Padrões de Testes Automatizados

Curso de Verão 2010 - IME/USP

www.agilcoop.org.br

Hugo Corbucci

hugo@agilcoop.org.br

AgilCoop

Introdução

```
@Test
public void deveriaSomarComOutroVetor() throws Exception {
    Vetor inicial = new Vetor(0, 5);
    Vetor aSomar = new Vetor(5, 0);

    Vetor resultado = inicial.soma(aSomar);

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

@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());
}
```

Introdução

- Código! De testes automatizados

Introdução

- Código! De testes automatizados

E o que Métodos Ágeis dizem que deve ser feito com código?

Introdução

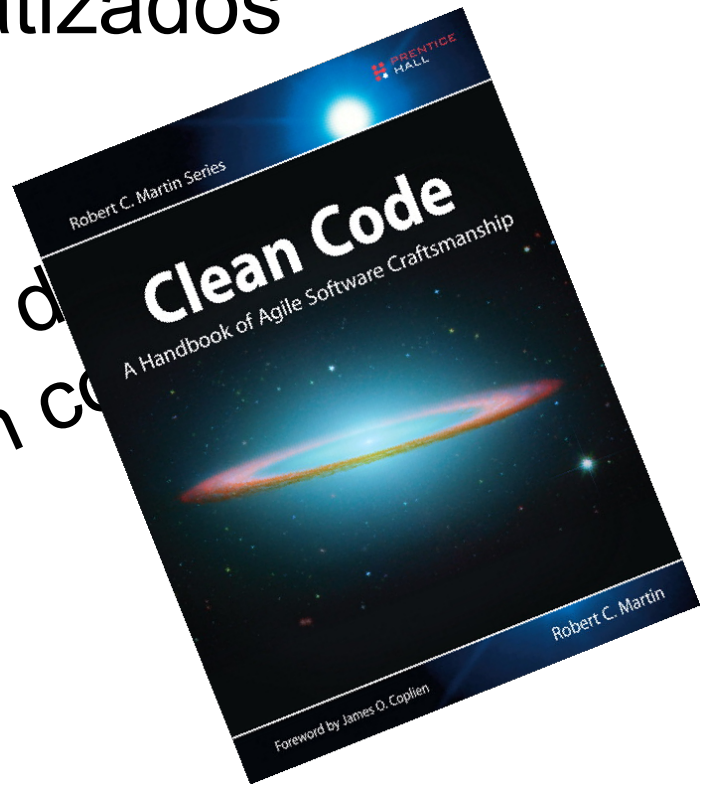
- Código! De testes automatizados

Métodos Ágeis dizem
eve ser feito com código?

Introdução

- Código! De testes automatizados

Métodos Ágeis d
eve ser feito com c



Introdução

- Código! De testes automatizados

Introdução

- O que vale pra código de produção vale pra código de testes automatizados

Introdução

- O que vale pra código de produção vale pra código de testes automatizados
  - Precisa ser mantido

Introdução

- O que vale pra código de produção vale pra código de testes automatizados
  - Precisa ser mantido
  - Precisa ser refatorado

Introdução

- O que vale pra código de produção vale pra código de testes automatizados
  - Precisa ser mantido
  - Precisa ser refatorado
  - Precisa ter um design (ainda que simples)

Introdução

- O que vale pra código de produção vale pra código de testes automatizados
  - Precisa ser mantido
  - Precisa ser refatorado
  - Precisa ter um design (ainda que simples)
  - Precisa ser expressivo

Introdução

- O que vale pra código de produção vale pra código de testes automatizados
  - Precisa ser mantido
  - Precisa ser refatorado
  - Precisa ter um design (ainda que simples)
  - Precisa ser expressivo
  - Pode ter erros

Por que Padrões?

“Um padrão de projeto é uma solução conhecida e reutilizável para um problema recorrente em design”

Padrões para testes

- Facilitam e otimizam a escrita dos testes

Padrões para testes

- Facilitam e otimizam a escrita dos testes
- Direcionam a refatoração do código de teste

Anti-Padrão

“Um anti-padrão é uma aparente solução a um problema recorrente que introduz outros problemas. Ou seja, um padrão cujo uso não é recomendado.”

