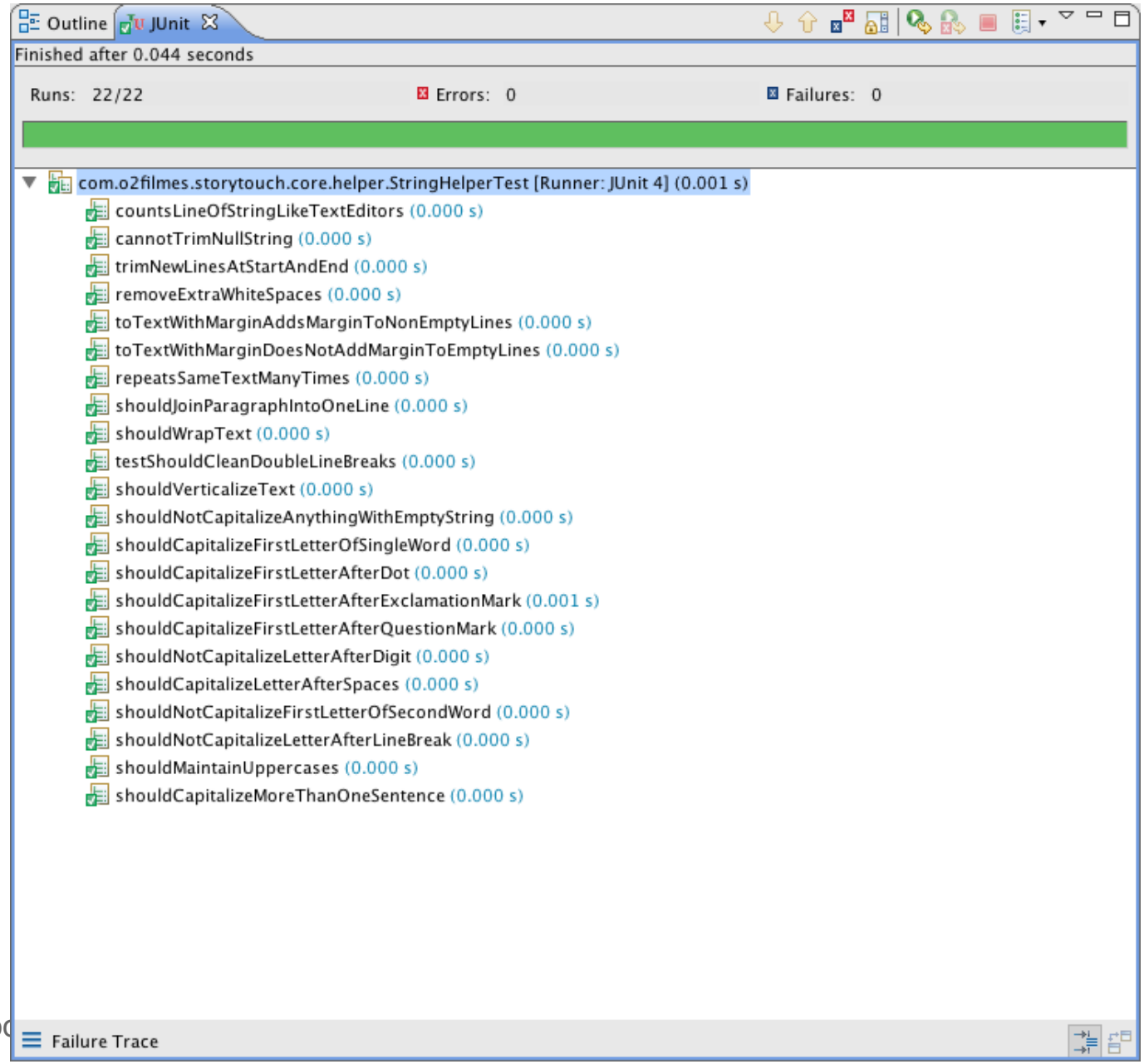
www.agilcoop.org.br

Hugo Corbucci

hugo@agilcoop.org.br

AgilCoop

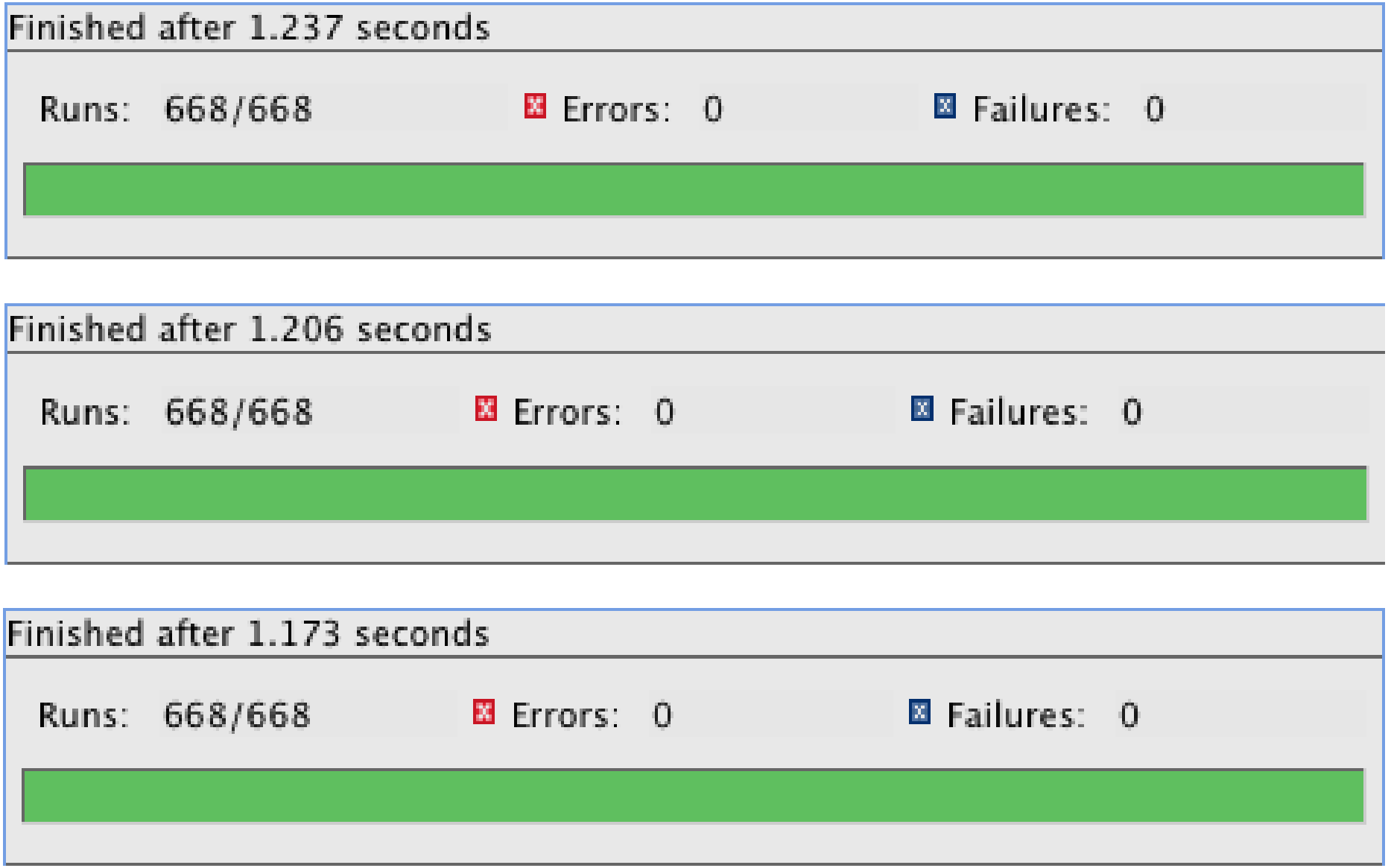
Caracterização



Caracterização

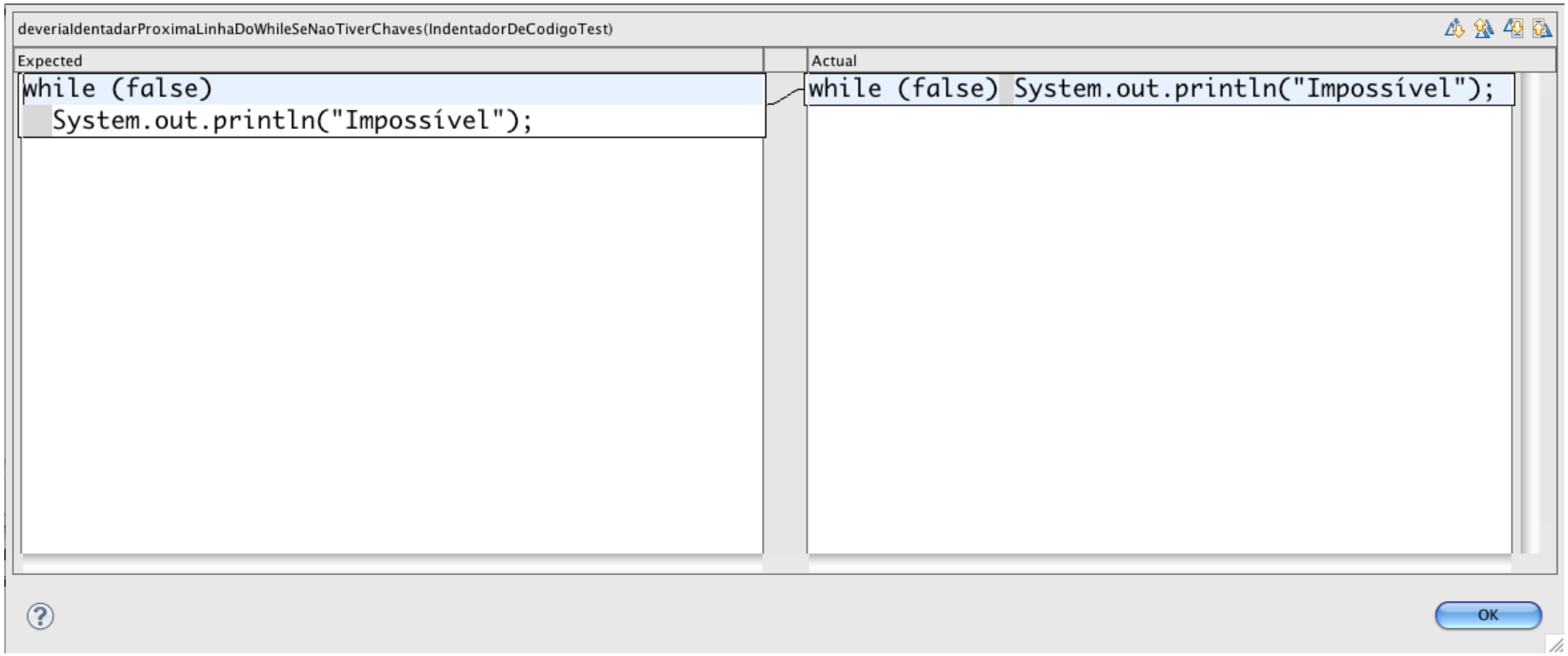
```
@Test
public void countsLineOfStringLikeTextEditors() throws Exception {
    assertEquals(0, StringHelper.countLinesOfString(null));

    assertEquals(1, StringHelper.countLinesOfString(""));
    assertEquals(1, StringHelper.countLinesOfString("3jsdh"));

