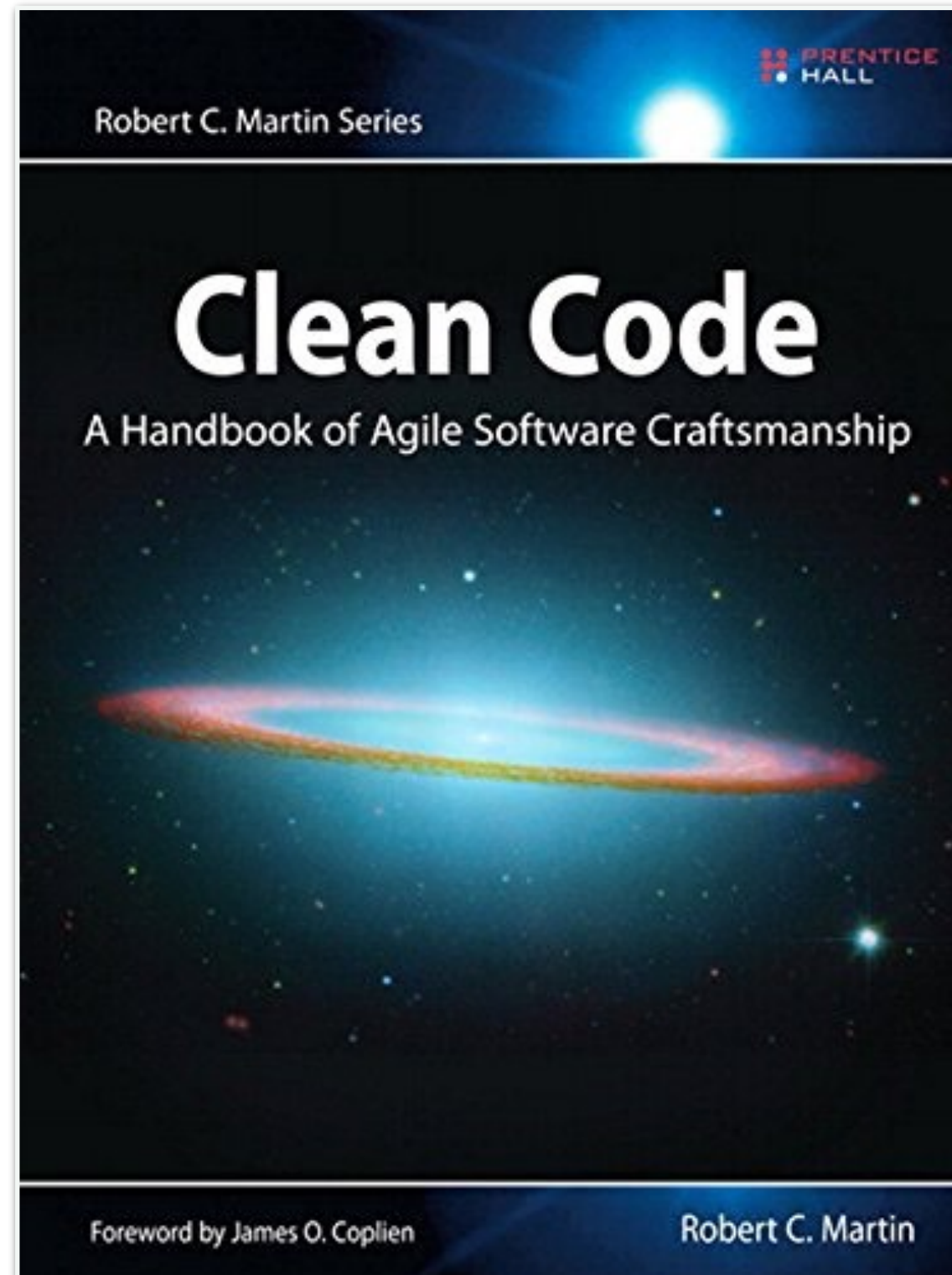


Engenharia de Software II

Código limpo - parte 3:
código externo e testes de unidade

Prof. André Hora
DCC/UFMG
2019.1



Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- **Código externo (cap 8)**
- **Testes de unidade (cap 9)**
- Classes (cap 10)