Cheiros

"Sintomas do código fonte que indicam a presença de algum problema."

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

- Código obscuro

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

- Código obscuro
- Lógica de teste condicional

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

- Código obscuro
- Lógica de teste condicional
- Duplicação de código de teste

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

- Código obscuro
- Lógica de teste condicional
- Duplicação de código de teste
- Produção de Lógica de teste

Cheiros no código

```
@Test
public void testA() throws Exception {
    Vetor v1 = new Vetor(0,5);
    Vetor v2 = new Vetor(5,5);
    Vetor v3 = new Vetor(5,0);
    int x = (int) (Math.random() * 10 - 5);
    int y = (int) (Math.random() * 10 - 5);
    Vetor v4 = new Vetor(x,y);
    int x1 = (int) (Math.random() * 10 - 5);
    int y2 = (int) (Math.random() * 10 - 5);
    Vetor v5 = new Vetor(x1,y2);
    if(v5.length() > 5) {
        int x2 = (int) (Math.random() * 10 - 5);
        int y3 = (int) (Math.random() * 10 - 5);
        v5 = new Vetor(x2,y3);
    }

    if(x < 0 ^ y < 0)
        assertEquals(vetorAtual, v1.soma(v4));
    else if(x<0)
        assertEquals(v2, v3.soma(vetorAtual));
    else
        assertEquals(v1, v4.soma(v3));

    assertEquals(vetorAtual, v1.soma(v4));
}
```

- Código obscuro
- Lógica de teste condicional
- Duplicação de código de teste
- Produção de Lógica de teste
- Código difícil de testar

Cheiros de comportamento

- Roleta de asserções

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

Cheiros de comportamento

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

- Roleta de asserções
- Intervenção manual

```
@Test
public void testeComIntervencao() throws Exception {
    InputStreamReader inputStream = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(inputStream);
    String nome = reader.readLine();

    Decorador decorador = new DecoraComoVencedor(nome);
    assertEquals(nome + " é o vencedor", decorador.decora());
}
```

Cheiros de comportamento

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

- Roleta de asserções
- Intervenção manual
- Testes erráticos

```
@Test
public void testeComIntervencao() throws Exception {
    InputStreamReader inputStream = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(inputStream);
    String nome = reader.readLine();

    Decorador decorador = new DecoraComoVencedor(nome);
    assertEquals(nome + " é o vencedor", decorador.decora());
}
```

Cheiros de comportamento

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

- Roleta de asserções
- Intervenção manual
- Testes erráticos
- Testes frágeis

```
@Test
public void testeComIntervencao() throws Exception {
    InputStreamReader inputStream = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(inputStream);
    String nome = reader.readLine();

    Decorador decorador = new DecoraComoVencedor(nome);
    assertEquals(nome + " é o vencedor", decorador.decora());
}
```

Cheiros de comportamento

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

- Roleta de asserções
- Intervenção manual
- Testes erráticos
- Testes frágeis
- Debug frequente

```
@Test
public void testeComIntervencao() throws Exception {
    InputStreamReader inputStream = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(inputStream);
    String nome = reader.readLine();

    Decorador decorador = new DecoraComoVencedor(nome);
    assertEquals(nome + " é o vencedor", decorador.decora());
}
```

Cheiros de comportamento

```
public static void assertContentsAreLinkedInternally(PlacedContent... list) {
    for (int i = 0; i < list.length; i++) {
        if (i > 0)
            assertEquals(list[i - 1], list[i].getPrevious());

        if (i < list.length - 1)
            assertEquals(list[i + 1], list[i].getNext());
    }
}
```

- Roleta de asserções
- Intervenção manual
- Testes erráticos
- Testes frágeis
- Debug frequente
- Testes lentos

```
@Test
public void testeComIntervencao() throws Exception {
    InputStreamReader inputStream = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(inputStream);
    String nome = reader.readLine();

    Decorador decorador = new DecoraComoVencedor(nome);
    assertEquals(nome + " é o vencedor", decorador.decora());
}
```