    assertEquals(2, StringHelper.countLinesOfString("\n"));
    assertEquals(2, StringHelper.countLinesOfString("bla\n"));

    assertEquals(4, StringHelper.countLinesOfString("bla\n\n\n"));
    assertEquals(4, StringHelper.countLinesOfString("bla\n\n\nble"));

    assertEquals(5, StringHelper.countLinesOfString("\niuaea\nsudysuds\nhgdshgds\n"));
}
```

Caracterização



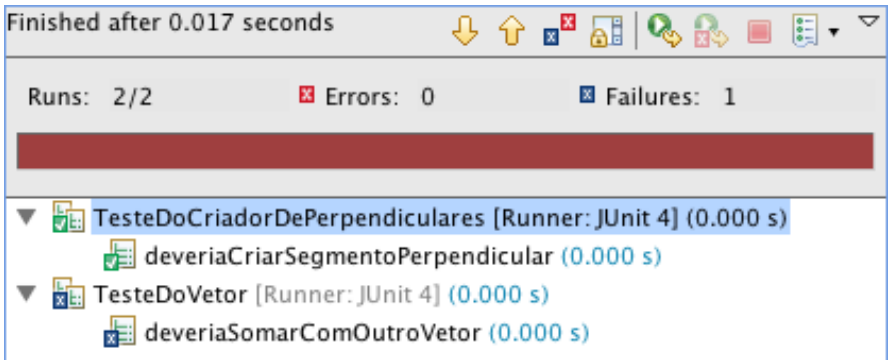
Caracterização

```
@Test
public void deveriaIdentadarProximaLinhaDoWhileSeNaoTiverChaves()
