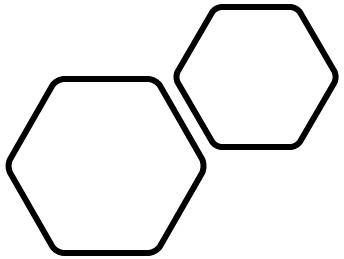
Introdução ao Teste Unitário



Prof. Bernardo Copstein

Baseado no livro “**Software Testing: From Theory to Practice**”

Maurice Aniche e Arie Van Deursen

Porque se preocupar com teste de software?

- Requisitos:
  - Dada uma lista de números inteiros o software deve retornar o maior e o menor elemento da lista.
- Solução:
  - percorrer a lista procurando por valores menores e ou maiores que os já armazenados

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            else if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

Como verificar se está correto?

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            else if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

```
public class NumFinderMain {
  public static void main (String[] args) {
      NumFinder nf = new NumFinder();

      nf.find(new int[] {4, 25, 7, 9});
      System.out.println(nf.getLargest());
      System.out.println(nf.getSmallest());

      nf.find(new int[] {4, 3, 2, 1});
      System.out.println(nf.getLargest());
      System.out.println(nf.getSmallest());
  }
}
```

- Normalmente os desenvolvedores fazem pequenas verificações
- Que resultados serão encontrados neste caso?

Como verificar se está correto?

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            else if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

```
public class NumFinderMain {
  public static void main (String[] args) {
      NumFinder nf = new NumFinder();

      nf.find(new int[] {4, 25, 7, 9});
      System.out.println(nf.getLargest());
      System.out.println(nf.getSmallest());

      nf.find(new int[] {4, 3, 2, 1});
      System.out.println(nf.getLargest());
      System.out.println(nf.getSmallest());
  }
}
```

→ 25 e 4

→ **-2147483648, 1**

Onde está o problema?

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            else if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

- Porque a sequencia 4,3,2,1 retorna o resultado incorreto?

Identificação do problema:

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

- O “else if” deveria ser apenas um “if”
- Caso contrário qualquer sequência em ordem decrescente irá gerar resultados incorretos

Solução:

```
public class NumFinder {
    private int smallest = Integer.MAX_VALUE;
    private int largest = Integer.MIN_VALUE;

    public void find(int[] nums) {
        for(int n : nums) {
            if(n < smallest)
                smallest = n;
            if (n > largest)
                largest = n;
        }
    }
    public int getSmallest(){return smallest;}
    public int getLargest(){return largest;}
}
```

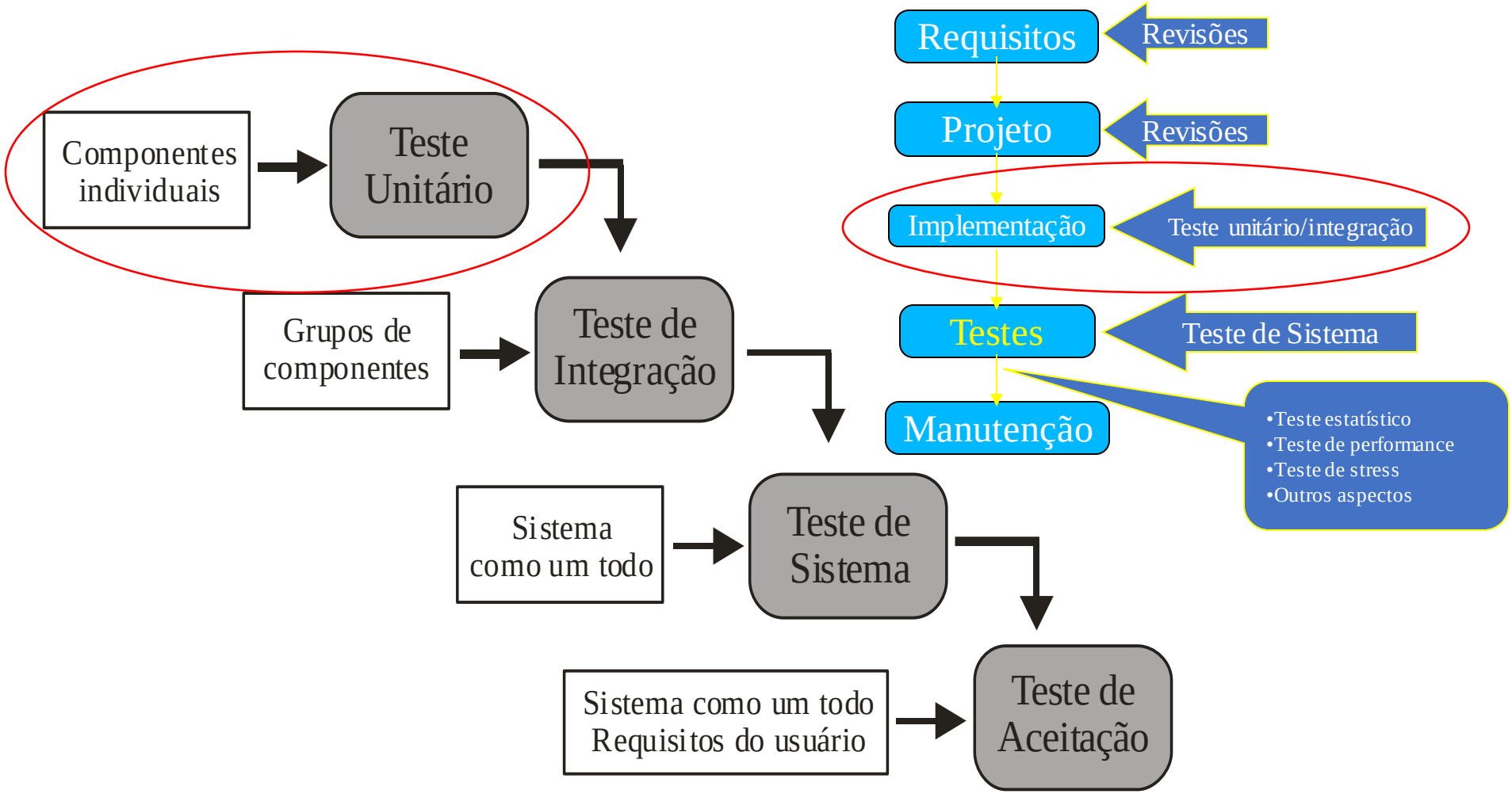
- Agora o método está correto !!