Código Externo

Código externo

- Raramente controlamos todo software relacionado ao nosso sistema
- Exemplos:
 - Importamos bibliotecas e frameworks
 - Utilizamos códigos open-source
 - Dependemos de componentes criados por outros times da própria empresa (que podem nem existir ainda)
- De certa forma, códigos externos devem ser integrado ao próprio código desenvolvido

Utilizando Código Externo

- Existe uma tensão natural entre produtores e usuários de interfaces
- **Produtores:** querem interfaces que funcionem em diversos ambientes para obter mais audiência
- **Usuários:** querem interfaces focadas em suas necessidades particulares

java.util.Map

- Interface ampla
- Interface com várias funcionalidades
- Mas também com deficiências (quais?)

```
• clear() void - Map
• containsKey(Object key) boolean - Map
• containsValue(Object value) boolean - Map
• entrySet() Set - Map
• equals(Object o) boolean - Map
• get(Object key) Object - Map
• getClass() Class<? extends Object> - Object
• hashCode() int - Map
• isEmpty() boolean - Map
• keySet() Set - Map
• notify() void - Object
• notifyAll() void - Object
• put(Object key, Object value) Object - Map
• putAll(Map t) void - Map
• remove(Object key) Object - Map
• size() int - Map
• toString() String - Object
• values() Collection - Map
• wait() void - Object
• wait(long timeout) void - Object
• wait(long timeout, int nanos) void - Object
```


java.util.Map

todo cliente pode deletar




**todo cliente pode adicionar
itens de qualquer tipo**



- `clear()` void - Map
- `containsKey(Object key)` boolean - Map
- `containsValue(Object value)` boolean - Map
- `entrySet()` Set - Map
- `equals(Object o)` boolean - Map
- `get(Object key)` Object - Map
- `getClass()` Class<? extends Object> - Object
- `hashCode()` int - Map
- `isEmpty()` boolean - Map
- `keySet()` Set - Map
- `notify()` void - Object
- `notifyAll()` void - Object
- `put(Object key, Object value)` Object - Map
- `putAll(Map t)` void - Map
- `remove(Object key)` Object - Map
- `size()` int - Map
- `toString()` String - Object
- `values()` Collection - Map
- `wait()` void - Object
- `wait(long timeout)` void - Object
- `wait(long timeout, int nanos)` void - Object


java.util.Map

```
Map<Sensor> sensors = new HashMap<Sensor>();  
Sensor s = sensors.get(sensorId);
```

- 
- Trecho de código repetido em diversas partes do sistema
 - Muito código para alterar se a interface de **Map** mudar
 - Mas a interface de **Map** nunca muda (?)
 - ...

java.util.Map

```
Map<Sensor> sensors = new HashMap<Sensor>( );  
Sensor s = sensors.get(sensorId);
```

- 
- Trecho de código repetido em diversas partes do sistema
 - Muito código para alterar se a interface de **Map** mudar
 - Mas a interface de **Map** nunca muda (?)
 - Mudou para suportar generics em Java 5


```
Map sensors = new HashMap( );
```

```
Sensor s = (Sensor)sensors.get(sensorId);
```

Java 5

java.util.Map

```
Map<Sensor> sensors = new HashMap<Sensor>();  
Sensor s = sensors.get(sensorId);
```

- 
- Trecho de código repetido em diversas partes do sistema
 - Muito código para alterar se a interface de **Map** mudar
 - Mas a interface de **Map** nunca muda (?)
 - Mudou para suportar generics em Java 5

```
Map sensors = new HashMap();
```

```
Sensor s = (Sensor)sensors.get(sensorId);
```