        throws Exception {
    String codigo = "while (false) System.out.println(\"Impossível\");";
    String esperado = "while (false)\n  System.out.println(\"Impossível\");\n";

    assertEquals(esperado, IndentadorDeCodigo.indentar(codigo));
}
```



Caracterização

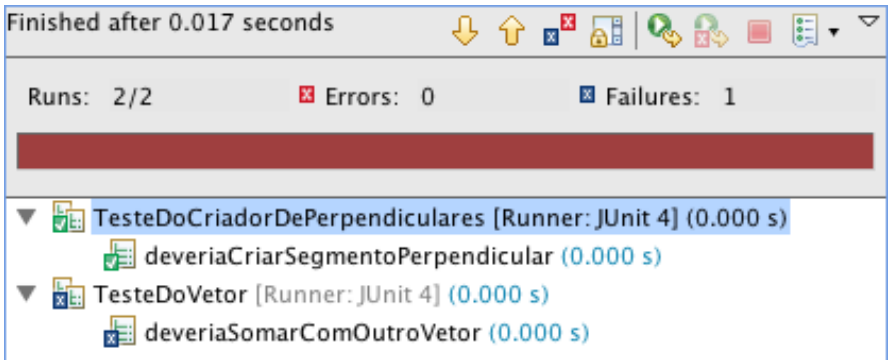
```
public class CriadorDeBissetrizes {
    public Vetor criaBissetrizDe(Vetor vetor, Vetor vetor2) {
        return vetor.soma(vetor2);
    }
}
```



Como?