Cheiros de projeto

- Testes bugados

Cheiros de projeto

- Testes bugados
- Testes não escrito por desenvolvedores

Cheiros de projeto

- Testes bugados
- Testes não escrito por desenvolvedores
- Alto custo de manutenção de testes

Cheiros de projeto

- Testes bugados
- Testes não escrito por desenvolvedores
- Alto custo de manutenção de testes
- Bugs de produção

Sugestões

- Nunca adicionar código de teste ao código de produção
  - Não usar a lógica dos testes no sistema

Sugestões

- Nunca adicionar código de teste ao código de produção
  - Não usar a lógica dos testes no sistema
- Não incluir código de testes em bibliotecas
  - Se precisar, crie outra biblioteca pros testes

Sugestões

- Nunca adicionar código de teste ao código de produção
  - Não usar a lógica dos testes no sistema
- Não incluir código de testes em bibliotecas
  - Se precisar, crie outra biblioteca pros testes
- Padronizar nomes de classes para facilitar a identificação e utilização de *scripts*
  - AlgumaClasse**Test** ou **TesteDe**AlgumaClasse

Padrões de Organização

- Baterias:
  - Named Test Suite

```
public class TestesComVetores {
    public static Test suite() {
        TestSuite suite = new TestSuite("Testes com vetores");
        // $JUnit-BEGIN$
        suite.addTest(new TesteDoVetor());
        suite.addTest(new TesteDoCriadorDePerpendiculares());
        // $JUnit-END$
        return suite;
    }
}
```

Padrões de Organização

- Baterias:
  - Named Test Suite
- Classes:
  - Uma classe de teste por Classe/Funcionalidade/Fixture

Padrões de Organização

- Baterias:
  - Named Test Suite
- Classes:
  - Uma classe de teste por Classe/Funcionalidade/Fixture
- Métodos:
  - Testes parametrizados
  - Superclasse de testes
  - Classe/Método de ajuda para testes

Verificações de Resultados

- Verificação de Estado
  - Inspecionar o sistema para saber se o estado está correto:

```
assertEquals(estadoEsperado, sistema.getEstado());
```

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
  - Sem estado. Usa um “espião” ou um “dublê”

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
  - Sem estado. Usa um “espião” ou um “dublê”

CUIDADO!
Verifique a funcionalidade e NÃO a implementação.

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
- Asserção personalizada
  - Criar seu próprio validador

```
public static Matcher<Integer> isLessThan(final int value) {
    return new BaseMatcher<Integer>() {
        public boolean matches(Object item) {
            return item instanceof Integer && ((Integer) item) < value;
        }

        public void describeTo(Description description) {
            description.appendText("a value less than ").appendValue(value);
        }
    };
}
```

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
- Asserção personalizada
- Asserção de mudanças
  - Verifico que a mudança que fiz tem o efeito esperado

```
assertEquals(tamanhoAnterior+1, lista.size());
```

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
- Asserção personalizada
- Asserção de mudanças
- Asserção de guarda
  - Asserções para condições iniciais

Verificações de Resultados

- Verificação de Estado
- Verificação de Comportamento
- Asserção personalizada
- Asserção de mudanças
- Asserção de guarda
- Asserção de teste inacabado
  - Marca de que o teste ainda não terminou

```
fail("Teste não implementado");
@Ignore("Produção ainda não está pronta")
```

Padrões de Valor

- Valores Literais
  - Testes NÃO fazem contas.
    Informação é inserida *hard-coded*.
  - Cuidado apenas para não tornar os testes não repetíveis

Padrões de Valor

- Valores Literais
  - Testes NÃO fazem contas.
    Informação é inserida *hard-coded*.
  - Cuidado para não tornar os testes não repetíveis
- Valores Derivados
  - Implementar um algoritmo que gere o valor esperado se for muito complexo
  - Cuidado para não ficar com uma cópia do código de produção