Java 5

Qual a solução?

java.util.Map

- Solução limpa: encapsular a interface utilizada (ao invés de espalhar)
- Classe **Sensors** melhora o projeto do sistema (melhor que **Map<Sensor>**)
- Vantagem: código mais fácil de entender e manter; controle da interface
- Se a interface de **Map** mudar, seu impacto no sistema será mínimo

```
public class Sensors {  
    private Map sensors = new HashMap();  
  
    public Sensor getById(String id) {  
        return (Sensor) sensors.get(id);  
    }  
  
    //snip  
}
```

java.util.Map

- Solução limpa: encapsular a interface utilizada (ao invés de espalhar)
- Classe **Sensors** melhora o projeto do sistema (melhor que **Map<Sensor>**)
- Vantagem: código mais fácil de entender e manter; controle da interface
- Se a interface de **Map** mudar, seu impacto no sistema será mínimo

```
public class Sensors {  
    private Map sensors = new HashMap();  
  
    public Sensor getById(String id) {  
        return (Sensor) sensors.get(id);  
    }  
  
    //snip  
}
```

- Claro, nem todo uso de **Map** deve ser transformado em classe
- Evitar **Map** quando for argumento ou retorno de APIs públicas

Exercício

Considerando APIs como classes, métodos e atributos públicos (ex: `java.util.ArrayList`):

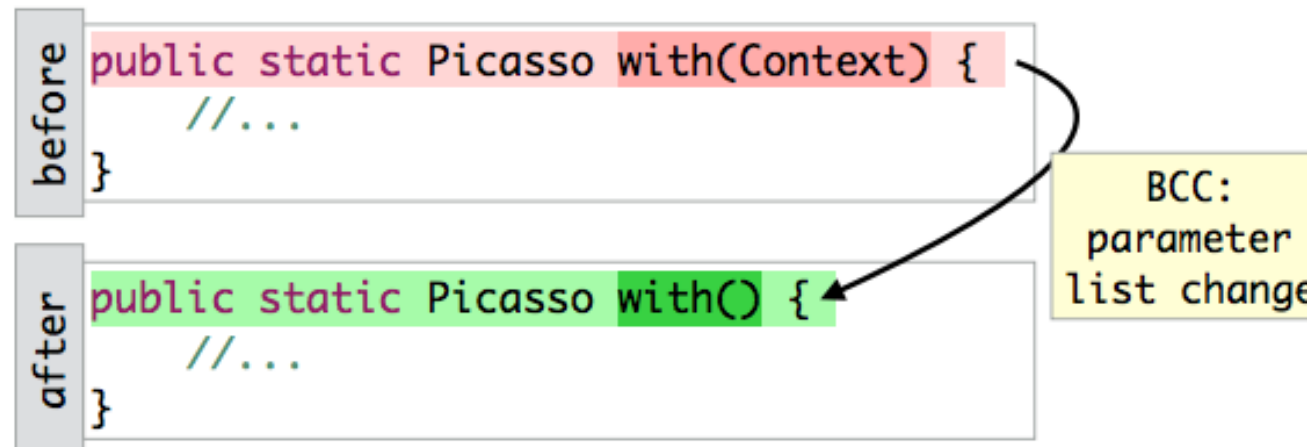
1. Apresente 10 formas de se quebrar o contrato de uma API (ex: remover um método público)
2. Imagine 3 razões para os desenvolvedores realizarem essa quebra de contrato

Quebra de Contrato de APIs

- APIs evoluem longo do tempo
- Mas devem evoluir com cautela, pois possuem clientes