Caracterização

```
public class CriadorDeBissetrizes {
    public Vetor criaBissetrizDe(Vetor vetor, Vetor vetor2) {
        return vetor.soma(vetor2);
    }
}   @Test
    public void deveriaCriarSegmentoPerpendicular() throws Exception {
        CriadorDeBissetrizes criador = new CriadorDeBissetrizes();
        Vetor vertical = new Vetor(0, 5) {
            @Override
            public Vetor soma(Vetor aSomar) {
                return new Vetor(5, 5);
            }
        };
        Vetor horizontal = new Vetor(5, 0) {
            @Override
            public Vetor soma(Vetor aSomar) {
                return new Vetor(5, 5);
            }
        };
        Vetor resultado = criador.criaBissetrizDe(vertical, horizontal);

        assertEquals(resultado.getDeltaX(), 5);
        assertEquals(resultado.getDeltaY(), 5);
    }
```



Ou seja....

- Testa uma Classe/Módulo, cada método/função
- Foco: Funcionalidade
- Baixo nível, básico mas muito importante
- Preciso: aponta o caso específico que dá problema
- Sólido e independente

Estratégias de testes

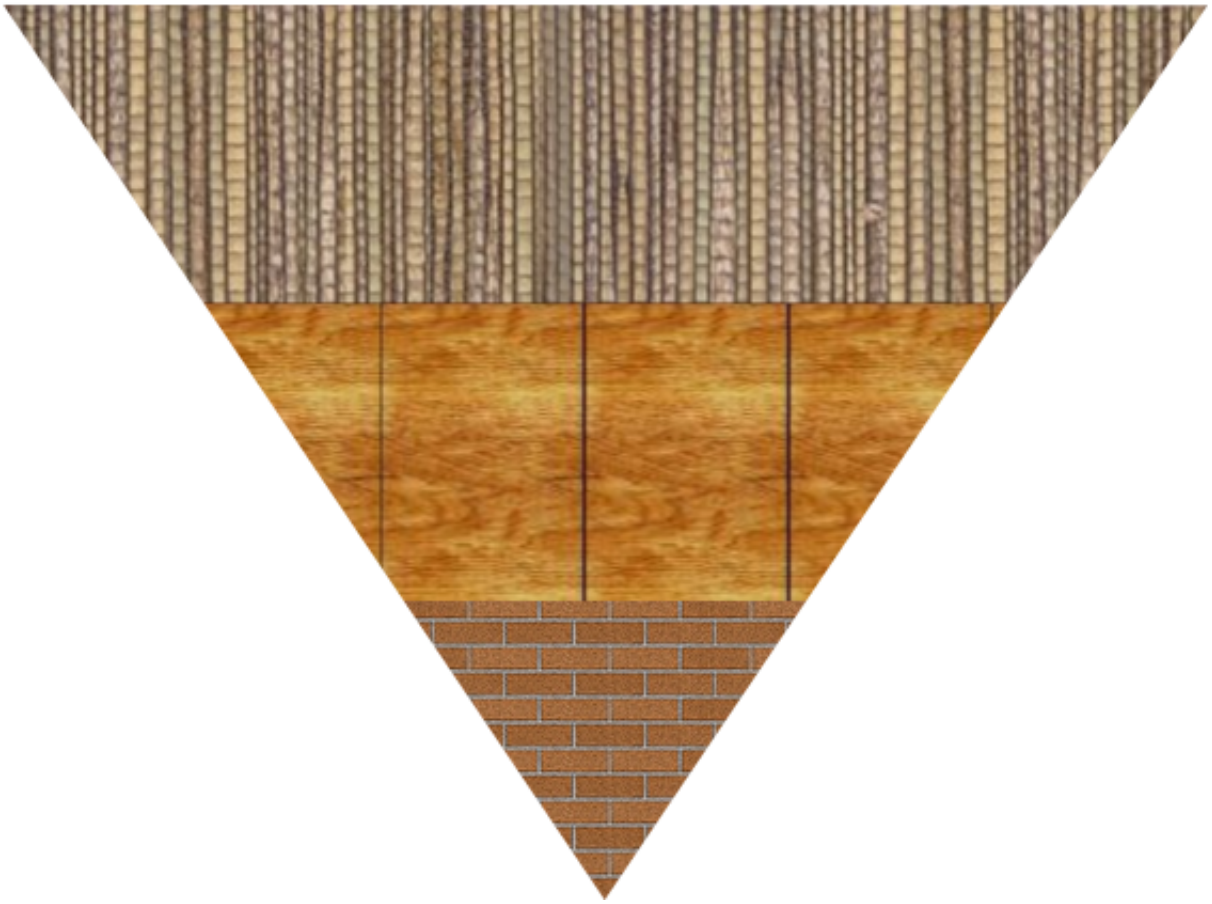
Testes de interface

Testes de integração, aceitação, história, etc.