Padrões de Valor

- Valor Gerado
  - Gerar valores distintos a cada teste
  - Útil para testes de Integração
  - Cuidado para não conseguir mais repetir esse teste caso ele falhe

Padrões de Valor

- Valor Gerado
  - Gerar valores distintos a cada teste
  - Útil para testes de Integração
  - Cuidado para não conseguir mais repetir esse teste caso ele falhe
- Objeto Bobo
  - Só para não atrapalhar

SetUp: Zerados

- In-line set up
  - Set Up é pequeno e muda para cada teste então cada teste tem o seu

SetUp: Zerados

- In-line set up
  - Set Up é pequeno e muda para cada teste então cada teste tem o seu
- Delegated set up
  - SetUp distintos para cada teste num método auxiliar. Chamada explícita.

SetUp: Zerados

- In-line set up
  - Set Up é pequeno e muda para cada teste então cada teste tem o seu
- Delegated set up
  - SetUp distintos para cada teste num método auxiliar. Chamada explícita.
- Implicit set up
  - TestCase por Fixture
  - Uso de arcabouços (*frameworks*)

SetUp: Compartilhados

- Fixtures pré-montadas
- Set up preguiçoso
- Set up para suite baseada em Fixtures
- Decoradores de Set up

Estratégias para tear down

- Tear down para coletar lixo
  - Delete
  - frame.cleanUp()
  - System.gc()

Estratégias para tear down

- Tear down para coletar lixo
  - Delete
  - frame.cleanUp()
  - System.gc()
- Tear down automático
  - Guardo uma lista do que inseri e o tear down se vira pra tirar

Organização de tear down

- In-line tear down
  - Tear down é curto então cada teste cuida do seu
  - Sempre deve ficar depois das verificações

Organização de tear down

- In-line tear down
  - Tear down é curto então cada teste cuida do seu
  - Sempre deve ficar depois das verificações
- Tear down implícito
  - Uso de arcabouços (*frameworks*)

Padrões de arquitetura testável

- Injeção de dependência
  - Injeta por construtor
  - Injeta via setter/variáveis públicas

Padrões de arquitetura testável

- Injeção de dependência
- Busca de dependências (*lookup*)
  - Teste usa um repositório de objeto

Padrões de arquitetura testável

- Injeção de dependência
- Busca de Dependências (lookup)
- Humble Object
  - Existem características que dificultam testar um objeto: quebre-o e teste a parte com lógica

Padrões de arquitetura testável

- Injeção de dependência
- Busca de Dependências (lookup)
- Humble Object

- Anti-Padrão: Hook de teste

  if(TESTANDO) { … } else { ... }

  - Modifica o sistema para funcionar diferente ao executar o teste.

Anti-padrões

- Incluir código no sistema para uso exclusivo dos testes

```
// Usado apenas para testes
public Sistema() {}
```

Anti-padrões

- Incluir código no sistema para uso exclusivo dos testes
- Torturar o código do teste para encaixar no sistema ao invés de refatorar o sistema para permitir testes

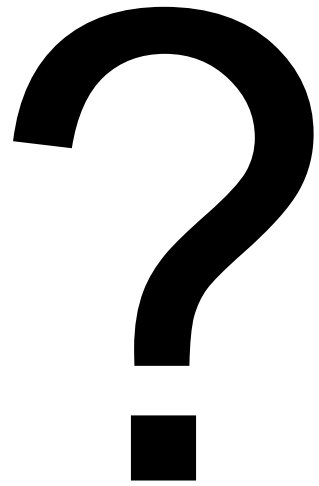
Anti-padrões

- Incluir código no sistema para uso exclusivo dos testes
- Torturar o código do teste para encaixar no sistema ao invés de refatorar o sistema para permitir testes
- Testar a implementação, não a funcionalidade

Anti-padrões

- Incluir código no sistema para uso exclusivo dos testes
- Torturar o código do teste para encaixar no sistema ao invés de refatorar o sistema para permitir testes
- Testar a implementação, não a funcionalidade
- Qualquer outro anti-padrão de código de produção

Perguntas



Hugo Corbucci

hugo@agilcoop.org.br