Element	BCC
Type	REMOVE CLASS, CHANGE IN ACCESS MODIFIERS, CHANGE IN SUPERTYPE, ADD FINAL MODIFIER, REMOVE STATIC MODIFIER
Method	REMOVE METHOD, CHANGE IN ACCESS MODIFIERS, CHANGE IN RETURN TYPE, CHANGE IN PARAMETER LIST, CHANGE IN EXCEPTION LIST, ADD FINAL MODIFIER, REMOVE STATIC MODIFIER
Field	REMOVE FIELD, CHANGE IN ACCESS MODIFIERS, CHANGE IN FIELD TYPE, CHANGE IN FIELD DEFAULT VALUE, ADD FINAL MODIFIER

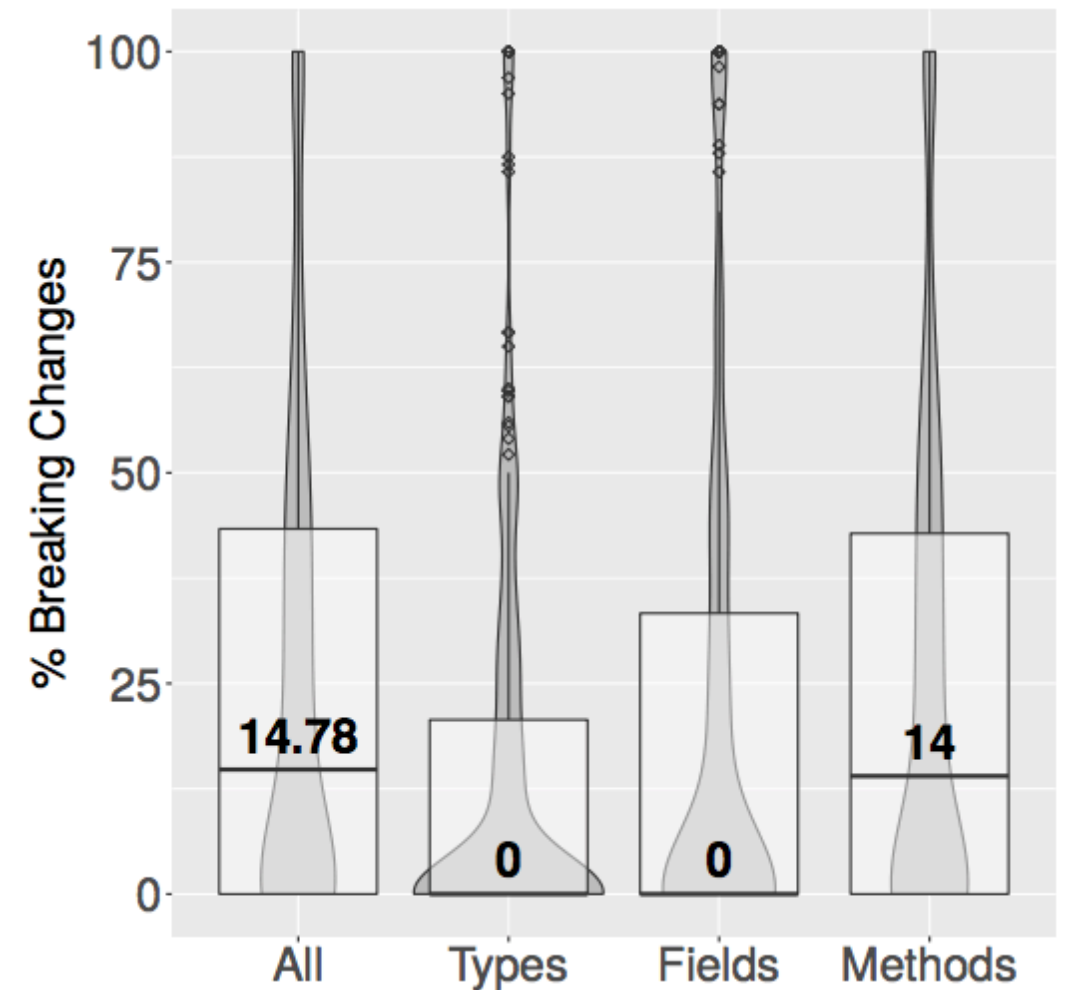
Quebra de Contrato de APIs



```
2  ...ee/backends/pipeline/DrawableFactory.java → ...agepipeline/drawable/DrawableFactory.java View
@@ -6,7 +6,7 @@
6  * LICENSE file in the root directory of this source tree.
   An additional grant
7  * of patent rights can be found in the PATENTS file in
   the same directory.
8  */
9  -package com.facebook.drawee.backends.pipeline;
10
11  import javax.annotation.Nullable;
12
6  * LICENSE file in the root directory of this source tree.
   An additional grant
7  * of patent rights can be found in the PATENTS file in
   the same directory.
8  */
9  +package com.facebook.imagepipeline.drawable;
10
11  import javax.annotation.Nullable;
12
```