Testes de unidade/micro/isolação

Estratégias de testes

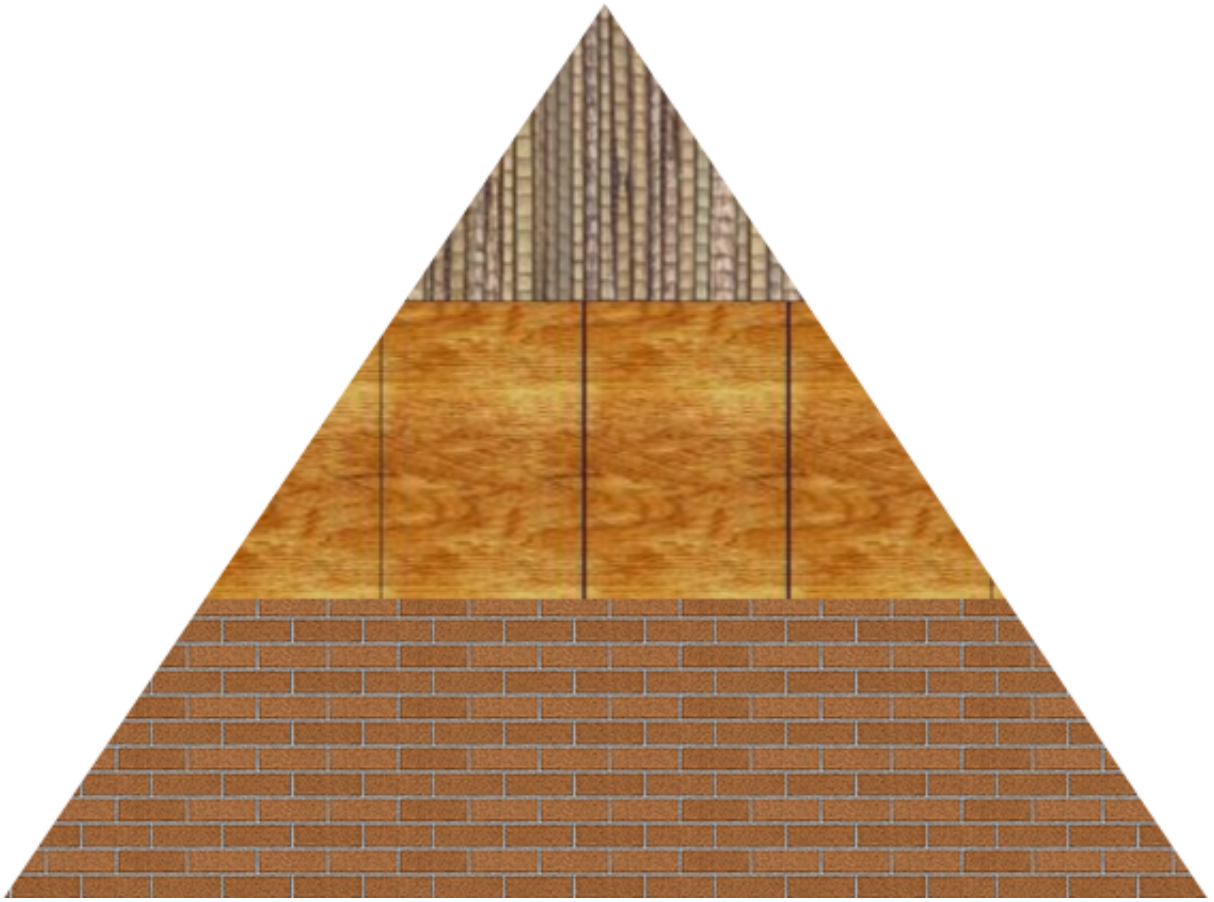
Muitos testes de interface



Poucos testes de unidade

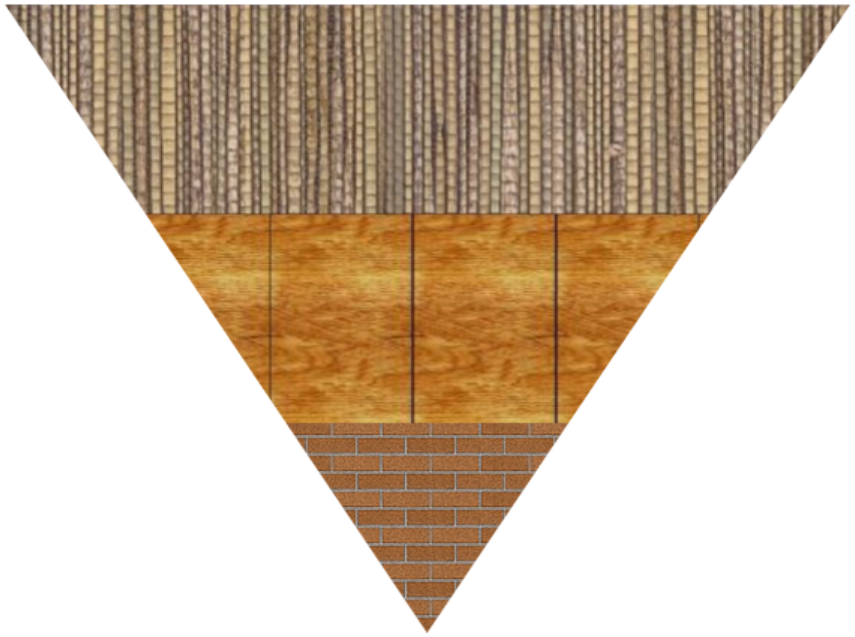
Estratégias de testes

Poucos testes de interface

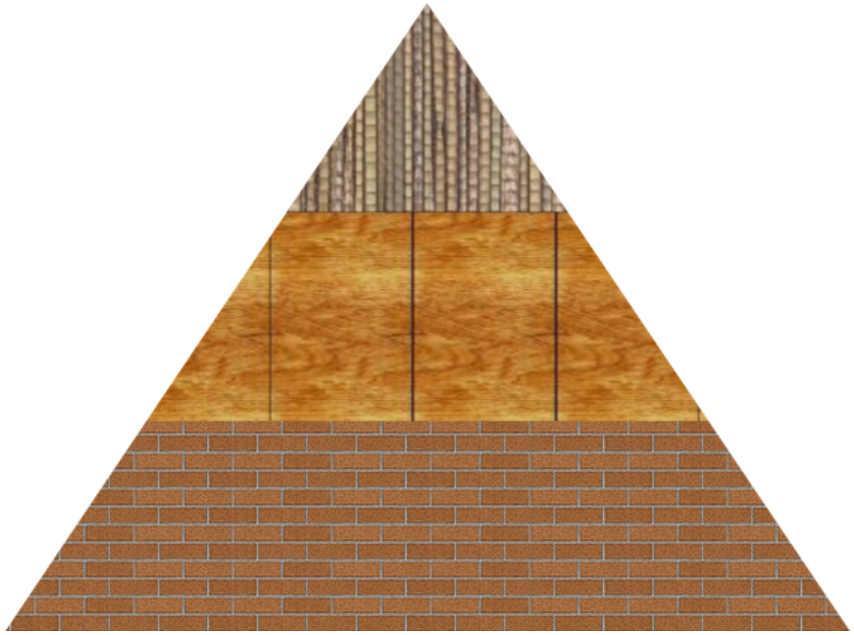


Muitos testes de unidade

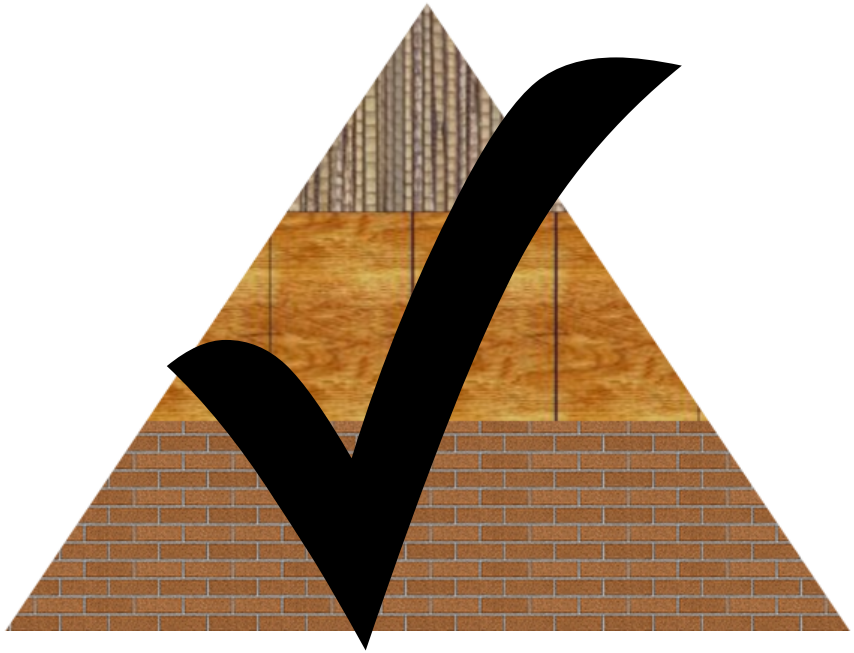
Estratégias de testes



ou



Estratégias de testes



Objetivos para Testes de Unidade

F ast (Rápidos)

I ndependent (Independentes)

R epeatable (Repetíveis)

S elf-verifying (Auto-verificantes)

T imely (Em tempo)

– Clean Code

Rápidos

```
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    }
```

Rápidos

```
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    }
```

Quantas vezes por dia você rodaria esses testes?

```
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    }
```

Rápidos

- Serão numerosos e você não pode esperar o dia todo para eles rodarem
- Na prática, há integração de unidades e os testes proveem feedback sobre as mudanças

Rápidos

- Então não faz testes que demoram?
- Apaga os testes que demoram?
- Ignora algoritmos pesados?

Rápidos

- Então não faz testes que demoram?
- Apaga os testes que demoram
- Melhora algoritmos lesados?

NÃO

O que pode tornar um teste lento?

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

Como lidar com esses casos?

- Separe-os do resto!

