Introdução a Teste de Software
– Teste Funcional –

Esses slides usam o conteúdo do livro Software Testing: From Theory to Practice, de Maurício Aniche (Universidade Técnica de Delft), disponível online no endereço sttp.site.

Tarefas do Teste de Software

- De forma simplificada, o processo de testar de software é composto de duas tarefas:
  - Definir os casos de teste
  - Executar os casos de teste

Definição de Casos de Teste

- A experiência conta muito para definir os casos de teste
- Porém, só a experiência não é suficiente
  - O desenvolvedor pode esquecer algum caso de teste
  - Varia de pessoa para pessoa
  - Sem um critério, é difícil decidir quando parar

Definição de Casos de Teste

- Que casos de teste, com base em sua experiência, você definiria para o programa de números romanos?

Definição de Casos de Teste

- Que casos de teste, com base em sua experiência, você definiria para o programa de números romanos?
  - Número com um dígito: I = 1, V = 5
  - Número com múltiplos dígitos diferentes: XVI = 16
  - Números com dígitos repetidos: III = 3
  - Números com subtração de dígitos: IX = 9
  - Dígito + subtração de dígitos: XIX = 19
  - Número inválido: VX, XXC

Definição de Casos de Teste

Existem técnicas que apoiam a definição de casos de teste de forma mais sistemática.

Teste Baseado na Especificação

- Muito conhecidos também como Teste Funcional ou Teste Caixa Preta
- Usa os requisitos do programa para definir os casos de testes
- Deriva uma série de entradas de forma que cada entrada exercita um **partição** do programa.

Testes Baseados na Especificação

Para achar um bom conjunto de casos de teste, também chamado de **suíte de teste**, dividimos o espaço de entradas em **partições**:

- Cada partição é única, no sentido de que duas não exercitam o mesmo comportamento;
- Podemos facilmente verificar se o comportamento relativo a determinada entrada é correto ou não.

Testes Baseados na Especificação

Particionando o espaço de entrada

Requisito: Ano bissexto

Dado um ano específico, o programa deve retornar *verdadeiro* se o ano for bissexto e *falso* se não for.

Um ano é bissexto se:

- o ano for divisível por 4;
- e o ano não for divisível por 100;
- exceto quando o ano for divisível por 400 (que nesse caso é ano bissexto).

Testes Baseados na Especificação

Particionando o espaço de entrada

Partições para os requisitos do programa do ano bissexto:

- Ano é divisível por 4, mas não é divisível por 100 = é ano bissexto (**verdadeiro**)
- Ano é divisível por 4, divisível por 100 e divisível por 400 = é bissexto (**verdadeiro**)
- Ano não é divisível por 4 = não é ano bissexto (**falso**)
- Divisível por 4, divisível por 100, mas não é divisível por 400 = não é ano bissexto (**falso**)

Testes Baseados na Especificação

Particionando o espaço de entrada

Partições para os requisitos do programa do ano bissexto:

- Ano é divisível por 4, mas não é divisível por 100 = **verdadeiro**
  - Ex.: 2016, 2020
- Ano é divisível por 4, divisível por 100 e divisível por 400 = **verdadeiro**
  - Ex.: 2000
- Ano não é divisível por 4 = **falso**
  - 2015, 2017
- Divisível por 4, divisível por 100, mas não é divisível por 400 = **falso**
  - 1500, 1900

Testes Baseados na Especificação

```
public class LeapYear {

  public boolean isLeapYear(int year) {
    if (year % 400 == 0)
      return true;
    if (year % 100 == 0)
      return false;

    return year % 4 == 0;
  }
}
```

Testes Baseados na Especificação

Testando a primeira partição

Ano é divisível por 4, mas não é divisível por 100 = **verdadeiro**

- Ex.: 2016, 2020, 2024, 2028 …

```
@Test
public void divisibleBy4_notDivisibleBy100() {
  LeapYear leapYear = new LeapYear();
  boolean leap = leapYear.isLeapYear(2016);
  Assertions.assertTrue(leap);
}
```

Nesse caso, usamos 2016. E os outros anos?

São equivalentes, pois exercitam o mesmo comportamento.

Basta testar com um deles apenas.

Particionamento por Equivalência

Testes Baseados na Especificação

```
public class LeapYearTests {

  private final LeapYear leapYear = new LeapYear();

  @Test
  public void divisibleBy4_notDivisibleBy100() {
    boolean leap = leapYear.isLeapYear(2016);
    assertTrue(leap);
  }

  @Test
  public void divisibleBy4_100_400() {
    boolean leap = leapYear.isLeapYear(2000);
    assertTrue(leap);
  }

  @Test
  public void notDivisibleBy4() {
    boolean leap = leapYear.isLeapYear(39);
    assertFalse(leap);
  }

  @Test
  public void divisibleBy4_and_100_not_400() {
    boolean leap = leapYear.isLeapYear(1900);
    assertFalse(leap);
  }
}
```

Testes Baseados na Especificação

Método Partição por Categorias

Uma forma mais sistemática de definir os casos de teste, baseado nas características dos parâmetros de entrada e suas combinações.