Quebra de Contrato de APIs

- 28% de 500K alterações de APIs quebram compatibilidade
- Por sistema: 14.78%



Quebra de Contrato de APIs: Razões

Motivation	Description	Occur.
NEW FEATURE	BCs to implement new features	19
API SIMPLIFICATION	BCs to simplify and reduce the API complexity and number of elements	17
MAINTAINABILITY	BCs to improve the maintainability and the structure of the code	14
BUG FIXING	BCs to fix bugs in the code	3
OTHER	BCs not fitting the previous cases	6

Testes de Unidade

Testes de Unidade

- Código de teste é tão importante quanto de produção
 - Devem ser projetados e mantidos
 - Garante que alterações não quebram o código existente
- **Com testes:** código se mantém flexível, pois facilita alterações; sistema pode ser melhorado
- **Sem testes:** alteração é uma possível fonte de bugs

Teste Limpo

- O que torna o teste limpo?
- O mesmo que torna qualquer código limpo:
 - Legibilidade
 - Simplicidade
 - ...

Teste Limpo

Qual o problema deste teste?

```
public void testGetPageHierarchyAsXml() throws Exception
{

    crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));

    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(
            new FitNesseContext(root), request);
    String xml = response.getContent();

    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
}
```

Teste Limpo

Qual o problema deste teste?

```
public void testGetPageHieratchyAsXml() throws Exception
{
    crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));

    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(
            new FitNesseContext(root), request);
    String xml = response.getContent();

    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
}
```

- Duplicação
- Código complexo

Teste Limpo

Qual o problema deste teste?

```
public void testGetPageHierarchyAsXml() throws Exception  
{
```

```
    crawler.addPage(root, PathParser.parse("PageOne"));  
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));  
    crawler.addPage(root, PathParser.parse("PageTwo"));
```

```
    request.setResource("root");  
    request.addInput("type", "pages");  
    Responder responder = new SerializedPageResponder();  
    SimpleResponse response =  
        (SimpleResponse) responder.makeResponse(  
            new FitNesseContext(root), request);  
    String xml = response.getContent();
```

```
    assertEquals("text/xml", response.getContentType());  
    assertSubString("<name>PageOne</name>", xml);  
    assertSubString("<name>PageTwo</name>", xml);  
    assertSubString("<name>ChildOne</name>", xml);
```

```
}
```

- Duplicação
- Código complexo

```
public void testGetPageHierarchyAsXml() throws Exception {  
    makePages("PageOne", "PageOne.ChildOne", "PageTwo");  
  
    submitRequest("root", "type:pages");  
  
    assertResponseIsXML();  
    assertResponseContains(  
        "<name>PageOne</name>", "<name>PageTwo</name>", "<name>ChildOne</name>"  
    );  
}
```