- Suites de Teste
  Pastas fontes diferentes
  Tarefas diferentes

Independentes

Esses testes passam?

```
@Test
public void vetoresOpostosSomadosValemVetorVazio() throws Exception {
    vetorAtual = new Vetor(5, 5).soma(new Vetor(-5, -5));
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}

@Test
public void vetorVazioMultiplicaPor10EhVazio() throws Exception {
    vetorAtual = vetorAtual.produtoEscalarCom(10);
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}
```

Independentes

Esses nessa ordem?

```
@Test
public void vetorVazioMultiplicaPor10EhVazio() throws Exception {
    vetorAtual = vetorAtual.produtoEscalarCom(10);
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}

@Test
public void vetoresOpostosSomadosValemVetorVazio() throws Exception {
    vetorAtual = new Vetor(5, 5).soma(new Vetor(-5, -5));
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}
```

Independentes

- Outros testes
  - Ordem de execução
  - Sucesso

```
@Test
public void vetoresOpostosSomadosValemVetorVazioQueMultiplicadoPor10EhVazio() throws Exception {
    vetorAtual = vetorAtual.soma(new Vetor(-vetorAtual.getDeltaX(),
            -vetorAtual.getDeltaY()));
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());

    vetorAtual = vetorAtual.produtoEscalarCom(10);
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}
```

Independentes

- De outros testes
  - Devem funcionar em qualquer ordem de execução

Independentes

- De outros testes
  - Devem funcionar em qualquer ordem de execução
  - Do sucesso dos anteriores

```
@Test
public void vetoresOpostosSomadosValemVetorVazioQueMultiplicadoPor10EhVazio() throws Exception {
    vetorAtual = new Vetor(5, 5).soma(new Vetor(-5, -5));
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());

    vetorAtual = vetorAtual.produtoEscalarCom(10);
    assertEquals(0, vetorAtual.getDeltaX());
    assertEquals(0, vetorAtual.getDeltaY());
}
```

Repetíveis

```
@Test
public void testaDownload() throws Exception {
    WebUtils webUtils = new WebUtils(new URL("http://www.agilcoop.org.br"));
    String slide = "portal/slides/cursos-de-verao-2010/AgilCoop-Verao2010-MetodosAgeisIntro.pdf";
    File arquivo = webUtils.download(slide);
    assertEquals(270107, arquivo.length());
}
```

O que precisa para esse teste passar?

Repetíveis

- Em vários ambientes
  - Com ou sem conexão externa: http, ftp, smtp, ...
  - Em produção, homologação ou desenvolvimento
  - Num servidor, no seu laptop, no micro de casa, ...

Repetíveis

```
@Test
public void sorteiaAmigoSecretoAleatoriamente() throws Exception {
    SorteadorDeAmigoSecreto sorteador = new SorteadorDeAmigoSecreto();
    sorteador.add("Paulo");
    sorteador.add("Hugo");
    sorteador.add("Mari");
    sorteador.add("Dairton");

    sorteador.sorteia();

    assertEquals("Paulo", sorteador.amigoDo("Hugo"));
    assertEquals("Dairton", sorteador.amigoDo("Paulo"));
    assertEquals("Mari", sorteador.amigoDo("Dairton"));
    assertEquals("Hugo", sorteador.amigoDo("Mari"));

    sorteador.sorteia();

    assertEquals("Paulo", sorteador.amigoDo("Hugo"));
    assertEquals("Mari", sorteador.amigoDo("Paulo"));
    assertEquals("Dairton", sorteador.amigoDo("Mari"));
    assertEquals("Hugo", sorteador.amigoDo("Dairton"));
}
```

Repetíveis

```
@Test
public void limiteParaSubmissoesEhDaquiAPouco() throws Exception {
    Calendar limite = Calendar.getInstance();
    limite.set(2010, 02, 28, 23, 59, 59);

    Calendar diferenca = Calendar.getInstance();
    diferenca.add(Calendar.DATE, 27);
    assertEquals(limite, diferenca);
}
```

Repetíveis

- Cuidado com situações não-determinísticas
  - Aleatoriedade
  - Tempo
  - Concorrência
  - etc...

Repetíveis

- Várias execuções dos testes deveriam ser idênticas
- Testes intermitentes são evidência de problema
- Isole a lógica e crie dublês

Auto-verificantes

```
@Test
public void deveriaDarErroAoCriarComValorInvalido() {
     new Timer(null);
     // Se não imprimir nada, está errado!
}
@Test
public void osAmigosDeveriamSerIguaisODoArquivo() {
    SorteadorDeAmigoSecreto sorteador = new SorteadorDeAmigoSecreto();
    sorteador.add("Paulo");
    sorteador.add("Hugo");
    sorteador.add("Mari");
    sorteador.add("Dairton");

    sorteador.sorteia();

    System.out.println(sorteador.amigoDo("Hugo"));
    System.out.println(sorteador.amigoDo("Paulo"));
    System.out.println(sorteador.amigoDo("Dairton"));
    System.out.println(sorteador.amigoDo("Mari"));
}
```