Testes Baseados na Especificação

Método Partição por Categorias

1. Identifique os parâmetros de entrada do programa. Por exemplo, os parâmetros que o método recebe;
2. Identifique as características do parâmetros.
   - A partir das especificações. Por exemplo, um `int year` pode ser um número inteiro positivo entre 0 e infinito;
   - Outras não estão nas especificações. Por exemplo, uma entrada não pode ser `null`.
3. Identifique restrições que permitem minimizar o número de casos de teste
   - Identifique combinações inválidas entre valores dos diferentes parâmetros;
   - Pode não ser necessário testar comportamento excepcional (ex.: valor null) de um parâmetro com todos os valores dos outros parâmetros;
4. Gere combinações dos valores de cada parâmetro.

Testes Baseados na Especificação

Método Partição por Categorias

Requisito: Desconto de Natal

O sistema deve dar um desconto de 25% no dia de Natal.

O método tem dois parâmetros: o preço total das compras no carrinho, e a data atual. Quando não é Natal, ele simplesmente retorna o preço original, caso contrário, aplica o desconto.

Testes Baseados na Especificação

Seguindo o Método Partição por Categorias:

1) Temos dois parâmetros: o preço total e a data atual
2) Definimos as características de cada parâmetro

   Data: a única característica é que ela pode ser Natal ou não;

   Preço: (i) pode ser um número positivo, (ii) em certas circunstâncias, pode ser zero, e (iii) tecnicamente, pode ser um número negativo, mas esse é um caso excepcional.

3) O número de parâmetros e características não é grande nesse caso. Mas, como preço negativo é um caso excepcional, podemos testá-lo apenas em uma combinação, ou com data do Natal ou com data normal.
4) Combinamos as outras características

Testes Baseados na Especificação

Seguindo o Método Partição por Categorias:

4) Combinamos as outras características

- Preço positivo no Natal
- Preço positivo em outra data
- Preço 0 no Natal
- Preço 0 em outra data
- Preço negativo no Natal

  * Vejam que só combinamos “preço negativo” com um dos valores do outro parâmetro.

  O próximo passo é implementar os casos de teste

Exemplo: Barras de Chocolate

- Um pacote pode conter um certo número de barras de chocolate, que podem ser pequenas (1 quilo cada) ou grandes (5 quilos cada)
- Assumindo que o pacote é sempre preenchido com o maior número possível de barras grande, retorne o número de barras pequenas necessárias para encher a caixa. E retorne -1 se não é possível encher o pacote completamente.
- A entrada do programa é: número de barras pequenas disponíveis, número de barras grandes disponíveis, e a quantidade de quilos que o pacote suporta.

Exemplo: Barras de Chocolate

Uma possível implementação para esse programa:

```
public class ChocolateBars {

    public static final int CANNOT_PACK_BAG = -1;

    public int calculate(int small, int big, int total) {
        int maxBigBoxes = total / 5;
        int bigBoxesWeCanUse = Math.min(maxBigBoxes, big);
        total -= (bigBoxesWeCanUse * 5);

        if(small <= total)
            return CANNOT_PACK_BAG;
        return total;

    }
}
```

Exemplo: Barras de Chocolate

- Nesse exemplo as partições são menos claras
- Uma forma de analisar o problema para derivar as partições é considerar como as entradas afetam a saída.
  - Três parâmetros: número de barras pequenas, número de barras grandes e número de quilos do pacote.
    - São todos inteiros que podem ir de 0 a infinito
  - Dado um número de quilos válido, a saída depende do número de barras pequenas e do número de barras grandes. Então, só é possível analisar essas variáveis juntas.

Exemplo: Barras de Chocolate

Partições

- Apenas barras pequenas são usadas.
- Apenas barras grandes são usadas.
- Barras pequenas e barras grandes são usadas.
- Não há barras suficientes

- Caso excepcional: número negativo em qualquer um dos parâmetros.

Exemplo: Barras de Chocolate

Casos de Teste

- **Apenas barras pequenas:** small = 4, big = 2, total = 3
- **Apenas barras grandes:** small = 5, big = 3, total = 10
- **Barras pequenas e barras grandes:** small = 5, big = 3, total = 17
- **Não há barras suficientes:** small = 1, big = 1, total = 10
- **Número negativo nos parâmetros:** small = -1, big = -1, total = -1

Exemplo: Barras de Chocolate

```
public class ChocolateBarsTest {
    private final ChocolateBars bars = new ChocolateBars();

    @Test
    void notEnoughBars() {
        assertEquals(-1, bars.calculate(1, 1, 10));
    }

    @Test
    void onlyBigBars() {
        assertEquals(0, bars.calculate(5, 3, 10));
    }

    @Test
    void bigAndSmallBars() {
        assertEquals(2, bars.calculate(5, 3, 17));
    }

    @Test
    void onlySmallBars() {
        assertEquals(3, bars.calculate(4, 2, 3));
    }

    @Test
    void invalidValues() {
      assertEquals(-1, bars.calculate(-1, -1, -1));
    }
}
```