Teste Limpo

Qual o problema deste teste?

```
public void testGetPageHieratchyAsXml() throws Exception  
{
```

```
    crawler.addPage(root, PathParser.parse("PageOne"));  
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));  
    crawler.addPage(root, PathParser.parse("PageTwo"));
```

```
    request.setResource("root");  
    request.addInput("type", "pages");  
    Responder responder = new SerializedPageResponder();  
    SimpleResponse response =  
        (SimpleResponse) responder.makeResponse(  
            new FitNesseContext(root), request);  
    String xml = response.getContent();
```

```
    assertEquals("text/xml", response.getContentType());  
    assertSubString("<name>PageOne</name>", xml);  
    assertSubString("<name>PageTwo</name>", xml);  
    assertSubString("<name>ChildOne</name>", xml);
```



- Duplicação
- Código complexo

```
public void testGetPageHierarchyAsXml() throws Exception {  
    makePages("PageOne", "PageOne.ChildOne", "PageTwo");  
  
    submitRequest("root", "type:pages");  
  
    assertResponseIsXML();  
    assertResponseContains(  
        "<name>PageOne</name>", "<name>PageTwo</name>", "<name>ChildOne</name>"  
    );  
}
```



```

public void testGetPageHieratchyAsXmlDoesntContainSymbolicLinks()
throws Exception {

    WikiPage pageOne = crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));

    PageData data = pageOne.getData();
    WikiPageProperties properties = data.getProperties();
    WikiPageProperty symLinks = properties.set(SymbolicPage.PROPERTY_NAME);
    symLinks.set("SymPage", "PageTwo");
    pageOne.commit(data);

    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(
            new FitNesseContext(root), request);
    String xml = response.getContent();

    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
    assertNotSubString("SymPage", xml);
}

```

Qual o problema
deste teste?

```
public void testGetPageHierarchyAsXmlDoesntContainSymbolicLinks()  
throws Exception {
```

```
    WikiPage pageOne = crawler.addPage(root, PathParser.parse("PageOne"));  
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));  
    crawler.addPage(root, PathParser.parse("PageTwo"));
```

```
    PageData data = pageOne.getData();  
    WikiPageProperties properties = data.getProperties();  
    WikiPageProperty symLinks = properties.set(SymbolicPage.PROPERTY_NAME);  
    symLinks.set("SymPage", "PageTwo");  
    pageOne.commit(data);
```

```
    request.setResource("root");  
    request.addInput("type", "pages");  
    Responder responder = new SerializedPageResponder();  
    SimpleResponse response =  
        (SimpleResponse) responder.makeResponse(  
            new FitNesseContext(root), request);  
    String xml = response.getContent();
```

```
    assertEquals("text/xml", response.getContentType());  
    assertSubString("<name>PageOne</name>", xml);  
    assertSubString("<name>PageTwo</name>", xml);  
    assertSubString("<name>ChildOne</name>", xml);  
    assertNotSubString("SymPage", xml);
```

```
}
```

Qual o problema
deste teste?

```
public void testGetPageHierachyAsXmlDoesntContainSymbolicLinks()
throws Exception {
```

```
    WikiPage pageOne = crawler.addPage(root, PathParser.parse("PageOne"));
    crawler.addPage(root, PathParser.parse("PageOne.ChildOne"));
    crawler.addPage(root, PathParser.parse("PageTwo"));
```

```
    PageData data = pageOne.getData();
    WikiPageProperties properties = data.getProperties();
    WikiPageProperty symLinks = properties.set(SymbolicPage.PROPERTY_NAME);
    symLinks.set("SymPage", "PageTwo");
    pageOne.commit(data);
```

```
    request.setResource("root");
    request.addInput("type", "pages");
    Responder responder = new SerializedPageResponder();
    SimpleResponse response =
        (SimpleResponse) responder.makeResponse(
            new FitNesseContext(root), request);
    String xml = response.getContent();
```

```
    assertEquals("text/xml", response.getContentType());
    assertSubString("<name>PageOne</name>", xml);
    assertSubString("<name>PageTwo</name>", xml);
    assertSubString("<name>ChildOne</name>", xml);
    assertNotSubString("SymPage", xml);
```

```
}
```