Auto-verificantes

OU

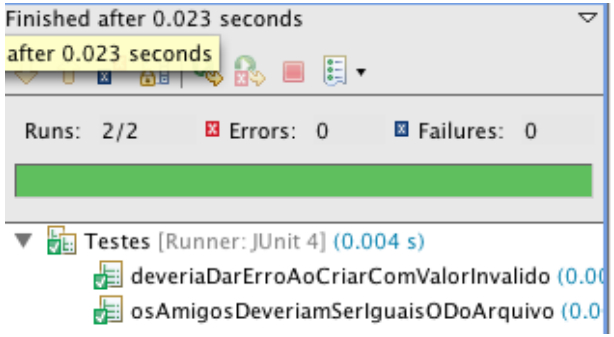
```
        @Test
        public void deveriaDarErroAoCriarComValorInvalido() {
            try {
                new Timer(null);
                fail("Deveria ter lançado uma excessão");
            } catch (Exception e) {
            }
        }
```

```
@Test
public void osAmigosDeveriamSerIguaisODoArquivo() {
    SorteadorDeAmigoSecreto sorteador = new SorteadorDeAmigoSecreto();
    sorteador.add("Paulo");
    sorteador.add("Hugo");
    sorteador.add("Mari");
    sorteador.add("Dairton");

    sorteador.sorteia();

    assertEquals("Paulo", sorteador.amigoDo("Hugo"));
    assertEquals("Dairton", sorteador.amigoDo("Paulo"));
    assertEquals("Mari", sorteador.amigoDo("Dairton"));
    assertEquals("Hugo", sorteador.amigoDo("Mari"));
}
```

Auto-verificantes

```
java.lang.NullPointerException
        at java.lang.Thread.setName(Thread.java:1020)
        at java.util.Timer.<init>(Timer.java:136)
        at TesteDoVetor.deveriaDarErroAoCriarComValorInvalido(TesteDoVetor.java:115)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:592)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:44)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:41)
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:28)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:31)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:73)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:46)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:180)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:41)
        at org.junit.runners.ParentRunner$1.evaluate(ParentRunner.java:173)
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:28)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:31)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:220)
        at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:46)
        at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:467)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:683)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:390)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:197)
Paulo
Mari
Hugo
Dairton
```

OU



Auto-verificantes

- Um teste tem UMA asserção
  Passa ou Falha
- Tem que falhar se estiver errado
- Tem que passar se estiver certo

Em tempo

- Muito antes e provavelmente precisará ser reescrito
- Muito depois e provavelmente nunca será escrito
- Junto com o código de produção
  - Antes, se possível em TDD

Em tempo

- E se já tiver código e não tiver testes ou estiverem incompletos?
- Para entender o sistema (testes de estudo)
- Antes de refatorar (não introduzir erros)
- Antes de mudar um comportamento

Objetivos para Testes de Unidade

F ast (Rápidos)

I ndependent (Independentes)

R epeatable (Repetíveis)

S elf-verifying (Auto-verificantes)

T imely (Em tempo)

– Clean Code

Além disso: Úteis!

Úteis

```
public class Usuario {
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Vale a pena testar isso?

Úteis

- Motivos para escrever um teste
  - Precisa mudar algo no código de produção
  - Descobriu um bug
  - Quer documentar um comportamento não evidente
  - Está em dúvida sobre o funcionamento

Casos especiais em O.O.

- Classes abstratas
  - Implementação simples pro teste
- Métodos de classe
- Protegidos => Teste no mesmo pacote
- Singletons

Sinais de problemas

- Vontade de testar
  - Classes Anônimas
  - Classes Privadas
  - Métodos Privados

- Sinal de que essas coisa auxiliares tem lógicas complexas demais. Refatore!

Dicas

- Verifique valores limites
  - Em loops (primeiro valor, último valor)
  - Em comparações (o menor de um tipo contra maior de outro)
- Identifique e agrupe conjuntos de teste com um mesmo objetivo
- Teste que tudo dá certo com coisas certas.
  E que dá errado com coisas erradas!

Dicas

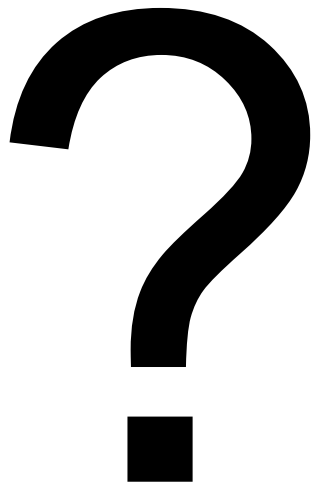
- Listas: cheias, vazias, nulas
- String: Vazias, nulas, grandes, caracteres estranhos
- Números: 0, negativos, positivos, grandes, pequenos, valores máximos e mínimos
- Expressões regulares: sequências repetidas, acentos, caracteres estranhos, pontuações

Ferramentas

- CxxTest (C++): http://cxxtest.tigris.org/
- 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

Perguntas



Hugo Corbucci

hugo@agilcoop.org.br

Mão na massa