Concluindo

Precisamos testar software

Porque bugs ocorrem. E eles podem ter um grande impacto nas nossas vidas !!

Teste Unitário



O que vem a ser teste unitário

- Teste unitário é o teste desenvolvido pelos próprio desenvolvedores durante a etapa de construção do código
- É focado no teste das unidades de código individuais
  - No caso de programas POO estas unidades correspondem a classes
  - Então estamos falando do teste de classes individuais
- O teste da interação entre classes corresponde ao que chamamos de teste de integração
  - Também é desenvolvido pelos desenvolvedores durante a construção do código
  - Pode usar as mesmas ferramentas de automação
  - Muda apenas o objetivo: testar a interação entre as classes

Resumindo

- O nivel de teste unitário é baseado na construção de classes “driver”.
- Para cada classe que se deseja testar deve-se definir um conjunto de casos de teste que exercitem a classe alvo buscando identificar defeitos.
- Para cada conjunto de casos de teste deve-se implementar uma ou mais classes “driver”. Cada método destas classes implementam um caso de teste.
- A execução do teste unitário implica na execução dos métodos de teste que compõem as classes drivers e na geração automática dos relatórios correspondentes.

Como testar?

- Já se viu que para testar uma classe é necessário escrever um trecho de código que "exercita" o código a ser testado
- No exemplo da rotina pra achar o maior e o menor de uma lista de números criamos um pequeno "main" em Java que dispara dois casos de teste
- Este tipo de estratégia foi muito utilizado quando os primeiros desenvolvedores se deram conta que era importante manter o código usado para testar uma classe
  - Criava-se um método "main" para cada classe desenvolvida.
  - Bastava executar este "programa" para conferir se a classe continuava atendendo a especificação

Exemplo 2: conversão de numerais romanos

- O requisito indica a necessidade de uma classe capaz de converter numerais romanos
- O código ao lado deveria atender esta especificação

```
public class RomanNumeral {
  private static Map<Character, Integer> map;
  static {
    map = new HashMap<Character, Integer>();
    map.put('I',1); map.put('V',5); map.put('X',10); map.put('L',50);
    map.put('C',100); map.put('D',500); map.put('M',1000);
  }
  public int convert(String s) {
    int convertedNumber = 0;
    for(int i = 0; i < s.length(); i++) {
      int currentNumber = map.get(s.charAt(i));
      int next = i+1 < s.length() ? map.get(s.charAt(i+1)) : 0;

      if(currentNumber >= next)
          convertedNumber += currentNumber;
      else
          convertedNumber -= currentNumber;
    }
    return convertedNumber;
  }
}
```