Qual o problema deste teste?

```
public void testSymbolicLinksAreNotInXmlPageHierarchy() throws Exception {
    WikiPage page = makePage("PageOne");
    makePages("PageOne.ChildOne", "PageTwo");

    addLinkTo(page, "PageTwo", "SymPage");

    submitRequest("root", "type:pages");

    assertResponseIsXML();
    assertResponseContains(
        "<name>PageOne</name>", "<name>PageTwo</name>",
        "<name>ChildOne</name>"
    );
    assertResponseDoesNotContain("SymPage");
}
```

Teste Limpo

```
@Test
    public void turnOnLoTempAlarmAtThreashold() throws Exception {
        hw.setTemp(WAY_TOO_COLD);
        controller.tic();
        assertTrue(hw.heaterState());
        assertTrue(hw.blowerState());
        assertFalse(hw.coolerState());
        assertFalse(hw.hiTempAlarm());
        assertTrue(hw.loTempAlarm());
    }
```


Teste Limpopo

```
@Test
    public void turnOnLoTempAlarmAtThreashold() throws Exception {
        hw.setTemp(WAY_TOO_COLD);
        controller.tic();
        assertTrue(hw.heaterState());
        assertTrue(hw.blowerState());
        assertFalse(hw.coolerState());
        assertFalse(hw.hiTempAlarm());
        assertTrue(hw.loTempAlarm());
    }
```

Teste Limpo

```
@Test
public void turnOnLoTempAlarmAtThreashold() throws Exception {
    hw.setTemp(WAY_TOO_COLD);
    controller.tic();
    assertTrue(hw.heaterState());
    assertTrue(hw.blowerState());
    assertFalse(hw.coolerState());
    assertFalse(hw.hiTempAlarm());
    assertTrue(hw.loTempAlarm());
}
```

Evitar detalhes

```
@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```


Teste Limpopo

```
@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    hw.setTemp(WAY_TOO_COLD);
    controller.tic();
    assertTrue(hw.heaterState());
    assertTrue(hw.blowerState());
    assertFalse(hw.coolerState());
    assertFalse(hw.hiTempAlarm());
    assertTrue(hw.loTempAlarm());
}
```

Evitar detalhes

```
@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```

{heater, blower, cooler, hi-temp-alarm, lo-temp-alarm}

Facilita leitura e criação de testes...

```
@Test
public void turnOnCoolerAndBlowerIfTooHot() throws Exception {
    tooHot();
    assertEquals("hBChl", hw.getState());
}

@Test
public void turnOnHeaterAndBlowerIfTooCold() throws Exception {
    tooCold();
    assertEquals("HBchl", hw.getState());
}

@Test
public void turnOnHiTempAlarmAtThreshold() throws Exception {
    wayTooHot();
    assertEquals("hBCHl", hw.getState());
}

@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```

Facilita leitura e criação de testes...

```
@Test
public void turnOnCoolerAndBlowerIfTooHot() throws Exception {
    tooHot();
    assertEquals("hBChl", hw.getState());
}
```

```
@Test
public void turnOnHeaterAndBlowerIfTooCold() throws Exception {
    tooCold();
    assertEquals("HBchl", hw.getState());
}
```

```
@Test
public void turnOnHiTempAlarmAtThreshold() throws Exception {
    wayTooHot();
    assertEquals("hBCHl", hw.getState());
}
```

```
@Test
public void turnOnLoTempAlarmAtThreshold() throws Exception {
    wayTooCold();
    assertEquals("HBchL", hw.getState());
}
```

```
public String getState() {
    String state = "";
    state += heater ? "H" : "h";
    state += blower ? "B" : "b";
    state += cooler ? "C" : "c";
    state += hiTempAlarm ? "H" : "h";
    state += loTempAlarm ? "L" : "l";
    return state;
}
```

Um assert por teste

- Uma diretriz determina que cada método de teste deve ter apenas um assert
- **Objetivo:** acelerar e facilitar o entendimento do teste
- Logo, devemos evitar casos como:

```
Address a = new Address("ADDR1$ADDR2$CITY IL 60563$COUNTRY");  
assertEquals("ADDR1", a.getAddress());  
assertEquals("CITY IL 60563", a.getCity());  
assertEquals("Country", a.getCountry());
```

```
Address a = new Address("ADDR1$ADDR2$CITY IL 60563$COUNTRY");  
assertEquals("ADDR1", a.getAddress());  
assertEquals("CITY IL 60563", a.getCity());  
assertEquals("Country", a.getCountry());
```

Solução?