Um conjunto de casos de teste

Defina um conjunto de casos de teste para o método de conversão de literais romanos

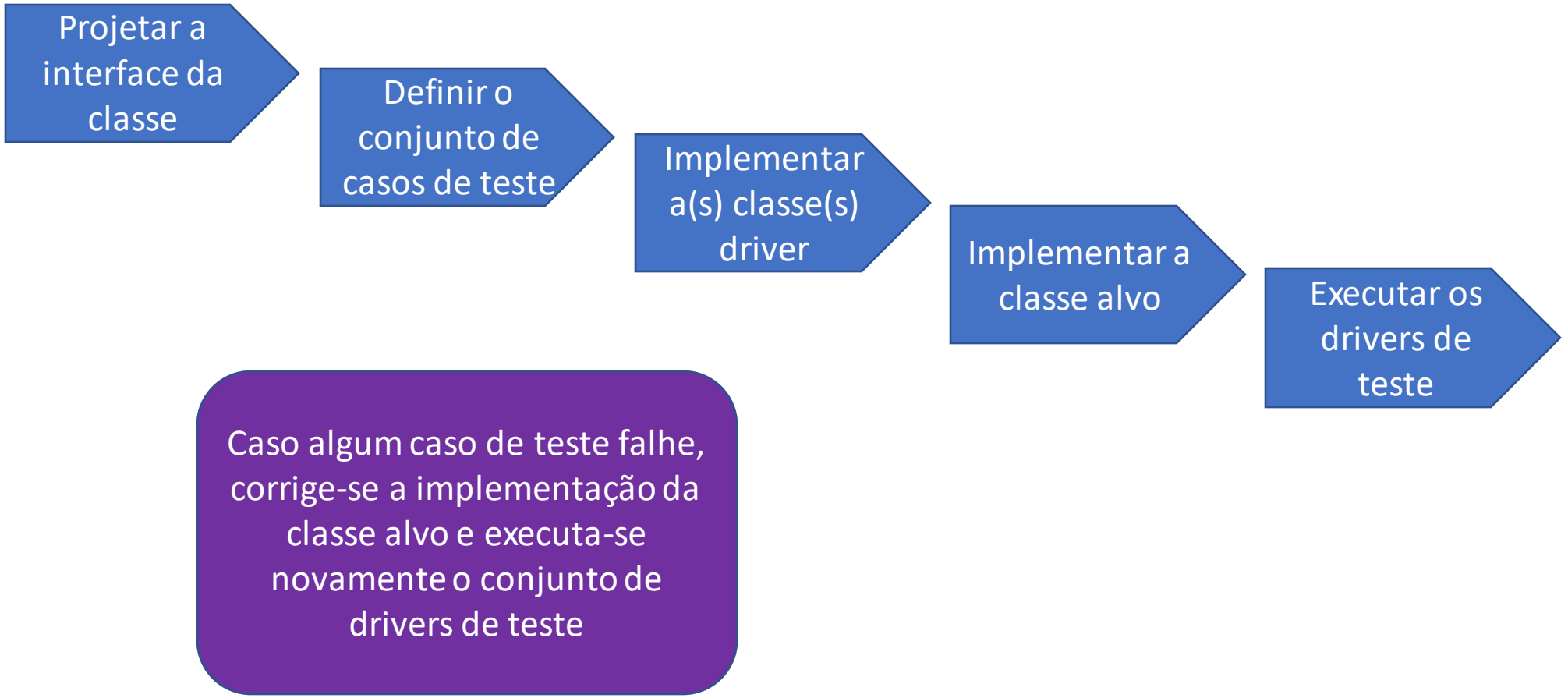
Valore de entrada	Resultado esperado
C	100
CLV	155
CM	900
…	

Exemplo de código de teste

```
public static void main(String args[]){
  int resp;
  RomanNumeral rn = new RomanNumeral();

  resp = rn.convert("C");
  if (resp == 100) System.out.println("teste 1 ok");
  else System.out.println("Teste 1 falhou. Esperado 100 mas obtido "+resp);
  resp = rn.convert("CLV");
  if (resp == 155) System.out.println("teste 2 ok");
  else System.out.println("Teste 2 falhou. Esperado 155 mas obtido "+resp);
  resp = rn.convert("CM");
  if (resp == 900) System.out.println("teste 3 ok");
  else System.out.println("Teste 3 falhou. Esperado 900 mas obtido "+resp);
  ...
}
```

Processo de teste unitário



Processo de Teste Unitário

Vantagens:

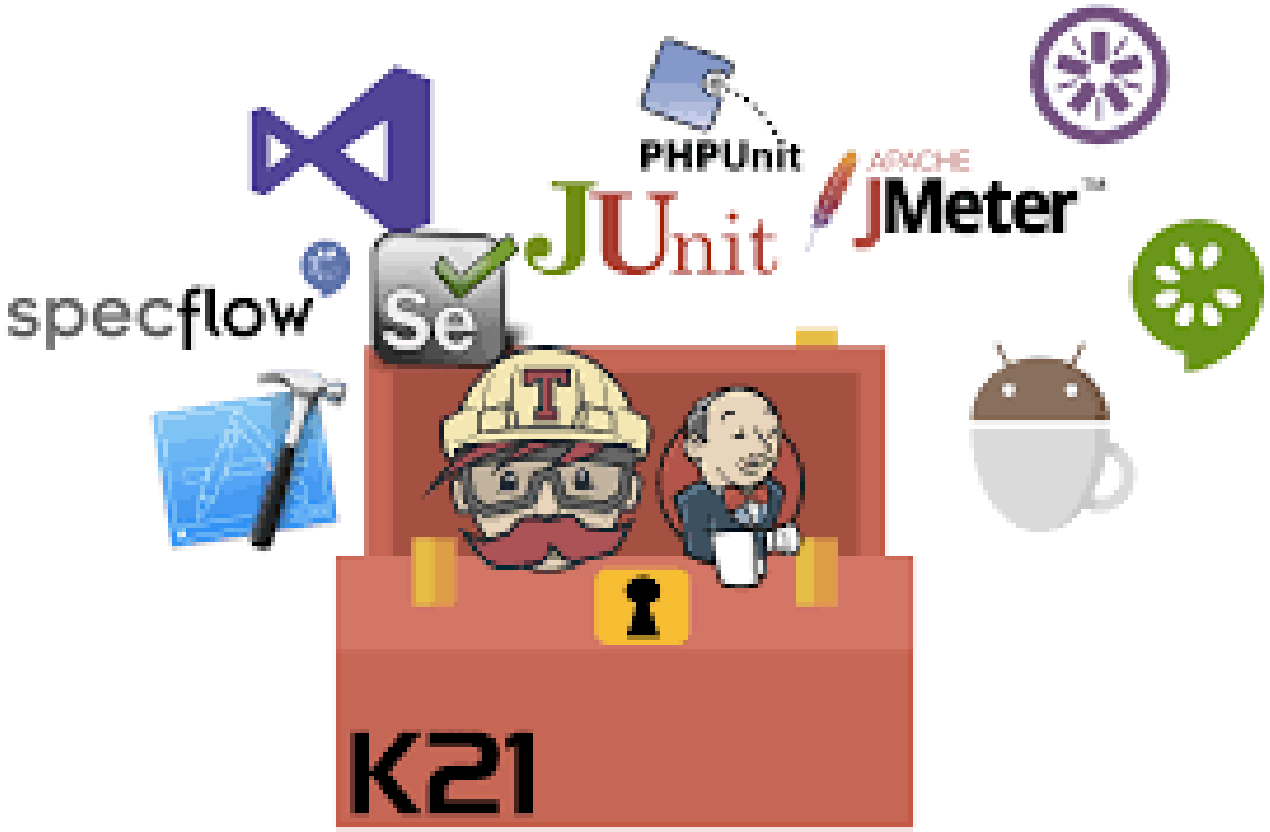
- Exige que se reflita sobre as funcionalidades da classe e sua implementação **antes** de seu desenvolvimento
- Permite a identificação rápida dos defeitos mais simples
- Permite garantir que a classe cumpre um conjunto de requisitos mínimos (os garantidos pelos testes)
- Facilita a detecção de defeitos no caso de manutenção ou refatoração

Desvantagens:

- Necessidade de construção do “cenário” em cada método.
- Dificuldade em se trabalhar com grandes conjuntos de dados de teste
- Dificuldade para coletar os resultados
- Dificuldade para automatizar a execução dos testes

Como Resolver as Desvantagens?

- Utilizar ferramentas de automação de testes
- Facilitam a execução dos *drivers* e a coleta de resultados




1. Explique a Primeira Lei de Newton com suas palavras.
!
Yakka Foob MaG. GRuG PubbaWuP ZiNK wattooM Gazork. CHUMBLE spuzz.
Adoro uma brecha nos regulamentos.