```
Address a = new Address("ADDR1$ADDR2$CITY IL 60563$COUNTRY");
assertEquals("ADDR1", a.getAddress());
assertEquals("CITY IL 60563", a.getCity());
assertEquals("Country", a.getCountry());
```

Solução

```
public class AddressTest extends TestCase {

    private Address anAddress;

    protected void setUp() throws Exception {
        anAddress = new Address("ADDR1$ADDR2$CITY IL 60563$COUNTRY");
    }

    public void testAddress() throws Exception {
        assertEquals("ADDR1", anAddress.getAddress());
    }

    public void testCity() throws Exception {
        assertEquals("CITY IL 60563", anAddress.getCity());
    }

    public void testCountry() throws Exception {
        assertEquals("COUNTRY", anAddress.getCountry());
    }
}
```

Um conceito por teste

- Outra diretriz determina que cada método de teste deve ter apenas um conceito
- **Objetivo:** criar métodos focados e evitar método longos

Um conceito por teste

```
public void testAddMonths() {  
    SerialDate d1 = SerialDate.createInstance(31, 5, 2004);  
  
    SerialDate d2 = SerialDate.addMonths(1, d1);  
    assertEquals(30, d2.getDayOfMonth());  
    assertEquals(6, d2.getMonth());  
    assertEquals(2004, d2.getYYYY());  
  
    SerialDate d3 = SerialDate.addMonths(2, d1);  
    assertEquals(31, d3.getDayOfMonth());  
    assertEquals(7, d3.getMonth());  
    assertEquals(2004, d3.getYYYY());  
  
    SerialDate d4 = SerialDate.addMonths(1, SerialDate.addMonths(1, d1));  
    assertEquals(30, d4.getDayOfMonth());  
    assertEquals(7, d4.getMonth());  
    assertEquals(2004, d4.getYYYY());  
}
```


Um conceito por teste

```
public void testAddMonths() {  
    SerialDate d1 = SerialDate.createInstance(31, 5, 2004);  
  
    SerialDate d2 = SerialDate.addMonths(1, d1);  
    assertEquals(30, d2.getDayOfMonth());  
    assertEquals(6, d2.getMonth());  
    assertEquals(2004, d2.getYYYY());  
  
    SerialDate d3 = SerialDate.addMonths(2, d1);  
    assertEquals(31, d3.getDayOfMonth());  
    assertEquals(7, d3.getMonth());  
    assertEquals(2004, d3.getYYYY());  
  
    SerialDate d4 = SerialDate.addMonths(1, SerialDate.addMonths(1, d1));  
    assertEquals(30, d4.getDayOfMonth());  
    assertEquals(7, d4.getMonth());  
    assertEquals(2004, d4.getYYYY());  
}
```



Exercício

- Apresente 3 características de um bom conjunto de testes de unidade (ex: ser legível)

FIRST

- **Fast:** testes devem ser rápidos; rodar frequentemente
- **Independent:** testes não devem depender de outros; para evitar efeito cascata (se um falha, outros falham)
- **Repeatable:** testes devem ser repetíveis em qualquer ambiente (produção e laptop), em qualquer ordem; rodar N vezes e obter o mesmo resultado
- **Self-Validating:** testes devem ter saída booleana; ou passam ou falham (sem avaliação manual de logs)
- **Timely:** testes devem ser escritos antes do código (TDD)